



OpenShift Container Platform 3.10

Configuring Clusters

OpenShift Container Platform 3.10 Installation and Configuration

OpenShift Container Platform 3.10 Configuring Clusters

OpenShift Container Platform 3.10 Installation and Configuration

Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

OpenShift Installation and Configuration topics cover the basics of installing and configuring OpenShift in your environment. Use these topics for the one-time tasks required to get OpenShift up and running.

Table of Contents

CHAPTER 1. OVERVIEW	19
CHAPTER 2. SETTING UP THE REGISTRY	20
2.1. REGISTRY OVERVIEW	20
2.1.1. About the Registry	20
2.1.2. Integrated or Stand-alone Registries	20
2.1.3. Red Hat Quay Registries	20
2.2. DEPLOYING A REGISTRY ON EXISTING CLUSTERS	20
2.2.1. Overview	20
2.2.2. Deploying the Registry	20
2.2.3. Deploying the Registry as a DaemonSet	21
2.2.4. Registry Compute Resources	21
2.2.5. Storage for the Registry	21
2.2.5.1. Production Use	22
2.2.5.1.1. Use Amazon S3 as a Storage Back-end	22
2.2.5.2. Non-Production Use	23
2.2.6. Enabling the Registry Console	24
2.2.6.1. Deploying the Registry Console	24
2.2.6.2. Securing the Registry Console	25
2.2.6.3. Troubleshooting the Registry Console	26
2.2.6.3.1. Debug Mode	26
2.2.6.3.2. Display SSL Certificate Path	27
2.3. ACCESSING THE REGISTRY	27
2.3.1. Viewing Logs	27
2.3.2. File Storage	27
2.3.3. Accessing the Registry Directly	29
2.3.3.1. User Prerequisites	29
2.3.3.2. Logging in to the Registry	30
2.3.3.3. Pushing and Pulling Images	30
2.3.4. Accessing Registry Metrics	31
2.4. SECURING AND EXPOSING THE REGISTRY	33
2.4.1. Overview	33
2.4.2. Manually Securing the Registry	33
2.4.3. Manually Exposing a Secure Registry	36
2.4.4. Manually Exposing a Non-Secure Registry	38
2.5. EXTENDED REGISTRY CONFIGURATION	40
2.5.1. Maintaining the Registry IP Address	40
2.5.2. Whitelisting Docker Registries	40
2.5.3. Setting the Registry Hostname	41
2.5.4. Overriding the Registry Configuration	42
2.5.5. Registry Configuration Reference	43
2.5.5.1. Log	44
2.5.5.2. Hooks	44
2.5.5.3. Storage	44
2.5.5.4. Auth	45
2.5.5.5. Middleware	45
2.5.5.5.1. CloudFront Middleware	46
2.5.5.5.2. Overriding Middleware Configuration Options	47
2.5.5.5.3. Image Pullthrough	48
2.5.5.5.4. Manifest Schema v2 Support	49
2.5.5.6. OpenShift	49

2.5.5.7. Reporting	51
2.5.5.8. HTTP	51
2.5.5.9. Notifications	51
2.5.5.10. Redis	51
2.5.5.11. Health	51
2.5.5.12. Proxy	51
2.5.5.13. Cache	51
2.6. KNOWN ISSUES	52
2.6.1. Overview	52
2.6.2. Concurrent Build with Registry Pull-through	52
2.6.3. Image Push Errors with Scaled Registry Using Shared NFS Volume	52
2.6.4. Pull of Internally Managed Image Fails with "not found" Error	53
2.6.5. Image Push Fails with "500 Internal Server Error" on S3 Storage	53
2.6.6. Image Pruning Fails	54
CHAPTER 3. SETTING UP A ROUTER	55
3.1. ROUTER OVERVIEW	55
3.1.1. About Routers	55
3.1.2. Router Service Account	55
3.1.2.1. Permission to Access Labels	55
3.2. USING THE DEFAULT HAPROXY ROUTER	56
3.2.1. Overview	56
3.2.2. Creating a Router	57
3.2.3. Other Basic Router Commands	57
3.2.4. Filtering Routes to Specific Routers	58
3.2.5. HAProxy Strict SN1	59
3.2.6. TLS Cipher Suites	59
3.2.7. Highly-Available Routers	59
3.2.8. Customizing the Router Service Ports	59
3.2.9. Working With Multiple Routers	60
3.2.10. Adding a Node Selector to a Deployment Configuration	60
3.2.11. Using Router Shards	61
3.2.11.1. Creating Router Shards	63
3.2.11.2. Modifying Router Shards	65
3.2.12. Finding the Host Name of the Router	66
3.2.13. Customizing the Default Routing Subdomain	67
3.2.14. Forcing Route Host Names to a Custom Routing Subdomain	67
3.2.15. Using Wildcard Certificates	68
3.2.16. Manually Redeploy Certificates	68
3.2.17. Using Secured Routes	69
3.2.18. Using Wildcard Routes (for a Subdomain)	71
3.2.19. Using the Container Network Stack	76
3.2.20. Exposing Router Metrics	77
3.2.21. Protecting Against DDoS Attacks	79
3.3. DEPLOYING A CUSTOMIZED HAPROXY ROUTER	80
3.3.1. Overview	80
3.3.2. Obtaining the Router Configuration Template	81
3.3.3. Modifying the Router Configuration Template	81
3.3.3.1. Background	81
3.3.3.2. Go Template Actions	82
3.3.3.3. Router Provided Information	83
3.3.3.4. Annotations	87
3.3.3.5. Environment Variables	88

3.3.3.6. Example Usage	88
3.3.4. Using a ConfigMap to Replace the Router Configuration Template	90
3.3.5. Using Stick Tables	91
3.3.6. Rebuilding Your Router	92
3.4. CONFIGURING THE HAProxy ROUTER TO USE THE PROXY PROTOCOL	93
3.4.1. Overview	93
3.4.2. Why Use the PROXY Protocol?	93
3.4.3. Using the PROXY Protocol	94
3.5. USING THE F5 ROUTER PLUG-IN	97
3.5.1. Overview	98
3.5.2. Prerequisites and Supportability	98
3.5.2.1. Configuring the Virtual Servers	99
3.5.3. Deploying the F5 Router Plug-in	100
3.5.4. F5 Router Plug-in Partition Paths	101
3.5.5. Setting Up F5 Router Plug-in	102
CHAPTER 4. DEPLOYING RED HAT CLOUDFORMS	105
4.1. DEPLOYING RED HAT CLOUDFORMS ON OPENSIFT CONTAINER PLATFORM	105
4.1.1. Introduction	105
4.2. REQUIREMENTS FOR RED HAT CLOUDFORMS ON OPENSIFT CONTAINER PLATFORM	106
4.3. CONFIGURING ROLE VARIABLES	107
4.3.1. Overview	107
4.3.2. General Variables	107
4.3.3. Customizing Template Parameters	108
4.3.4. Database Variables	108
4.3.4.1. Containerized (Podified) Database	108
4.3.4.2. External Database	108
4.3.5. Storage Class Variables	109
4.3.5.1. NFS (Default)	110
4.3.5.2. NFS External	110
4.3.5.3. Cloud Provider	111
4.3.5.4. Preconfigured (Advanced)	111
4.4. RUNNING THE INSTALLER	111
4.4.1. Deploying Red Hat CloudForms During or After OpenShift Container Platform Installation	111
4.4.2. Example Inventory Files	112
4.4.2.1. All Defaults	112
4.4.2.2. External NFS Storage	112
4.4.2.3. Override PV Sizes	112
4.4.2.4. Override Memory Requirements	113
4.4.2.5. External PostgreSQL Database	113
4.5. ENABLING CONTAINER PROVIDER INTEGRATION	113
4.5.1. Adding a Single Container Provider	113
4.5.1.1. Adding Manually	113
4.5.1.2. Adding Automatically	114
4.5.2. Multiple Container Providers	114
4.5.2.1. Preparing the Script	114
4.5.2.1.1. Example	115
4.5.2.2. Running the Playbook	116
4.5.3. Refreshing Providers	116
4.6. UNINSTALLING RED HAT CLOUDFORMS	116
4.6.1. Running the Uninstall Playbook	116
4.6.2. Troubleshooting	116

CHAPTER 5. MASTER AND NODE CONFIGURATION	118
5.1. INSTALLATION DEPENDENCIES	118
5.2. CONFIGURING MASTERS AND NODES	118
5.3. MAKING CONFIGURATION CHANGES USING ANSIBLE	118
5.3.1. Using the <code>htpasswd</code> command	120
5.4. MAKING MANUAL CONFIGURATION CHANGES	121
5.5. MASTER CONFIGURATION FILES	122
5.5.1. Admission Control Configuration	122
5.5.2. Asset Configuration	123
5.5.3. Authentication and Authorization Configuration	124
5.5.4. Controller Configuration	125
5.5.5. <code>etcd</code> Configuration	125
5.5.6. Grant Configuration	127
5.5.7. Image Configuration	127
5.5.8. Image Policy Configuration	128
5.5.9. Kubernetes Master Configuration	128
5.5.10. Network Configuration	129
5.5.11. OAuth Authentication Configuration	131
5.5.12. Project Configuration	133
5.5.13. Scheduler Configuration	134
5.5.14. Security Allocator Configuration	135
5.5.15. Service Account Configuration	135
5.5.16. Serving Information Configuration	136
5.5.17. Volume Configuration	137
5.5.18. Basic Audit	138
5.5.19. Advanced Audit	139
5.5.20. Specifying TLS ciphers for <code>etcd</code>	142
5.6. NODE CONFIGURATION FILES	144
5.6.1. Pod and Node Configuration	145
5.6.2. Docker Configuration	145
5.6.3. Local Storage Configuration	146
5.6.4. Parallel Image Pulls with Docker 1.9+	146
5.7. PASSWORDS AND OTHER SENSITIVE DATA	147
5.8. CREATING NEW CONFIGURATION FILES	147
5.9. LAUNCHING SERVERS USING CONFIGURATION FILES	148
5.10. CONFIGURING LOGGING LEVELS	149
5.11. RESTARTING MASTER AND NODE SERVICES	153
CHAPTER 6. OPENSIFT ANSIBLE BROKER CONFIGURATION	154
6.1. OVERVIEW	154
6.2. MODIFYING THE OPENSIFT ANSIBLE BROKER CONFIGURATION	155
6.3. REGISTRY CONFIGURATION	155
6.3.1. Production or Development	156
6.3.2. Storing Registry Credentials	157
6.3.3. Mock Registry	159
6.3.4. Dockerhub Registry	159
6.3.5. APB Filtering	159
6.3.6. Local OpenShift Container Registry	160
6.3.7. Red Hat Container Catalog Registry	161
6.3.8. Red Hat Connect Partner Registry	161
6.3.9. Multiple Registries	161
6.4. BROKER AUTHENTICATION	162
6.4.1. Basic Auth	162

6.4.1.1. Deployment Template and Secrets	162
6.4.1.2. Configuring Service Catalog and Broker Communication	163
6.4.2. Bearer Auth	164
6.4.2.1. Deployment Template and Secrets	165
6.4.2.2. Configuring Service Catalog and Broker Communication	165
6.5. DAO CONFIGURATION	166
6.6. LOG CONFIGURATION	166
6.7. OPENSIFT CONFIGURATION	166
6.8. BROKER CONFIGURATION	167
6.9. SECRETS CONFIGURATION	168
6.10. RUNNING BEHIND A PROXY	168
6.10.1. Registry Adapter Whitelists	168
6.10.2. Configuring the Broker Behind a Proxy Using Ansible	169
6.10.3. Configuring the Broker Behind a Proxy Manually	169
6.10.4. Setting Proxy Environment Variables in Pods	170
CHAPTER 7. ADDING HOSTS TO AN EXISTING CLUSTER	171
7.1. ADDING HOSTS	171
Procedure	171
7.2. ADDING ETCD HOSTS TO EXISTING CLUSTER	173
7.3. REPLACING EXISTING MASTERS WITH ETCD COLOCATED	174
7.4. MIGRATING THE NODES	176
CHAPTER 8. ADDING THE DEFAULT IMAGE STREAMS AND TEMPLATES	177
8.1. OVERVIEW	177
8.2. OFFERINGS BY SUBSCRIPTION TYPE	177
8.2.1. OpenShift Container Platform Subscription	177
8.2.2. xPaaS Middleware Add-on Subscriptions	178
8.3. BEFORE YOU BEGIN	178
8.4. PREREQUISITES	178
8.5. CREATING IMAGE STREAMS FOR OPENSIFT CONTAINER PLATFORM IMAGES	179
8.6. CREATING IMAGE STREAMS FOR XPAAS MIDDLEWARE IMAGES	179
8.7. CREATING DATABASE SERVICE TEMPLATES	180
8.8. CREATING INSTANT APP AND QUICKSTART TEMPLATES	180
8.9. WHAT'S NEXT?	181
CHAPTER 9. CONFIGURING CUSTOM CERTIFICATES	182
9.1. OVERVIEW	182
9.2. CONFIGURING CUSTOM CERTIFICATES DURING INSTALLATION	182
9.3. CONFIGURING CUSTOM CERTIFICATES FOR THE WEB CONSOLE OR CLI	182
9.4. CONFIGURING A CUSTOM MASTER HOST CERTIFICATE	184
9.5. CONFIGURING A CUSTOM WILDCARD CERTIFICATE FOR THE DEFAULT ROUTER	185
9.6. CONFIGURING A CUSTOM CERTIFICATE FOR THE IMAGE REGISTRY	186
9.7. CONFIGURING A CUSTOM CERTIFICATE FOR A LOAD BALANCER	187
9.8. RETROFIT CUSTOM CERTIFICATES INTO A CLUSTER	188
9.8.1. Retrofit Custom Master Certificates into a Cluster	188
9.8.2. Retrofit Custom Router Certificates into a Cluster	188
9.9. USING CUSTOM CERTIFICATES WITH OTHER COMPONENTS	189
CHAPTER 10. REDEPLOYING CERTIFICATES	190
10.1. OVERVIEW	190
10.2. CHECKING CERTIFICATE EXPIRATIONS	190
10.2.1. Role Variables	190
10.2.2. Running Certificate Expiration Playbooks	191

Other Example Playbooks	192
10.2.3. Output Formats	192
HTML Report	192
JSON Report	193
10.3. REDEPLOYING CERTIFICATES	193
10.3.1. Redeploying All Certificates Using the Current OpenShift Container Platform and etcd CA	194
10.3.2. Redeploying a New or Custom OpenShift Container Platform CA	194
10.3.3. Redeploying a New etcd CA	196
10.3.4. Redeploying Master Certificates Only	196
10.3.5. Redeploying etcd Certificates Only	196
10.3.6. Redeploying Node Certificates Only	197
10.3.7. Redeploying Registry or Router Certificates Only	197
10.3.7.1. Redeploying Registry Certificates Only	197
10.3.7.2. Redeploying Router Certificates Only	197
10.3.8. Redeploying Custom Registry or Router Certificates	197
10.3.8.1. Redeploying Registry Certificates Manually	198
10.3.8.2. Redeploying Router Certificates Manually	199
CHAPTER 11. CONFIGURING AUTHENTICATION AND USER AGENT	202
11.1. OVERVIEW	202
11.2. IDENTITY PROVIDER PARAMETERS	202
11.3. CONFIGURING IDENTITY PROVIDERS	203
11.3.1. Configuring identity providers with Ansible	203
11.3.2. Configuring identity providers in the master configuration file	204
11.3.3. Configuring an identity provider or method	205
11.3.3.1. Manually provisioning a user when using the lookup mapping method	205
11.3.4. Allow all	205
11.3.5. Deny all	206
11.3.6. HTTPasswd	207
11.3.7. Keystone	208
11.3.7.1. Configuring authentication on the master	208
11.3.7.2. Creating Users with Keystone Authentication	210
11.3.7.3. Verifying Users	210
11.3.8. LDAP authentication	211
11.3.9. Basic authentication (remote)	214
11.3.9.1. Configuring authentication on the master	215
11.3.9.2. Troubleshooting	216
11.3.10. Request header	217
Apache authentication using Request header	220
Installing the prerequisites	220
Configuring Apache	221
Additional mod_auth_form requirements	223
Creating users	223
Configuring the master	223
Restarting services	224
Verifying the configuration	224
11.3.11. GitHub	225
11.3.11.1. Registering the application on GitHub	225
11.3.11.2. Configuring authentication on the master	225
11.3.11.3. Creating users with GitHub authentication	227
11.3.11.4. Verifying users	228
11.3.12. GitLab	228
11.3.13. Google	229

11.3.14. OpenID connect	230
11.4. TOKEN OPTIONS	233
11.5. GRANT OPTIONS	234
11.6. SESSION OPTIONS	234
11.7. PREVENTING CLI VERSION MISMATCH WITH USER AGENT	235
CHAPTER 12. SYNCING GROUPS WITH LDAP	238
12.1. OVERVIEW	238
12.2. CONFIGURING LDAP SYNC	238
12.2.1. LDAP client configuration	238
12.2.2. LDAP query definition	239
12.2.3. User-defined name mapping	240
12.3. RUNNING LDAP SYNC	240
12.4. RUNNING A GROUP PRUNING JOB	241
12.5. SYNC EXAMPLES	241
12.5.1. Syncing groups by using RFC 2307 schema	241
12.5.1.1. RFC2307 with user-defined name mappings	244
12.5.2. Syncing groups by using RFC 2307 with user-defined error tolerances	245
12.5.3. Syncing groups by using Active Directory	248
12.5.4. Syncing groups by using augmented Active Directory	250
12.6. NESTED MEMBERSHIP SYNC EXAMPLE	252
12.7. LDAP SYNC CONFIGURATION SPECIFICATION	256
12.7.1. v1.LDAPSyncConfig	256
12.7.2. v1.StringSource	258
12.7.3. v1.LDAPQuery	258
12.7.4. v1.RFC2307Config	259
12.7.5. v1.ActiveDirectoryConfig	261
12.7.6. v1.AugmentedActiveDirectoryConfig	262
CHAPTER 13. CONFIGURING LDAP FAILOVER	264
13.1. PREREQUISITES FOR CONFIGURING BASIC REMOTE AUTHENTICATION	264
13.2. GENERATING AND SHARING CERTIFICATES WITH THE REMOTE BASIC AUTHENTICATION SERVER	264
13.3. CONFIGURING SSSD FOR LDAP FAILOVER	265
13.4. CONFIGURING APACHE TO USE SSSD	267
13.5. CONFIGURING OPENSIFT CONTAINER PLATFORM TO USE SSSD AS THE BASIC REMOTE AUTHENTICATION SERVER	270
CHAPTER 14. CONFIGURING THE SDN	272
14.1. OVERVIEW	272
14.2. AVAILABLE SDN PROVIDERS	272
Installing VMware NSX-T (™) on OpenShift Container Platform	272
14.3. CONFIGURING THE POD NETWORK WITH ANSIBLE	272
14.4. CONFIGURING THE POD NETWORK ON MASTERS	273
14.5. CONFIGURING THE POD NETWORK ON NODES	274
14.6. EXPANDING THE SERVICE NETWORK	274
14.7. MIGRATING BETWEEN SDN PLUG-INS	276
14.7.1. Migrating from ovs-multitenant to ovs-networkpolicy	277
14.8. EXTERNAL ACCESS TO THE CLUSTER NETWORK	277
14.9. USING FLANNEL	278
CHAPTER 15. CONFIGURING NUAGE SDN	281
15.1. NUAGE SDN AND OPENSIFT CONTAINER PLATFORM	281
15.2. DEVELOPER WORKFLOW	281

15.3. OPERATIONS WORKFLOW	281
15.4. INSTALLATION	281
CHAPTER 16. CONFIGURING KURYR SDN	284
16.1. KURYR SDN AND OPENSIFT CONTAINER PLATFORM	284
16.2. INSTALLATION	284
16.3. VERIFICATION	286
CHAPTER 17. CONFIGURING FOR AMAZON WEB SERVICES (AWS)	287
17.1. OVERVIEW	287
17.2. PERMISSIONS	287
17.3. CONFIGURING A SECURITY GROUP	288
17.3.1. Overriding Detected IP Addresses and Host Names	289
17.4. CONFIGURING AWS VARIABLES	289
17.5. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR AWS	290
17.5.1. Configuring OpenShift Container Platform for AWS with Ansible	290
17.5.2. Manually Configuring OpenShift Container Platform Masters for AWS	291
17.5.3. Manually Configuring OpenShift Container Platform Nodes for AWS	291
17.5.4. Manually Setting Key-Value Access Pairs	291
17.6. APPLYING CONFIGURATION CHANGES	292
17.7. LABELING CLUSTERS FOR AWS	292
17.7.1. Resources That Need Tags	293
17.7.2. Tagging an Existing Cluster	293
CHAPTER 18. CONFIGURING FOR RED HAT VIRTUALIZATION	294
18.1. CONFIGURING RED HAT VIRTUALIZATION OBJECTS	294
18.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR RED HAT VIRTUALIZATION	295
CHAPTER 19. CONFIGURING FOR OPENSTACK	298
19.1. OVERVIEW	298
19.2. PERMISSIONS	298
19.3. CONFIGURING A SECURITY GROUP	298
19.4. CONFIGURING OPENSTACK VARIABLES	299
19.5. CONFIGURING OPENSIFT CONTAINER PLATFORM MASTERS FOR OPENSTACK	299
19.5.1. Configuring OpenShift Container Platform for OpenStack with Ansible	299
19.5.2. Manually Configuring OpenShift Container Platform Masters for OpenStack	300
19.5.3. Manually Configuring OpenShift Container Platform Nodes for OpenStack	301
19.5.4. Installing OpenShift Container Platform by Using an Ansible Playbook	301
19.6. APPLYING CONFIGURATION CHANGES	302
19.6.1. Configuring Zone Labels for Dynamically Created OpenStack PVs	302
CHAPTER 20. CONFIGURING FOR GOOGLE COMPUTE ENGINE	304
20.1. BEFORE YOU BEGIN	304
20.1.1. Configuring authorization for Google Cloud Platform	304
20.1.2. Google Compute Engine objects	305
20.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR GCE	308
20.2.1. Option 1: Configuring OpenShift Container Platform for GCP using Ansible	308
20.2.2. Option 2: Manually configuring OpenShift Container Platform for GCE	310
20.2.2.1. Manually configuring master hosts for GCE	310
20.2.2.2. Manually configuring node hosts for GCE	311
20.2.3. Configuring the OpenShift Container Platform registry for GCP	312
20.2.3.1. Manually configuring OpenShift Container Platform registry for GCP	313
20.2.3.1.1. Verify the registry is using GCP object storage	313
20.2.4. Configuring OpenShift Container Platform to use GCP storage	316

20.3. USING THE GCP EXTERNAL LOAD BALANCER AS A SERVICE	317
CHAPTER 21. CONFIGURING FOR AZURE	319
21.1. BEFORE YOU BEGIN	319
21.1.1. Configuring authorization for Microsoft Azure	319
21.1.2. Configuring Microsoft Azure objects	320
21.2. THE AZURE CONFIGURATION FILE	321
21.3. EXAMPLE INVENTORY FOR OPENSIFT CONTAINER PLATFORM ON MICROSOFT AZURE	322
21.4. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR MICROSOFT AZURE	325
21.4.1. Configuring OpenShift Container Platform for Azure using Ansible	325
21.4.2. Manually configuring OpenShift Container Platform for Microsoft Azure	326
21.4.2.1. Manually configuring master hosts for Microsoft Azure	326
21.4.2.2. Manually configuring node hosts for Microsoft Azure	327
21.4.3. Configuring the OpenShift Container Platform registry for Microsoft Azure	328
21.4.4. Configuring OpenShift Container Platform to use Microsoft Azure storage	332
21.5. USING THE MICROSOFT AZURE EXTERNAL LOAD BALANCER AS A SERVICE	333
21.5.1. Deploying a sample application using a load balancer	334
CHAPTER 22. CONFIGURING FOR VMWARE VSPHERE	336
22.1. BEFORE YOU BEGIN	336
22.1.1. VMware vSphere cloud provider prerequisites	336
22.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR VSPHERE	338
22.2.1. Option 1: Configuring OpenShift Container Platform for vSphere using Ansible	338
22.2.2. Option 2: Manually configuring OpenShift Container Platform for vSphere	342
22.2.2.1. Manually configuring master hosts for vSphere	342
22.2.2.2. Manually configuring node hosts for vSphere	344
22.2.2.3. Applying Configuration Changes	345
22.2.3. Configuring OpenShift Container Platform to use vSphere storage	345
Prerequisites	346
22.2.3.1. Provisioning VMware vSphere volumes	346
22.2.3.1.1. Creating persistent volumes	346
22.2.3.1.2. Formatting VMware vSphere volumes	347
22.2.3.2. Provisioning VMware vSphere volumes via a Storage Class	347
22.2.4. Configuring the OpenShift Container Platform registry for vSphere	348
22.2.4.1. Configuring the OpenShift Container Platform registry for vSphere using Ansible	348
22.2.4.2. Manually configuring OpenShift Container Platform registry for vSphere	349
22.3. BACKUP OF PERSISTENT VOLUMES	350
CHAPTER 23. CONFIGURING LOCAL VOLUMES	351
23.1. OVERVIEW	351
23.2. MOUNTING LOCAL VOLUMES	351
23.3. CONFIGURING THE LOCAL PROVISIONER	352
23.4. DEPLOYING THE LOCAL PROVISIONER	353
23.5. ADDING NEW DEVICES	354
23.6. CONFIGURING RAW BLOCK DEVICES	354
23.6.1. Preparing raw block devices	355
23.6.2. Deploying raw block device provisioners	356
23.6.3. Using raw block device persistent volumes	357
CHAPTER 24. CONFIGURING PERSISTENT STORAGE	359
24.1. OVERVIEW	359
24.2. PERSISTENT STORAGE USING NFS	359
24.2.1. Overview	359
24.2.2. Provisioning	360

24.2.3. Enforcing Disk Quotas	361
24.2.4. NFS Volume Security	361
24.2.4.1. Group IDs	362
24.2.4.2. User IDs	363
24.2.4.3. SELinux	364
24.2.4.4. Export Settings	364
24.2.5. Reclaiming Resources	365
24.2.6. Automation	366
24.2.7. Additional Configuration and Troubleshooting	366
24.3. PERSISTENT STORAGE USING RED HAT GLUSTER STORAGE	366
24.3.1. Overview	367
24.3.1.1. converged mode	367
24.3.1.2. independent mode	367
24.3.1.3. Standalone Red Hat Gluster Storage	367
24.3.1.4. GlusterFS Volumes	368
24.3.1.5. gluster-block Volumes	368
24.3.1.6. Gluster S3 Storage	369
24.3.2. Considerations	369
24.3.2.1. Software Prerequisites	369
24.3.2.2. Hardware Requirements	369
24.3.2.3. Storage Sizing	370
24.3.2.4. Volume Operation Behaviors	371
24.3.2.5. Volume Security	371
24.3.2.5.1. POSIX Permissions	371
24.3.2.5.2. SELinux	372
24.3.3. Support Requirements	372
24.3.4. Installation	373
24.3.4.1. independent mode: Installing Red Hat Gluster Storage Nodes	373
24.3.4.2. Using the Installer	373
24.3.4.2.1. Example: Basic converged mode Installation	375
24.3.4.2.2. Example: Basic independent mode Installation	376
24.3.4.2.3. Example: converged mode with an Integrated OpenShift Container Registry	378
24.3.4.2.4. Example: converged mode for OpenShift Logging and Metrics	379
24.3.4.2.5. Example: converged mode for Applications, Registry, Logging, and Metrics	381
24.3.4.2.6. Example: independent mode for Applications, Registry, Logging, and Metrics	383
24.3.5. Uninstall converged mode	386
24.3.6. Provisioning	387
24.3.6.1. Static Provisioning	387
24.3.6.2. Dynamic Provisioning	390
24.4. PERSISTENT STORAGE USING OPENSTACK CINDER	391
24.4.1. Overview	391
24.4.2. Provisioning Cinder PVs	391
24.4.2.1. Creating the Persistent Volume	391
24.4.2.2. Cinder PV format	393
24.4.2.3. Cinder volume security	393
24.5. PERSISTENT STORAGE USING CEPH RADOS BLOCK DEVICE (RBD)	394
24.5.1. Overview	394
24.5.2. Provisioning	394
24.5.2.1. Creating the Ceph Secret	395
24.5.2.2. Creating the Persistent Volume	395
24.5.3. Ceph Volume Security	397
24.6. PERSISTENT STORAGE USING AWS ELASTIC BLOCK STORE	398
24.6.1. Overview	398

24.6.2. Provisioning	398
24.6.2.1. Creating the Persistent Volume	398
24.6.2.2. Volume Format	400
24.6.2.3. Maximum Number of EBS Volumes on a Node	400
24.7. PERSISTENT STORAGE USING GCE PERSISTENT DISK	400
24.7.1. Overview	400
24.7.2. Provisioning	400
24.7.2.1. Creating the Persistent Volume	401
24.7.2.2. Volume Format	402
24.8. PERSISTENT STORAGE USING ISCSI	402
24.8.1. Overview	402
24.8.2. Provisioning	402
24.8.2.1. Enforcing Disk Quotas	403
24.8.2.2. iSCSI Volume Security	403
24.8.2.3. iSCSI Multipathing	403
24.8.2.4. iSCSI Custom Initiator IQN	404
24.9. PERSISTENT STORAGE USING FIBRE CHANNEL	404
24.9.1. Overview	404
24.9.2. Provisioning	405
24.9.2.1. Enforcing Disk Quotas	405
24.9.2.2. Fibre Channel Volume Security	405
24.10. PERSISTENT STORAGE USING AZURE DISK	406
24.10.1. Overview	406
24.10.2. Prerequisites	406
24.10.3. Provisioning	406
24.10.4. Configuring Azure Disk for regional cloud	406
24.10.4.1. Creating the Persistent Volume	407
24.10.4.2. Volume Format	408
24.11. PERSISTENT STORAGE USING AZURE FILE	408
24.11.1. Overview	408
24.11.2. Before you begin	409
24.11.3. Configuring Azure File for regional cloud	410
24.11.4. Creating the PV	410
24.11.5. Creating the Azure Storage Account secret	410
24.12. PERSISTENT STORAGE USING FLEXVOLUME PLUG-INS	412
24.12.1. Overview	412
24.12.2. FlexVolume drivers	412
24.12.2.1. FlexVolume drivers with master-initiated attach/detach	413
24.12.2.2. FlexVolume drivers without master-initiated attach/detach	416
24.12.3. Installing FlexVolume drivers	416
24.12.4. Consuming storage using FlexVolume drivers	417
24.13. USING VMWARE VSPHERE VOLUMES FOR PERSISTENT STORAGE	418
24.13.1. Overview	418
Prerequisites	418
24.13.2. Provisioning VMware vSphere volumes	419
24.13.2.1. Creating persistent volumes	419
24.13.2.2. Formatting VMware vSphere volumes	420
24.14. PERSISTENT STORAGE USING LOCAL VOLUME	420
24.14.1. Overview	420
24.14.2. Provisioning	421
24.14.3. Creating Local Persistent Volume Claim	421
24.14.4. Feature Status	421
24.15. PERSISTENT STORAGE USING CONTAINER STORAGE INTERFACE (CSI)	422

24.15.1. Overview	422
24.15.2. Architecture	422
24.15.2.1. External CSI Controllers	423
24.15.2.2. CSI Driver DaemonSet	424
24.15.3. Example Deployment	424
24.15.4. Dynamic Provisioning	429
24.15.5. Usage	429
24.16. PERSISTENT STORAGE USING OPENSTACK MANILA	429
24.16.1. Overview	429
24.16.2. Installation and Setup	430
24.16.2.1. Starting the External Provisioner	430
24.16.3. Usage	433
24.17. DYNAMIC PROVISIONING AND CREATING STORAGE CLASSES	433
24.17.1. Overview	433
24.17.2. Available dynamically provisioned plug-ins	434
24.17.3. Defining a StorageClass	435
24.17.3.1. Basic StorageClass object definition	435
24.17.3.2. StorageClass annotations	436
24.17.3.3. OpenStack Cinder object definition	436
24.17.3.4. AWS ElasticBlockStore (EBS) object definition	437
24.17.3.5. GCE PersistentDisk (gcePD) object definition	438
24.17.3.6. GlusterFS object definition	438
24.17.3.7. Ceph RBD object definition	439
24.17.3.8. Trident object definition	440
24.17.3.9. VMware vSphere object definition	441
24.17.3.10. Azure File object definition	441
24.17.3.11. Azure Disk object definition	442
24.17.4. Changing the default StorageClass	443
24.17.5. Additional information and examples	444
24.18. VOLUME SECURITY	444
24.18.1. Overview	444
24.18.2. SCCs, Defaults, and Allowed Ranges	444
24.18.3. Supplemental Groups	448
24.18.4. fsGroup	451
24.18.5. User IDs	453
24.18.6. SELinux Options	455
24.19. SELECTOR-LABEL VOLUME BINDING	456
24.19.1. Overview	456
24.19.2. Motivation	456
24.19.3. Deployment	457
24.19.3.1. Prerequisites	457
24.19.3.2. Define the Persistent Volume and Claim	457
24.19.3.3. Deploy the Persistent Volume and Claim	458
24.20. ENABLING CONTROLLER-MANAGED ATTACHMENT AND DETACHMENT	459
24.20.1. Overview	459
24.20.2. Determining What Is Managing Attachment and Detachment	459
24.20.3. Configuring Nodes to Enable Controller-managed Attachment and Detachment	459
24.21. PERSISTENT VOLUME SNAPSHOTS	460
24.21.1. Overview	460
24.21.2. Features	460
24.21.3. Installation and Setup	460
24.21.3.1. Starting the External Controller and Provisioner	460
24.21.3.2. Managing Snapshot Users	463

24.21.4. Lifecycle of a Volume Snapshot and Volume Snapshot Data	464
24.21.4.1. Persistent Volume Claim and Persistent Volume	464
24.21.4.1.1. Snapshot Promoter	464
24.21.4.2. Create Snapshot	465
24.21.4.3. Restore Snapshot	466
24.21.4.4. Delete Snapshot	466
CHAPTER 25. PERSISTENT STORAGE EXAMPLES	467
25.1. OVERVIEW	467
25.2. SHARING AN NFS MOUNT ACROSS TWO PERSISTENT VOLUME CLAIMS	467
25.2.1. Overview	467
25.2.2. Creating the Persistent Volume	467
25.2.3. Creating the Persistent Volume Claim	468
25.2.4. Ensuring NFS Volume Access	469
25.2.5. Creating the Pod	470
25.2.6. Creating an Additional Pod to Reference the Same PVC	474
25.3. COMPLETE EXAMPLE USING CEPH RBD	476
25.3.1. Overview	476
25.3.2. Installing the ceph-common Package	476
25.3.3. Creating the Ceph Secret	476
25.3.4. Creating the Persistent Volume	477
25.3.5. Creating the Persistent Volume Claim	478
25.3.6. Creating the Pod	479
25.3.7. Defining Group and Owner IDs (Optional)	480
25.3.8. Setting ceph-user-secret as Default for Projects	480
25.4. USING CEPH RBD FOR DYNAMIC PROVISIONING	481
25.4.1. Overview	481
25.4.2. Creating a pool for dynamic volumes	481
25.4.3. Using an existing Ceph cluster for dynamic persistent storage	482
25.4.4. Setting ceph-user-secret as the default for projects	485
25.5. COMPLETE EXAMPLE USING GLUSTERFS	486
25.5.1. Overview	486
25.5.2. Prerequisites	486
25.5.3. Static Provisioning	487
25.5.4. Using the Storage	490
25.6. COMPLETE EXAMPLE USING GLUSTERFS FOR DYNAMIC PROVISIONING	492
25.6.1. Overview	492
25.6.2. Prerequisites	492
25.6.3. Dynamic Provisioning	493
25.6.4. Using the Storage	494
25.7. MOUNTING VOLUMES ON PRIVILEGED PODS	496
25.7.1. Overview	496
25.7.2. Prerequisites	496
25.7.3. Creating the Persistent Volume	496
25.7.4. Creating a Regular User	497
25.7.5. Creating the Persistent Volume Claim	497
25.7.6. Verifying the Setup	498
25.7.6.1. Checking the Pod SCC	498
25.7.6.2. Verifying the Mount	499
25.8. SWITCHING AN INTEGRATED OPENSIFT CONTAINER REGISTRY TO GLUSTERFS	499
25.8.1. Overview	499
25.8.2. Prerequisites	499
25.8.3. Manually Provision the GlusterFS PersistentVolumeClaim	499

25.8.4. Attach the PersistentVolumeClaim to the Registry	503
25.9. BINDING PERSISTENT VOLUMES BY LABELS	503
25.9.1. Overview	503
25.9.1.1. Assumptions	504
25.9.2. Defining Specifications	504
25.9.2.1. Persistent Volume with Labels	504
25.9.2.2. Persistent Volume Claim with Selectors	504
25.9.2.3. Volume Endpoints	505
25.9.2.4. Deploy the PV, PVC, and Endpoints	505
25.10. USING STORAGE CLASSES FOR DYNAMIC PROVISIONING	506
25.10.1. Overview	506
25.10.2. Scenario 1: Basic Dynamic Provisioning with Two Types of StorageClasses	506
25.10.3. Scenario 2: How to enable Default StorageClass behavior for a Cluster	509
25.11. USING STORAGE CLASSES FOR EXISTING LEGACY STORAGE	513
25.11.1. Overview	513
25.11.1.1. Scenario 1: Link StorageClass to existing Persistent Volume with Legacy Data	513
25.12. CONFIGURING AZURE BLOB STORAGE FOR INTEGRATED DOCKER REGISTRY	516
25.12.1. Overview	516
25.12.2. Before You Begin	516
25.12.3. Overriding Registry Configuration	516
CHAPTER 26. CONFIGURING EPHEMERAL STORAGE	518
26.1. OVERVIEW	518
26.2. ENABLING EPHEMERAL STORAGE	518
CHAPTER 27. WORKING WITH HTTP PROXIES	520
27.1. OVERVIEW	520
27.2. CONFIGURING NO_PROXY	520
27.3. CONFIGURING HOSTS FOR PROXIES	521
27.4. CONFIGURING HOSTS FOR PROXIES USING ANSIBLE	521
27.5. PROXYING DOCKER PULL	522
27.6. USING MAVEN BEHIND A PROXY	523
27.7. CONFIGURING S2I BUILDS FOR PROXIES	523
27.8. CONFIGURING DEFAULT TEMPLATES FOR PROXIES	523
27.9. SETTING PROXY ENVIRONMENT VARIABLES IN PODS	524
27.10. GIT REPOSITORY ACCESS	524
CHAPTER 28. CONFIGURING GLOBAL BUILD DEFAULTS AND OVERRIDES	525
28.1. OVERVIEW	525
28.2. SETTING GLOBAL BUILD DEFAULTS	525
28.2.1. Configuring Global Build Defaults with Ansible	526
28.2.2. Manually Setting Global Build Defaults	527
28.3. SETTING GLOBAL BUILD OVERRIDES	528
28.3.1. Configuring Global Build Overrides with Ansible	528
28.3.2. Manually Setting Global Build Overrides	529
CHAPTER 29. CONFIGURING PIPELINE EXECUTION	531
29.1. OVERVIEW	531
29.2. OPENSIFT JENKINS CLIENT PLUGIN	532
29.3. OPENSIFT JENKINS SYNC PLUGIN	532
CHAPTER 30. CONFIGURING ROUTE TIMEOUTS	534
CHAPTER 31. CONFIGURING NATIVE CONTAINER ROUTING	535
31.1. NETWORK OVERVIEW	535

31.2. CONFIGURE NATIVE CONTAINER ROUTING	535
31.3. SETTING UP A NODE FOR CONTAINER NETWORKING	536
31.4. SETTING UP A ROUTER FOR CONTAINER NETWORKING	536
CHAPTER 32. ROUTING FROM EDGE LOAD BALANCERS	537
32.1. OVERVIEW	537
32.2. INCLUDING THE LOAD BALANCER IN THE SDN	537
32.3. ESTABLISHING A TUNNEL USING A RAMP NODE	537
32.3.1. Configuring a Highly-Available Ramp Node	540
CHAPTER 33. AGGREGATING CONTAINER LOGS	541
33.1. OVERVIEW	541
33.2. PRE-DEPLOYMENT CONFIGURATION	541
33.3. SPECIFYING LOGGING ANSIBLE VARIABLES	542
33.4. DEPLOYING THE EFK STACK	551
33.5. UNDERSTANDING AND ADJUSTING THE DEPLOYMENT	551
33.5.1. Ops Cluster	552
33.5.2. Elasticsearch	552
33.5.3. Fluentd	556
33.5.4. Kibana	560
33.5.5. Curator	561
33.5.5.1. Creating the Curator Configuration	563
33.6. CLEANUP	564
33.7. TROUBLESHOOTING KIBANA	564
33.8. SENDING LOGS TO AN EXTERNAL ELASTICSEARCH INSTANCE	566
33.9. SENDING LOGS TO AN EXTERNAL SYSLOG SERVER	566
33.10. PERFORMING ADMINISTRATIVE ELASTICSEARCH OPERATIONS	569
33.11. CHANGING THE AGGREGATED LOGGING DRIVER	570
33.12. MANUAL ELASTICSEARCH ROLLOUTS	572
33.12.1. Performing an Elasticsearch Rolling Cluster Restart	572
33.12.2. Performing an Elasticsearch Full Cluster Restart	573
CHAPTER 34. AGGREGATE LOGGING SIZING GUIDELINES	575
34.1. OVERVIEW	575
34.2. INSTALLATION	575
34.2.1. Large Clusters	577
34.3. SYSTEMD-JOURNALD AND RSYSLOG	577
34.4. SCALING UP EFK LOGGING	578
34.5. STORAGE CONSIDERATIONS	579
CHAPTER 35. ENABLING CLUSTER METRICS	581
35.1. OVERVIEW	581
35.2. BEFORE YOU BEGIN	581
35.3. METRICS PROJECT	581
35.4. METRICS DATA STORAGE	581
35.4.1. Persistent Storage	582
35.4.2. Capacity Planning for Cluster Metrics	582
Known Issues and Limitations	584
35.4.3. Non-Persistent Storage	584
35.5. METRICS ANSIBLE ROLE	584
35.5.1. Specifying Metrics Ansible Variables	585
35.5.2. Using Secrets	588
35.5.2.1. Providing Your Own Certificates	588
35.6. DEPLOYING THE METRIC COMPONENTS	588

35.6.1. Metrics Diagnostics	589
35.7. SETTING THE METRICS PUBLIC URL	590
35.8. ACCESSING HAWKULAR METRICS DIRECTLY	590
35.8.1. OpenShift Container Platform Projects and Hawkular Tenants	591
35.8.2. Authorization	591
35.9. SCALING OPENSIFT CONTAINER PLATFORM CLUSTER METRICS PODS	591
35.10. CLEANUP	591
35.11. PROMETHEUS ON OPENSIFT CONTAINER PLATFORM	591
35.11.1. Setting Prometheus Role Variables	592
35.11.2. Deploying Prometheus Using Ansible Installer	593
35.11.2.1. Additional Methods for Deploying Prometheus	594
35.11.2.2. Accessing the Prometheus Web UI	594
35.11.2.3. Configuring Prometheus for OpenShift Container Platform	594
35.11.3. OpenShift Container Platform Metrics via Prometheus	595
35.11.3.1. Current Metrics	595
35.11.4. Undeploying Prometheus	598
CHAPTER 36. CUSTOMIZING THE WEB CONSOLE	599
36.1. OVERVIEW	599
36.2. LOADING EXTENSION SCRIPTS AND STYLESHEETS	599
36.2.1. Setting Extension Properties	600
36.3. EXTENSION OPTION FOR EXTERNAL LOGGING SOLUTIONS	601
36.4. CUSTOMIZING AND DISABLING THE GUIDED TOUR	601
36.5. CUSTOMIZING DOCUMENTATION LINKS	601
36.6. CUSTOMIZING THE LOGO	602
36.7. CUSTOMIZING THE MEMBERSHIP WHITELIST	602
36.8. CHANGING LINKS TO DOCUMENTATION	602
36.9. ADDING OR CHANGING LINKS TO DOWNLOAD THE CLI	603
36.9.1. Customizing the About Page	603
36.10. CONFIGURING NAVIGATION MENUS	604
36.10.1. Top Navigation Dropdown Menus	604
36.10.2. Application Launcher	605
36.10.3. System Status Badge	605
36.10.4. Project Left Navigation	606
36.11. CONFIGURING FEATURED APPLICATIONS	608
36.12. CONFIGURING CATALOG CATEGORIES	608
36.13. CONFIGURING QUOTA NOTIFICATION MESSAGES	610
36.14. CONFIGURING THE CREATE FROM URL NAMESPACE WHITELIST	610
36.15. DISABLING THE COPY LOGIN COMMAND	611
36.15.1. Enabling Wildcard Routes	611
36.16. CUSTOMIZING THE LOGIN PAGE	611
36.16.1. Example Usage	612
36.17. CUSTOMIZING THE OAUTH ERROR PAGE	612
36.18. CHANGING THE LOGOUT URL	612
36.19. CONFIGURING WEB CONSOLE CUSTOMIZATIONS WITH ANSIBLE	613
36.20. CHANGING THE WEB CONSOLE URL PORT AND CERTIFICATES	614
CHAPTER 37. DEPLOYING EXTERNAL PERSISTENT VOLUME PROVISIONERS	615
37.1. OVERVIEW	615
37.2. BEFORE YOU BEGIN	615
37.2.1. External Provisioners Ansible Role	615
37.2.2. External Provisioners Ansible Variables	615
37.2.3. AWS EFS Provisioner Ansible Variables	616

37.3. DEPLOYING THE PROVISIONERS	617
37.3.1. Deploying the AWS EFS Provisioner	617
37.3.1.1. AWS EFS Object Definition	617
37.4. CLEANUP	618

CHAPTER 1. OVERVIEW

This guide covers further configuration options available for your OpenShift Container Platform cluster post-installation.

CHAPTER 2. SETTING UP THE REGISTRY

2.1. REGISTRY OVERVIEW

2.1.1. About the Registry

OpenShift Container Platform can build [container images](#) from your source code, deploy them, and manage their lifecycle. To enable this, OpenShift Container Platform provides an internal, [integrated Docker registry](#) that can be deployed in your OpenShift Container Platform environment to locally manage images.

2.1.2. Integrated or Stand-alone Registries

During an initial installation of a full OpenShift Container Platform cluster, it is likely that the registry was deployed automatically during the installation process. If it was not, or if you want to further customize the configuration of your registry, see [Deploying a Registry on Existing Clusters](#).

While it can be deployed to run as an integrated part of your full OpenShift Container Platform cluster, the OpenShift Container Platform registry can alternatively be installed separately as a stand-alone container image registry.

To install a stand-alone registry, follow [Installing a Stand-alone Registry](#). This installation path deploys an all-in-one cluster running a registry and specialized web console.

2.1.3. Red Hat Quay Registries

If you need an enterprise-quality container image registry, Red Hat Quay is available both as a hosted service and as software you can install in your own data center or cloud environment. Advanced registry features in Red Hat Quay include geo-replication, image scanning, and the ability to rollback images.

Visit the [Quay.io](#) site to set up your own hosted Quay registry account. After that, the [Quay Tutorial](#) helps you login to the Quay registry and start managing your images. Alternatively, refer to [Getting Started with Red Hat Quay](#) for information on setting up your own Red Hat Quay registry.

At the moment, you access your Red Hat Quay registry from OpenShift as you would any remote container image registry. To learn how to set up credentials to access Red Hat Quay as a secured registry, refer to [Allowing Pods to Reference Images from Other Secured Registries](#).

2.2. DEPLOYING A REGISTRY ON EXISTING CLUSTERS

2.2.1. Overview

If the integrated registry was not previously deployed automatically during the initial installation of your OpenShift Container Platform cluster, or if it is no longer running successfully and you need to redeploy it on your existing cluster, see the following sections for options on deploying a new registry.



NOTE

This topic is not required if you installed a [stand-alone registry](#).

2.2.2. Deploying the Registry

To deploy the integrated Docker registry, use the **oc adm registry** command as a user with cluster administrator privileges. For example:

```
$ oc adm registry --config=/etc/origin/master/admin.kubeconfig \ 1
    --service-account=registry \ 2
    --images='registry.access.redhat.com/openshift3/ose-
    ${component}:${version}' 3
```

1 **--config** is the path to the [CLI configuration file](#) for the [cluster administrator](#).

2 **--service-account** is the service account used to run the registry's pod.

3 Required to pull the correct image for OpenShift Container Platform.

This creates a service and a deployment configuration, both called **docker-registry**. Once deployed successfully, a pod is created with a name similar to **docker-registry-1-cpty9**.

To see a full list of options that you can specify when creating the registry:

```
$ oc adm registry --help
```

The value for **--fs-group** must be [permitted by the SCC](#) used by the registry (typically, the restricted SCC).

2.2.3. Deploying the Registry as a DaemonSet

Use the **oc adm registry** command to deploy the registry as a **DaemonSet** with the **--daemonset** option.

Daemonsets ensure that when nodes are created, they contain copies of a specified pod. When the nodes are removed, the pods are garbage collected.

For more information on **DaemonSets**, see [Using Daemonsets](#).

2.2.4. Registry Compute Resources

By default, the registry is created with no settings for [compute resource requests or limits](#). For production, it is highly recommended that the deployment configuration for the registry be updated to set resource requests and limits for the registry pod. Otherwise, the registry pod will be considered a **BestEffort pod**.

See [Compute Resources](#) for more information on configuring requests and limits.

2.2.5. Storage for the Registry

The registry stores container images and metadata. If you simply deploy a pod with the registry, it uses an ephemeral volume that is destroyed if the pod exits. Any images anyone has built or pushed into the registry would disappear.

This section lists the supported registry storage drivers. See [the Docker registry documentation for more information](#).

The following list includes storage drivers that need to be configured in the registry's configuration file:

- [Filesystem](#). Filesystem is the default and does not need to be configured.
- [S3](#). See the [CloudFront configuration](#) documentation for more information.
- [OpenStack Swift](#)
- [Google Cloud Storage \(GCS\)](#)
- [Microsoft Azure](#)
- [Aliyun OSS](#)

General registry storage configuration options are supported. See [the Docker registry documentation for more information](#).

The following storage options need to be configured through the [filesystem driver](#):

- [GlusterFS Storage](#)
- [Ceph Rados Block Device](#)



NOTE

For more information on supported persistent storage drivers, see [Configuring Persistent Storage](#) and [Persistent Storage Examples](#).

2.2.5.1. Production Use

For production use, attach a remote volume or [define and use the persistent storage method of your choice](#).

For example, to use an existing persistent volume claim:

```
$ oc volume deploymentconfigs/docker-registry --add --name=registry-storage -t pvc \
    --claim-name=<pvc_name> --overwrite
```



IMPORTANT

Testing shows issues with using the NFS server on RHEL as storage backend for the container registry. This includes the OpenShift Container Registry and Quay. Therefore, using NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift core components.

2.2.5.1.1. Use Amazon S3 as a Storage Back-end

There is also an option to use Amazon Simple Storage Service storage with the internal Docker registry. It is a secure cloud storage manageable through [AWS Management Console](#). To use it, the registry's configuration file must be manually edited and mounted to the registry pod. However, before you start with the configuration, look at upstream's [recommended steps](#).

Take a [default YAML configuration file](#) as a base and replace the **filesystem** entry in the **storage** section with **s3** entry such as below. The resulting storage section may look like this:

```
storage:
  cache:
    layerinfo: inmemory
  delete:
    enabled: true
  s3:
    accesskey: awsaccesskey 1
    secretkey: awssecretkey 2
    region: us-west-1
    regionendpoint: http://myobjects.local
    bucket: bucketname
    encrypt: true
    keyid: mykeyid
    secure: true
    v4auth: false
    chunksize: 5242880
    rootdirectory: /s3/object/name/prefix
```

1 Replace with your Amazon access key.

2 Replace with your Amazon secret key.

All of the **s3** configuration options are documented in upstream's [driver reference documentation](#).

[Overriding the registry configuration](#) will take you through the additional steps on mounting the configuration file into pod.

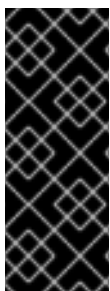


WARNING

When the registry runs on the S3 storage back-end, there are [reported issues](#).

2.2.5.2. Non-Production Use

For non-production use, you can use the `--mount-host=<path>` option to specify a directory for the registry to use for persistent storage. The registry volume is then created as a host-mount at the specified `<path>`.



IMPORTANT

The `--mount-host` option mounts a directory from the node on which the registry container lives. If you scale up the **docker-registry** deployment configuration, it is possible that your registry pods and containers will run on different nodes, which can result in two or more registry containers, each with its own local storage. This will lead to unpredictable behavior, as subsequent requests to pull the same image repeatedly may not always succeed, depending on which container the request ultimately goes to.

The `--mount-host` option requires that the registry container run in privileged mode. This is automatically enabled when you specify `--mount-host`. However, not all pods are allowed to run [privileged containers](#) by default. If you still want to use this option, create the registry and specify that it use the **registry** service account that was created during installation:

```
$ oc adm registry --service-account=registry \
  --config=/etc/origin/master/admin.kubeconfig \
  --images='registry.access.redhat.com/openshift3/ose-
${component}:${version}' \
  --mount-host=<path>
```

IMPORTANT

The Docker registry pod runs as user **1001**. This user must be able to write to the host directory. You may need to change directory ownership to user ID **1001** with this command:

```
$ sudo chown 1001:root <path>
```

2.2.6. Enabling the Registry Console

OpenShift Container Platform provides a web-based interface to the integrated registry. This registry console is an optional component for browsing and managing images. It is deployed as a stateless service running as a pod.

NOTE

If you installed OpenShift Container Platform as a [stand-alone registry](#), the registry console is already deployed and secured automatically during installation.

IMPORTANT

If Cockpit is already running, you'll need to shut it down before proceeding in order to avoid a port conflict (9090 by default) with the registry console.

2.2.6.1. Deploying the Registry Console

IMPORTANT

You must first have [exposed the registry](#).

1. Create a passthrough route in the **default** project. You will need this when creating the registry console application in the next step.

```
$ oc create route passthrough --service registry-console \
  --port registry-console \
  -n default
```

2. Deploy the registry console application. Replace `<openshift_oauth_url>` with the URL of the OpenShift Container Platform OAuth provider, which is typically the master.

```
$ oc new-app -n default --template=registry-console \
```

```
-p
OPENSIFT_OAUTH_PROVIDER_URL="https://<openshift_oauth_url>:8443" \
-p REGISTRY_HOST=$(oc get route docker-registry -n default --
template='{{ .spec.host }}') \
-p COCKPIT_KUBE_URL=$(oc get route registry-console -n default -
-template='https://{{ .spec.host }}')
```



NOTE

If the redirection URL is wrong when you are trying to log in to the registry console, check your OAuth client with **oc get oauthclients**.

3. Finally, use a web browser to view the console using the route URL.

2.2.6.2. Securing the Registry Console

By default, the registry console generates self-signed TLS certificates if deployed manually per the steps in [Deploying the Registry Console](#). See [Troubleshooting the Registry Console](#) for more information.

Use the following steps to add your organization's signed certificates as a secret volume. This assumes your certificates are available on the **oc** client host.

1. Create a **.cert** file containing the certificate and key. Format the file with:

- One or more **BEGIN CERTIFICATE** blocks for the server certificate and the intermediate certificate authorities
 - A block containing a **BEGIN PRIVATE KEY** or similar for the key. The key must not be encrypted
- For example:

```
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgNV
BAoMIGI00GE2NGNkNmMwNTQ1YThhZTgx0TEzZDE5YmJjMmRjMRIwEAYDVQQDDAIs
...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgNV
BAoMIGI00GE2NGNkNmMwNTQ1YThhZTgx0TEzZDE5YmJjMmRjMRIwEAYDVQQDDAIs
...
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBAgEAAoIBAQCyOJ5garOYw0sm
8TBCDSqQ/H1awGMzDYdB11xuHhsxYS2VepPMzMzryHR137I4dGFLhvdTvJUH8lUS
...
-----END PRIVATE KEY-----
```

- The secured registry should contain the following Subject Alternative Names (SAN) list:
 - Two service hostnames.

For example:

```
docker-registry.default.svc.cluster.local
docker-registry.default.svc
```

- o Service IP address.

For example:

```
172.30.124.220
```

Use the following command to get the Docker registry service IP address:

```
oc get service docker-registry --  
template='{{.spec.clusterIP}}'
```

- o Public hostname.

For example:

```
docker-registry-default.apps.example.com
```

Use the following command to get the Docker registry public hostname:

```
oc get route docker-registry --template '{{.spec.host}}'
```

For example, the server certificate should contain SAN details similar to the following:

```
X509v3 Subject Alternative Name:  
DNS:docker-registry-public.openshift.com,  
DNS:docker-registry.default.svc, DNS:docker-  
registry.default.svc.cluster.local, DNS:172.30.2.98, IP  
Address:172.30.2.98
```

The registry console loads a certificate from the **/etc/cockpit/ws-certs.d** directory. It uses the last file with a **.cert** extension in alphabetical order. Therefore, the **.cert** file should contain at least two PEM blocks formatted in the OpenSSL style.

If no certificate is found, a self-signed certificate is created using the **openssl** command and stored in the **0-self-signed.cert** file.

2. Create the secret:

```
$ oc create secret generic console-secret \  
--from-file=/path/to/console.cert
```

3. Add the secrets to the **registry-console** deployment configuration:

```
$ oc volume dc/registry-console --add --type=secret \  
--secret-name=console-secret -m /etc/cockpit/ws-certs.d
```

This triggers a new deployment of the registry console to include your signed certificates.

2.2.6.3. Troubleshooting the Registry Console

2.2.6.3.1. Debug Mode

The registry console debug mode is enabled using an environment variable. The following command redeploys the registry console in debug mode:

```
$ oc set env dc registry-console G_MESSAGES_DEBUG=cockpit-ws,cockpit-wraper
```

Enabling debug mode allows more verbose logging to appear in the registry console's pod logs.

2.2.6.3.2. Display SSL Certificate Path

To check which certificate the registry console is using, a command can be run from inside the console pod.

1. List the pods in the **default** project and find the registry console's pod name:

```
$ oc get pods -n default
NAME                                READY    STATUS    RESTARTS    AGE
registry-console-1-rssrw           1/1      Running   0            1d
```

2. Using the pod name from the previous command, get the certificate path that the **cockpit-ws** process is using. This example shows the console using the auto-generated certificate:

```
$ oc exec registry-console-1-rssrw remotectl certificate
certificate: /etc/cockpit/ws-certs.d/0-self-signed.cert
```

2.3. ACCESSING THE REGISTRY

2.3.1. Viewing Logs

To view the logs for the Docker registry, use the **oc logs** command with the deployment configuration:

```
$ oc logs dc/docker-registry
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info
msg="redis not configured" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info
msg="using inmemory layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info
msg="Using OpenShift Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info
msg="listening on :5000" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

2.3.2. File Storage

Tag and image metadata is stored in OpenShift Container Platform, but the registry stores layer and signature data in a volume that is mounted into the registry container at **/registry**. As **oc exec** does not work on privileged containers, to view a registry's contents you must manually SSH into the node housing the registry pod's container, then run **docker exec** on the container itself:

1. List the current pods to find the pod name of your Docker registry:

```
# oc get pods
```

Then, use **oc describe** to find the host name for the node running the container:

```
# oc describe pod <pod_name>
```

2. Log into the desired node:

```
# ssh node.example.com
```

3. List the running containers from the default project on the node host and identify the container ID for the Docker registry:

```
# docker ps --filter=name=registry_docker-registry.*_default_
```

4. List the registry contents using the **oc rsh** command:

```
# oc rsh dc/docker-registry find /registry
/registry/docker
/registry/docker/registry
/registry/docker/registry/v2
/registry/docker/registry/v2/blobs ❶
/registry/docker/registry/v2/blobs/sha256
/registry/docker/registry/v2/blobs/sha256/ed
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
d83c648c24f92cece5f89d95ac6c34ce751111810/data ❷
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845/data
/registry/docker/registry/v2/repositories ❸
/registry/docker/registry/v2/repositories/p1
/registry/docker/registry/v2/repositories/p1/pause ❹
/registry/docker/registry/v2/repositories/p1/pause/_manifests
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures ❺
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
```



```

ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810/link 6
/registry/docker/registry/v2/repositories/p1/pause/_uploads 7
/registry/docker/registry/v2/repositories/p1/pause/_layers 8
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4/link
9
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845/link

```

- 1 This directory stores all layers and signatures as blobs.
- 2 This file contains the blob's contents.
- 3 This directory stores all the image repositories.
- 4 This directory is for a single image repository **p1/pause**.
- 5 This directory contains signatures for a particular image manifest revision.
- 6 This file contains a reference back to a blob (which contains the signature data).
- 7 This directory contains any layers that are currently being uploaded and staged for the given repository.
- 8 This directory contains links to all the layers this repository references.
- 9 This file contains a reference to a specific layer that has been linked into this repository via an image.

2.3.3. Accessing the Registry Directly

For advanced usage, you can access the registry directly to invoke **docker** commands. This allows you to push images to or pull them from the integrated registry directly using operations like **docker push** or **docker pull**. To do so, you must be logged in to the registry using the **docker login** command. The operations you can perform depend on your user permissions, as described in the following sections.

2.3.3.1. User Prerequisites

To access the registry directly, the user that you use must satisfy the following, depending on your intended usage:

- For any direct access, you must have a [regular user](#) for your preferred [identity provider](#). A

regular user can generate an access token required for logging in to the registry. [System users](#), such as **system:admin**, cannot obtain access tokens and, therefore, cannot access the registry directly.

For example, if you are using **HTPASSWD** authentication, you can create one using the following command:

```
# htpasswd /etc/origin/openshift-htpasswd <user_name>
```

- For pulling images, for example when using the **docker pull** command, the user must have the **registry-viewer** role. To add this role:

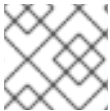
```
$ oc policy add-role-to-user registry-viewer <user_name>
```

- For writing or pushing images, for example when using the **docker push** command, the user must have the **registry-editor** role. To add this role:

```
$ oc policy add-role-to-user registry-editor <user_name>
```

For more information on user permissions, see [Managing Role Bindings](#).

2.3.3.2. Logging in to the Registry



NOTE

Ensure your user satisfies the [prerequisites](#) for accessing the registry directly.

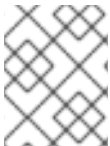
To log in to the registry directly:

1. Ensure you are logged in to OpenShift Container Platform as a **regular user**:

```
$ oc login
```

2. Log in to the Docker registry by using your access token:

```
docker login -u openshift -p $(oc whoami -t) <registry_ip>:<port>
```

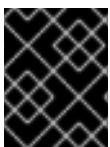


NOTE

You can pass any value for the username, the token contains all necessary information. Passing a username that contains colons will result in a login failure.

2.3.3.3. Pushing and Pulling Images

After [logging in to the registry](#), you can perform **docker pull** and **docker push** operations against your registry.



IMPORTANT

You can pull arbitrary images, but if you have the **system:registry** role added, you can only push images to the registry in your project.

In the following examples, we use:

Component	Value
<code><registry_ip></code>	172.30.124.220
<code><port></code>	5000
<code><project></code>	openshift
<code><image></code>	busybox
<code><tag></code>	omitted (defaults to latest)

1. Pull an arbitrary image:

```
$ docker pull docker.io/busybox
```

2. Tag the new image with the form `<registry_ip>:<port>/<project>/<image>`. The project name **must** appear in this [pull specification](#) for OpenShift Container Platform to correctly place and later access the image in the registry.

```
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
```



NOTE

Your regular user must have the **system:image-builder** role for the specified project, which allows the user to write or push an image. Otherwise, the **docker push** in the next step will fail. To test, you can [create a new project](#) to push the **busybox** image.

3. Push the newly-tagged image to your registry:

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403ca
b55
```

2.3.4. Accessing Registry Metrics

The OpenShift Container Registry provides an endpoint for [Prometheus metrics](#). Prometheus is a stand-alone, open source systems monitoring and alerting toolkit.

The metrics are exposed at the `/extensions/v2/metrics` path of the registry endpoint. However, this route must first be enabled; see [Extended Registry Configuration](#) for instructions.

The following is a simple example of a metrics query:

```
$ curl -s -u <user>:<secret> \ ❶
    http://172.30.30.30:5000/extensions/v2/metrics | grep openshift | head
-n 10

# HELP openshift_build_info A metric with a constant '1' value labeled by
major, minor, git commit & git version from which OpenShift was built.
# TYPE openshift_build_info gauge
openshift_build_info{gitCommit="67275e1",gitVersion="v3.6.0-
alpha.1+67275e1-803",major="3",minor="6+"} 1
# HELP openshift_registry_request_duration_seconds Request latency summary
in microseconds for each operation
# TYPE openshift_registry_request_duration_seconds summary
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.5"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.9"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.99"} 0
openshift_registry_request_duration_seconds_sum{name="test/origin-
pod",operation="blobstore.create"} 0
openshift_registry_request_duration_seconds_count{name="test/origin-
pod",operation="blobstore.create"} 5
```

❶ **<user>** can be arbitrary, but **<secret>** must match the value specified in the [registry configuration](#).

Another method to access the metrics is to use a cluster role. You still need to enable the endpoint, but you do not need to specify a **<secret>**. The part of the configuration file responsible for metrics should look like this:

```
openshift:
  version: 1.0
  metrics:
    enabled: true
  ...
```

You must create a cluster role if you do not already have one to access the metrics:

```
$ cat <<EOF |
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-scraper
rules:
- apiGroups:
  - image.openshift.io
  resources:
  - registry/metrics
  verbs:
  - get
EOF
oc create -f -
```

To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
```

See the [upstream Prometheus documentation](#) for more advanced queries and recommended visualizers.

2.4. SECURING AND EXPOSING THE REGISTRY

2.4.1. Overview

By default, the OpenShift Container Platform registry is secured during cluster installation so that it serves traffic via TLS. A passthrough route is also created by default to expose the service externally.

If for any reason your registry has not been secured or exposed, see the following sections for steps on how to manually do so.

2.4.2. Manually Securing the Registry

To manually secure the registry to serve traffic via TLS:

1. [Deploy the registry](#).
2. Fetch the service IP and port of the registry:

```
$ oc get svc/docker-registry
NAME                                LABELS
SELECTOR                            IP(S)                                PORT(S)
docker-registry                    docker-registry=default              docker-
registry=default                    172.30.124.220                      5000/TCP
```

3. You can use an existing server certificate, or create a key and server certificate valid for specified IPs and host names, signed by a specified CA. To create a server certificate for the registry service IP and the **docker-registry.default.svc.cluster.local** host name, run the following command from the first master listed in the Ansible host inventory file, by default **/etc/ansible/hosts**:

```
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --signer-serial=/etc/origin/master/ca.serial.txt \
  --hostnames='docker-registry.default.svc.cluster.local,docker-
registry.default.svc,172.30.124.220' \
  --cert=/etc/secrets/registry.crt \
  --key=/etc/secrets/registry.key
```

If the router will be [exposed externally](#), add the public route host name in the **--hostnames** flag:

```
--hostnames='mydocker-registry.example.com,docker-
registry.default.svc.cluster.local,172.30.124.220 \
```

See [Redeploying Registry and Router Certificates](#) for additional details on updating the default certificate so that the route is externally accessible.

**NOTE**

The **oc adm ca create-server-cert** command generates a certificate that is valid for two years. This can be altered with the **--expire-days** option, but for security reasons, it is recommended to not make it greater than this value.

4. Create the secret for the registry certificates:

```
$ oc create secret generic registry-certificates \
  --from-file=/etc/secrets/registry.crt \
  --from-file=/etc/secrets/registry.key
```

5. Add the secret to the registry pod's service accounts (including the **default** service account):

```
$ oc secrets link registry registry-certificates
$ oc secrets link default registry-certificates
```

**NOTE**

Limiting secrets to only the service accounts that reference them is disabled by default. This means that if **serviceAccountConfig.limitSecretReferences** is set to **false** (the default setting) in the master configuration file, linking secrets to a service is not required.

6. Pause the **docker-registry** service:

```
$ oc rollout pause dc/docker-registry
```

7. Add the secret volume to the registry deployment configuration:

```
$ oc volume dc/docker-registry --add --type=secret \
  --secret-name=registry-certificates -m /etc/secrets
```

8. Enable TLS by adding the following environment variables to the registry deployment configuration:

```
$ oc set env dc/docker-registry \
  REGISTRY_HTTP_TLS_CERTIFICATE=/etc/secrets/registry.crt \
  REGISTRY_HTTP_TLS_KEY=/etc/secrets/registry.key
```

See the [Configuring a registry section of the Docker documentation](#) for more information.

9. Update the scheme used for the registry's liveness probe from HTTP to HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers": [{
  "name": "registry",
  "livenessProbe": {"httpGet": {"scheme": "HTTPS"}}
}]}}}]'
```

10. If your registry was initially deployed on OpenShift Container Platform 3.2 or later, update the scheme used for the registry's readiness probe from HTTP to HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers": [{
  "name": "registry",
  "readinessProbe": {"httpGet": {"scheme": "HTTPS"}}
}]}}}]'
```

11. Resume the **docker-registry** service:

```
$ oc rollout resume dc/docker-registry
```

12. Validate the registry is running in TLS mode. Wait until the latest **docker-registry** deployment completes and verify the Docker logs for the registry container. You should find an entry for **listening on :5000, tls**.

```
$ oc logs dc/docker-registry | grep tls
time="2015-05-27T05:05:53Z" level=info msg="listening on :5000, tls"
instance.id=deeba528-c478-41f5-b751-dc48e4935fc2
```

13. Copy the CA certificate to the Docker certificates directory. This must be done on all nodes in the cluster:

```
$ dcertsdir=/etc/docker/certs.d
$ destdir_addr=$dcertsdir/172.30.124.220:5000
$ destdir_name=$dcertsdir/docker-
registry.default.svc.cluster.local:5000

$ sudo mkdir -p $destdir_addr $destdir_name
$ sudo cp ca.crt $destdir_addr
$ sudo cp ca.crt $destdir_name
```

1 The **ca.crt** file is a copy of **/etc/origin/master/ca.crt** on the master.

14. When using authentication, some versions of **docker** also require you to configure your cluster to trust the certificate at the OS level.

- a. Copy the certificate:

```
$ cp /etc/origin/master/ca.crt /etc/pki/ca-
trust/source/anchors/myregistrydomain.com.crt
```

- b. Run:

```
$ update-ca-trust enable
```

15. Remove the **--insecure-registry** option only for this particular registry in the **/etc/sysconfig/docker** file. Then, reload the daemon and restart the **docker** service to reflect this configuration change:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

16. Validate the **docker** client connection. Running **docker push** to the registry or **docker pull** from the registry should succeed. Make sure you have [logged into the registry](#).

```
$ docker tag|push <registry/image> <internal_registry/project/image>
```

For example:

```
$ docker pull busybox
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403ca
b55
```

2.4.3. Manually Exposing a Secure Registry

Instead of logging in to the OpenShift Container Platform registry from within the OpenShift Container Platform cluster, you can gain external access to it by first securing the registry and then exposing it with a route. This allows you to log in to the registry from outside the cluster using the route address, and to tag and push images using the route host.

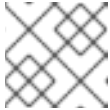
1. Each of the following prerequisite steps are performed by default during a typical cluster installation. If they have not been, perform them manually:
 - a. [Manually deploy the registry](#).
 - b. [Manually secure the registry](#).
 - c. [Manually deploy a router](#).
2. A [passthrough](#) route should have been created by default for the registry during the initial cluster installation:
 - a. Verify whether the route exists:

```
$ oc get route/docker-registry -o yaml
apiVersion: v1
kind: Route
metadata:
  name: docker-registry
spec:
  host: <host> ❶
  to:
    kind: Service
    name: docker-registry ❷
  tls:
    termination: passthrough ❸
```

❶ The host for your route. You must be able to resolve this name externally via DNS to the router's IP address.

❷ The service name for your registry.

- 3 Specifies this route as a passthrough route.



NOTE

Re-encrypt routes are also supported for exposing the secure registry.

- b. If it does not exist, create the route via the **oc create route passthrough** command, specifying the registry as the route's service. By default, the name of the created route is the same as the service name:

- i. Get the **docker-registry** service details:

```
$ oc get svc
NAME                                CLUSTER_IP          EXTERNAL_IP    PORT(S)
SELECTOR                            AGE
docker-registry                    172.30.69.167       <none>         5000/TCP
docker-registry=default            4h
kubernetes                         172.30.0.1          <none>
443/TCP, 53/UDP, 53/TCP            <none>              4h
router                             172.30.172.132      <none>         80/TCP
router=router                      4h
```

- ii. Create the route:

```
$ oc create route passthrough \
  --service=docker-registry \ 1
  --hostname=<host>
route "docker-registry" created 2
```

- 1 Specifies the registry as the route's service.
- 2 The route name is identical to the service name.

3. Next, you must trust the certificates being used for the registry on your host system to allow the host to push and pull images. The certificates referenced were created when you secured your registry.

```
$ sudo mkdir -p /etc/docker/certs.d/<host>
$ sudo cp <ca_certificate_file> /etc/docker/certs.d/<host>
$ sudo systemctl restart docker
```

4. [Log in to the registry](#) using the information from securing the registry. However, this time point to the host name used in the route rather than your service IP. When logging in to a secured and exposed registry, make sure you specify the registry in the **docker login** command:

```
# docker login -e user@company.com \
  -u f83j5h6 \
  -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
  <host>
```

5. You can now tag and push images using the route host. For example, to tag and push a **busybox** image in a project called **test**:

```
$ oc get imagestreams -n test
```

NAME	DOCKER REPO	TAGS	UPDATED
busybox	172.30.11.215:5000/test/busybox	latest	2 seconds ago

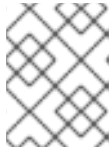
```

$ docker pull busybox
$ docker tag busybox <host>/test/busybox
$ docker push <host>/test/busybox
The push refers to a repository [<host>/test/busybox] (len: 1)
8c2e06607696: Image already exists
6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image successfully pushed
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31
Status: Image is up to date for <host>/test/busybox:latest

$ oc get imagestreams -n test
```

NAME	DOCKER REPO	TAGS	UPDATED
busybox	172.30.11.215:5000/test/busybox	latest	2 seconds ago

**NOTE**

Your image streams will have the IP address and port of the registry service, not the route name and port. See `oc get imagestreams` for details.

2.4.4. Manually Exposing a Non-Secure Registry

Instead of securing the registry in order to expose the registry, you can simply expose a non-secure registry for non-production OpenShift Container Platform environments. This allows you to have an external route to the registry without using SSL certificates.

**WARNING**

Only non-production environments should expose a non-secure registry to external access.

To expose a non-secure registry:

1. Expose the registry:

```
# oc expose service docker-registry --hostname=<hostname> -n default
```

This creates the following JSON file:

```
apiVersion: v1
kind: Route
metadata:
  creationTimestamp: null
  labels:
    docker-registry: default
  name: docker-registry
spec:
  host: registry.example.com
  port:
    targetPort: "5000"
  to:
    kind: Service
    name: docker-registry
status: {}
```

2. Verify that the route has been created successfully:

```
# oc get route
NAME                                HOST/PORT                                PATH      SERVICE
LABELS                                INSECURE POLICY    TLS TERMINATION
docker-registry    registry.example.com                                docker-registry
docker-registry=default
```

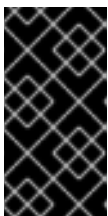
3. Check the health of the registry:

```
$ curl -v http://registry.example.com/healthz
```

Expect an HTTP 200/OK message.

After exposing the registry, update your `/etc/sysconfig/docker` file by adding the port number to the **OPTIONS** entry. For example:

```
OPTIONS='--selinux-enabled --insecure-registry=172.30.0.0/16 --
insecure-registry registry.example.com:80'
```



IMPORTANT

The above options should be added on the client from which you are trying to log in.

Also, ensure that Docker is running on the client.

When [logging in](#) to the non-secured and exposed registry, make sure you specify the registry in the **docker login** command. For example:

```
# docker login -e user@company.com \
-u f83j5h6 \
-p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
<host>
```

2.5. EXTENDED REGISTRY CONFIGURATION

2.5.1. Maintaining the Registry IP Address

OpenShift Container Platform refers to the integrated registry by its service IP address, so if you decide to delete and recreate the **docker-registry** service, you can ensure a completely transparent transition by arranging to re-use the old IP address in the new service. If a new IP address cannot be avoided, you can minimize cluster disruption by rebooting only the masters.

Re-using the Address

To re-use the IP address, you must save the IP address of the old **docker-registry** service prior to deleting it, and arrange to replace the newly assigned IP address with the saved one in the new **docker-registry** service.

1. Make a note of the **clusterIP** for the service:

```
$ oc get svc/docker-registry -o yaml | grep clusterIP:
```

2. Delete the service:

```
$ oc delete svc/docker-registry dc/docker-registry
```

3. Create the registry definition in **registry.yaml**, replacing **<options>** with, for example, those used in step 3 of the instructions in the [Non-Production Use](#) section:

```
$ oc adm registry <options> -o yaml > registry.yaml
```

4. Edit **registry.yaml**, find the **Service** there, and change its **clusterIP** to the address noted in step 1.
5. Create the registry using the modified **registry.yaml**:

```
$ oc create -f registry.yaml
```

Rebooting the Masters

If you are unable to re-use the IP address, any operation that uses a [pull specification](#) that includes the old IP address will fail. To minimize cluster disruption, you must reboot the masters:

```
# master-restart api
# master-restart controllers
```

This ensures that the old registry URL, which includes the old IP address, is cleared from the cache.



NOTE

We recommend against rebooting the entire cluster because that incurs unnecessary downtime for pods and does not actually clear the cache.

2.5.2. Whitelisting Docker Registries

You can specify a whitelist of docker registries, allowing you to curate a set of images and templates that are available for download by OpenShift Container Platform users. This curated set can be placed in one

or more docker registries, and then added to the whitelist. When using a whitelist, only the specified registries are accessible within OpenShift Container Platform, and all other registries are denied access by default.

To configure a whitelist:

1. Edit the `/etc/sysconfig/docker` file to block all registries:

```
BLOCK_REGISTRY='--block-registry=all'
```

You may need to uncomment the **BLOCK_REGISTRY** line.

2. In the same file, add registries to which you want to allow access:

```
ADD_REGISTRY='--add-registry=<registry1> --add-registry=<registry2>'
```

Allowing Access to Registries

```
ADD_REGISTRY='--add-registry=registry.access.redhat.com'
```

This example would restrict access to images available on the [Red Hat Customer Portal](#).

Once the whitelist is configured, if a user tries to pull from a docker registry that is not on the whitelist, they will receive an error message stating that this registry is not allowed.

2.5.3. Setting the Registry Hostname

You can configure the hostname and port the registry is known by for both internal and external references. By doing this, image streams will provide hostname based push and pull specifications for images, allowing consumers of the images to be isolated from changes to the registry service ip and potentially allowing image streams and their references to be portable between clusters.

To set the hostname used to reference the registry from within the cluster, set the **internalRegistryHostname** in the **imagePolicyConfig** section of the master configuration file. The external hostname is controlled by setting the **externalRegistryHostname** value in the same location.

Image Policy Configuration

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

The registry itself must be configured with the same internal hostname value. This can be accomplished by setting the **REGISTRY_OPENSHIFT_SERVER_ADDR** environment variable on the registry deployment configuration, or by setting the value in the [OpenShift section](#) of the registry configuration.



NOTE

If you have enabled TLS for your registry the server certificate must include the hostnames by which you expect the registry to be referenced. See [securing the registry](#) for instructions on adding hostnames to the server certificate.

2.5.4. Overriding the Registry Configuration

You can override the integrated registry's default configuration, found by default at `/config.yml` in a running registry's container, with your own [custom configuration](#).



NOTE

Upstream configuration options in this file may also be overridden using environment variables. The [middleware section](#) is an exception as there are just a few options that can be overridden using environment variables. [Learn how to override specific configuration options](#).

To enable management of the registry configuration file directly and deploy an updated configuration using a **ConfigMap**:

1. [Deploy the registry](#).
2. Edit the registry configuration file locally as needed. The initial YAML file deployed on the registry is provided below. [Review supported options](#).

Registry Configuration File

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /registry
delete:
  enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
repository:
  - name: openshift
    options:
      acceptschema2: true
      pullthrough: true
      enforcequota: false
      projectcachettl: 1m
      blobrepositorycachettl: 10m
storage:
  - name: openshift
openshift:
  version: 1.0
metrics:
  enabled: false
  secret: <secret>
```

3. Create a **ConfigMap** holding the content of each file in this directory:

```
$ oc create configmap registry-config \
  --from-file=</path/to/custom/registry/config.yml>/
```

4. Add the **registry-config** ConfigMap as a volume to the registry's deployment configuration to mount the custom configuration file at **/etc/docker/registry/**:

```
$ oc volume dc/docker-registry --add --type=configmap \
  --configmap-name=registry-config -m /etc/docker/registry/
```

5. Update the registry to reference the configuration path from the previous step by adding the following environment variable to the registry's deployment configuration:

```
$ oc set env dc/docker-registry \
  REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yml
```

This may be performed as an iterative process to achieve the desired configuration. For example, during troubleshooting, the configuration may be temporarily updated to put it in **debug** mode.

To update an existing configuration:



WARNING

This procedure will overwrite the currently deployed registry configuration.

1. Edit the local registry configuration file, **config.yml**.
2. Delete the **registry-config** configmap:

```
$ oc delete configmap registry-config
```

3. Recreate the configmap to reference the updated configuration file:

```
$ oc create configmap registry-config \
  --from-file=</path/to/custom/registry/config.yml>/
```

4. Redeploy the registry to read the updated configuration:

```
$ oc rollout latest docker-registry
```

TIP

Maintain configuration files in a source control repository.

2.5.5. Registry Configuration Reference

There are many configuration options available in the upstream [docker distribution](#) library. Not all [configuration options](#) are supported or enabled. Use this section as a reference when [overriding the registry configuration](#).

**NOTE**

Upstream configuration options in this file may also be overridden using environment variables. However, the [middleware section](#) may **not** be overridden using environment variables. [Learn how to override specific configuration options](#).

2.5.5.1. Log

[Upstream options](#) are supported.

Example:

```
log:
  level: debug
  formatter: text
  fields:
    service: registry
    environment: staging
```

2.5.5.2. Hooks

Mail hooks are not supported.

2.5.5.3. Storage

This section lists the supported registry storage drivers. See [the Docker registry documentation for more information](#).

The following list includes storage drivers that need to be configured in the registry's configuration file:

- [Filesystem](#). Filesystem is the default and does not need to be configured.
- [S3](#). See the [CloudFront configuration](#) documentation for more information.
- [OpenStack Swift](#)
- [Google Cloud Storage \(GCS\)](#)
- [Microsoft Azure](#)
- [Aliyun OSS](#)

General registry storage configuration options are supported. See [the Docker registry documentation for more information](#).

The following storage options need to be configured through the [filesystem driver](#):

- [GlusterFS Storage](#)
- [Ceph Rados Block Device](#)

**NOTE**

For more information on supported persistent storage drivers, see [Configuring Persistent Storage](#) and [Persistent Storage Examples](#).

General Storage Configuration Options

```
storage:
  delete:
    enabled: true ❶
  redirect:
    disable: false
  cache:
    blobdescriptor: inmemory
  maintenance:
    uploadpurging:
      enabled: true
      age: 168h
      interval: 24h
      dryrun: false
    readonly:
      enabled: false
```

❶ This entry is **mandatory** for image pruning to work properly.

2.5.5.4. Auth

Auth options should not be altered. The **openshift** extension is the only supported option.

```
auth:
  openshift:
    realm: openshift
```

2.5.5.5. Middleware

The **repository** middleware extension allows to configure OpenShift Container Platform middleware responsible for interaction with OpenShift Container Platform and image proxying.

```
middleware:
  registry:
    - name: openshift ❶
  repository:
    - name: openshift ❷
    options:
      acceptschema2: true ❸
      pullthrough: true ❹
      mirrorpullthrough: true ❺
      enforcequota: false ❻
      projectcachettl: 1m ❼
```

```

    blobrepositorycachettl: 10m 8
  storage:
    - name: openshift 9

```

1 2 9 These entries are mandatory. Their presence ensures required components are loaded. These values should not be changed.

3 Allows you to store [manifest schema v2](#) during a push to the registry. See [below](#) for more details.

4 Allows the registry to act as a proxy for remote blobs. See [below](#) for more details.

5 Allows the registry cache blobs to be served from remote registries for fast access later. The mirroring starts when the blob is accessed for the first time. The option has no effect if the [pullthrough](#) is disabled.

6 Prevents blob uploads exceeding the size limit, which are defined in the targeted project.

7 An expiration timeout for limits cached in the registry. The lower the value, the less time it takes for the limit changes to propagate to the registry. However, the registry will query limits from the server more frequently and, as a consequence, pushes will be slower.

8 An expiration timeout for remembered associations between blob and repository. The higher the value, the higher probability of fast lookup and more efficient registry operation. On the other hand, memory usage will raise as well as a risk of serving image layer to user, who is no longer authorized to access it.

2.5.5.5.1. CloudFront Middleware

The [CloudFront middleware extension](#) can be added to support AWS, CloudFront CDN storage provider. CloudFront middleware speeds up distribution of image content internationally. The blobs are distributed to several edge locations around the world. The client is always directed to the edge with the lowest latency.



NOTE

The [CloudFront middleware extension](#) can be only used with [S3](#) storage. It is utilized only during blob serving. Therefore, only blob downloads can be speeded up, not uploads.

The following is an example of minimal configuration of S3 storage driver with a CloudFront middleware:

```

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  s3: 1
    accesskey: BJKMSZBRESWJQXRWMAEQ
    secretkey: 5ah5I91SNXbeoUXXDasFtadRqOdy62Jz1n0W1goS

```

```

    region: us-east-1
    bucket: docker.myregistry.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
  storage:
    - name: cloudfront ❷
      options:
        baseurl: https://jrpbyn0k5k88bi.cloudfront.net/ ❸
        privatekey: /etc/docker/cloudfront-ABCDEFGHJKLMNOPQRST.pem ❹
        keypairid: ABCDEFGHJKLMNOPQRST ❺
    - name: openshift

```

- ❶ The S3 storage must be configured the same way regardless of CloudFront middleware.
- ❷ The CloudFront storage middleware needs to be listed before OpenShift middleware.
- ❸ The CloudFront base URL. In the AWS management console, this is listed as **Domain Name** of CloudFront distribution.
- ❹ The location of your AWS private key on the filesystem. This must be not confused with Amazon EC2 key pair. See the [AWS documentation](#) on creating CloudFront key pairs for your trusted signers. The file needs to be mounted as a [secret](#) into the registry pod.
- ❺ The ID of your Cloudfront key pair.

2.5.5.5.2. Overriding Middleware Configuration Options

The **middleware** section cannot be overridden using environment variables. There are a few exceptions, however. For example:

```

middleware:
  repository:
    - name: openshift
      options:
        acceptschema2: true ❶
        pullthrough: true ❷
        mirrorpullthrough: true ❸
        enforcequota: false ❹
        projectcachettl: 1m ❺
        blobrepositorycachettl: 10m ❻

```

- ❶ A configuration option that can be overridden by the boolean environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSIFT_ACCEPTSCHEMA2**, which allows for the ability to accept manifest schema v2 on manifest put requests. Recognized values are **true** and **false** (which applies to all the other boolean variables below).

❷

A configuration option that can be overridden by the boolean environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PULLTHROUGH**, which enables a proxy

- 3 A configuration option that can be overridden by the boolean environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_MIRRORPULLTHROUGH**, which instructs registry to mirror blobs locally if serving remote blobs.
- 4 A configuration option that can be overridden by the boolean environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA**, which allows the ability to turn quota enforcement on or off. By default, quota enforcement is off.
- 5 A configuration option that can be overridden by the environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PROJECTCACHETTTL**, specifying an eviction timeout for project quota objects. It takes a valid time duration string (for example, **2m**). If empty, you get the default timeout. If zero (**0m**), caching is disabled.
- 6 A configuration option that can be overridden by the environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_BLOBREPOSITORYCACHETTTL**, specifying an eviction timeout for associations between blob and containing repository. The format of the value is the same as in **projectcachettl** case.

2.5.5.5.3. Image Pullthrough

If enabled, the registry will attempt to fetch requested blob from a remote registry unless the blob exists locally. The remote candidates are calculated from **DockerImage** entries stored in status of the [image stream](#), a client pulls from. All the unique remote registry references in such entries will be tried in turn until the blob is found.

Pullthrough will only occur if an image stream tag exists for the image being pulled. For example, if the image being pulled is **docker-registry.default.svc:5000/yourproject/yourimage:prod** then the registry will look for an image stream tag named **yourimage:prod** in the project **yourproject**. If it finds one, it will attempt to pull the image using the **DockerImageReference** associated with that image stream tag.

When performing pullthrough, the registry will use pull credentials found in the project associated with the image stream tag that is being referenced. This capability also makes it possible for you to pull images that reside on a registry they do not have credentials to access, as long as you have access to the image stream tag that references the image.

You must ensure that your registry has appropriate certificates to trust any external registries you do a pullthrough against. The certificates need to be placed in the **/etc/pki/tls/certs** directory on the pod. You can mount the certificates using a [configuration map](#) or [secret](#). Note that the entire **/etc/pki/tls/certs** directory must be replaced. You must include the new certificates and replace the system certificates in your secret or configuration map that you mount.

Note that by default image stream tags use a reference policy type of **Source** which means that when the image stream reference is resolved to an image pull specification, the specification used will point to the source of the image. For images hosted on external registries, this will be the external registry and as a result the resource will reference and pull the image by the external registry. For example, **registry.access.redhat.com/openshift3/jenkins-2-rhel7** and pullthrough will not apply. To ensure that resources referencing image streams use a pull specification that points to the internal registry, the image stream tag should use a reference policy type of **Local**. More information is available on [Reference Policy](#).

This feature is on by default. However, it can be disabled using a [configuration option](#).

By default, all the remote blobs served this way are stored locally for subsequent faster access unless **mirrorpullthrough** is disabled. The downside of this mirroring feature is an increased storage usage.



NOTE

The mirroring starts when a client tries to fetch at least a single byte of the blob. To pre-fetch a particular image into integrated registry before it is actually needed, you can run the following command:

```
$ oc get imagestreamtag/${IS}:${TAG} -o jsonpath='{
  .image.dockerImageLayers[*].name }' | \
  xargs -n1 -I {} curl -H "Range: bytes=0-1" -u user:${TOKEN} \
  http://${REGISTRY_IP}:${PORT}/v2/default/mysql/blobs/{}"
```



NOTE

This OpenShift Container Platform mirroring feature should not be confused with the upstream [registry pull through cache feature](#), which is a similar but distinct capability.

2.5.5.5.4. Manifest Schema v2 Support

Each image has a manifest describing its blobs, instructions for running it and additional metadata. The manifest is versioned, with each version having different structure and fields as it evolves over time. The same image can be represented by multiple manifest versions. Each version will have different digest though.

The registry currently supports [manifest v2 schema 1](#) (**schema1**) and [manifest v2 schema 2](#) (**schema2**). The former is being obsoleted but will be supported for an extended amount of time.

You should be wary of compatibility issues with various Docker clients:

- Docker clients of version 1.9 or older support only **schema1**. Any manifest this client pulls or pushes will be of this legacy schema.
- Docker clients of version 1.10 support both **schema1** and **schema2**. And by default, it will push the latter to the registry if it supports newer schema.

The registry, storing an image with **schema1** will always return it unchanged to the client. **Schema2** will be transferred unchanged only to newer Docker client. For the older one, it will be converted on-the-fly to **schema1**.

This has significant consequences. For example an image pushed to the registry by a newer Docker client cannot be pulled by the older Docker by its digest. That's because the stored image's manifest is of **schema2** and its digest can be used to pull only this version of manifest.

For this reason, the registry is configured by default not to store **schema2**. This ensures that any docker client will be able to pull from the registry any image pushed there regardless of client's version.

Once you're confident that all the registry clients support **schema2**, you'll be safe to enable its support in the registry. See the [middleware configuration reference](#) above for particular option.

2.5.5.6. OpenShift

This section reviews the configuration of global settings for features specific to OpenShift Container Platform. In a future release, **openshift**-related settings in the [Middleware](#) section will be obsoleted.

Currently, this section allows you to configure registry metrics collection:

```
openshift:
  version: 1.0 ❶
  server:
    addr: docker-registry.default.svc ❷
  metrics:
    enabled: false ❸
    secret: <secret> ❹
  requests:
    read:
      maxrunning: 10 ❺
      maxinqueue: 10 ❻
      maxwaitinqueue 2m ❼
    write:
      maxrunning: 10 ❽
      maxinqueue: 10 ❾
      maxwaitinqueue 2m ❿
```

- ❶ A mandatory entry specifying configuration version of this section. The only supported value is **1.0**.
- ❷ The hostname of the registry. Should be set to the same value configured on the master. It can be overridden by the environment variable **REGISTRY_OPENSIFT_SERVER_ADDR**.
- ❸ Can be set to **true** to enable metrics collection. It can be overridden by the boolean environment variable **REGISTRY_OPENSIFT_METRICS_ENABLED**.
- ❹ A secret used to authorize client requests. Metrics clients must use it as a bearer token in **Authorization** header. It can be overridden by the environment variable **REGISTRY_OPENSIFT_METRICS_SECRET**.
- ❺ Maximum number of simultaneous pull requests. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_READ_MAXRUNNING**. Zero indicates no limit.
- ❻ Maximum number of queued pull requests. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_READ_MAXINQUEUE**. Zero indicates no limit.
- ❼ Maximum time a pull request can wait in the queue before being rejected. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_READ_MAXWAITINQUEUE**. Zero indicates no limit.
- ❽ Maximum number of simultaneous push requests. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_WRITE_MAXRUNNING**. Zero indicates no limit.
- ❾ Maximum number of queued push requests. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_WRITE_MAXINQUEUE**. Zero indicates no limit.
- ❿ Maximum time a push request can wait in the queue before being rejected. It can be overridden by the environment variable **REGISTRY_OPENSIFT_REQUESTS_WRITE_MAXWAITINQUEUE**. Zero indicates no limit.

See [Accessing Registry Metrics](#) for usage information.

2.5.5.7. Reporting

Reporting is unsupported.

2.5.5.8. HTTP

[Upstream options](#) are supported. [Learn how to alter these settings via environment variables](#). Only the **tls** section should be altered. For example:

```
http:
  addr: :5000
  tls:
    certificate: /etc/secrets/registry.crt
    key: /etc/secrets/registry.key
```

2.5.5.9. Notifications

[Upstream options](#) are supported. The [REST API Reference](#) provides more comprehensive integration options.

Example:

```
notifications:
  endpoints:
    - name: registry
      disabled: false
      url: https://url:port/path
      headers:
        Accept:
          - text/plain
      timeout: 500
      threshold: 5
      backoff: 1000
```

2.5.5.10. Redis

Redis is not supported.

2.5.5.11. Health

[Upstream options](#) are supported. The registry deployment configuration provides an integrated health check at **/healthz**.

2.5.5.12. Proxy

Proxy configuration should not be enabled. This functionality is provided by the [OpenShift Container Platform repository middleware extension](#), **pullthrough: true**.

2.5.5.13. Cache

The integrated registry actively caches data to reduce the number of calls to slow external resources. There are two caches:

1. The storage cache that is used to cache blobs metadata. This cache does not have an expiration time and the data is there until it is explicitly deleted.
2. The application cache contains association between blobs and repositories. The data in this cache has an expiration time.

In order to completely turn off the cache, you need to change the configuration:

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache: {} ❶
openshift:
  version: 1.0
  cache:
    disabled: true ❷
    blobrepositoryttl: 10m
```

- ❶ Disables cache of metadata accessed in the storage backend. Without this cache, the registry server will constantly access the backend for metadata.
- ❷ Disables the cache in which contains the blob and repository associations. Without this cache, the registry server will continually re-query the data from the master API and recompute the associations.

2.6. KNOWN ISSUES

2.6.1. Overview

The following are the known issues when deploying or using the integrated registry.

2.6.2. Concurrent Build with Registry Pull-through

The local **docker-registry** deployment takes on additional load. By default, it now caches content from **registry.access.redhat.com**. The images from **registry.access.redhat.com** for STI builds are now stored in the local registry. Attempts to pull them result in pulls from the local **docker-registry**. As a result, there are circumstances where extreme numbers of concurrent builds can result in timeouts for the pulls and the build can possibly fail. To alleviate the issue, scale the **docker-registry** deployment to more than one replica. Check for timeouts in the builder pod's logs.

2.6.3. Image Push Errors with Scaled Registry Using Shared NFS Volume

When using a scaled registry with a shared NFS volume, you may see one of the following errors during the push of an image:

- **digest invalid: provided digest did not match uploaded content**
- **blob upload unknown**
- **blob upload invalid**

These errors are returned by an internal registry service when Docker attempts to push the image. Its cause originates in the synchronization of file attributes across nodes. Factors such as NFS client side caching, network latency, and layer size can all contribute to potential errors that might occur when pushing an image using the default round-robin load balancing configuration.

You can perform the following steps to minimize the probability of such a failure:

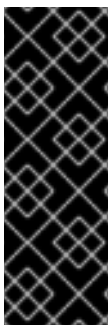
1. Ensure that the **sessionAffinity** of your **docker-registry** service is set to **ClientIP**:

```
$ oc get svc/docker-registry --template='{{.spec.sessionAffinity}}'
```

This should return **ClientIP**, which is the default in recent OpenShift Container Platform versions. If not, change it:

```
$ oc patch svc/docker-registry -p '{"spec":{"sessionAffinity":  
"ClientIP"}}'
```

2. Ensure that the NFS export line of your registry volume on your NFS server has the **no_wdelay** options listed. The **no_wdelay** option prevents the server from delaying writes, which greatly improves read-after-write consistency, a requirement of the registry.



IMPORTANT

Testing shows issues with using the NFS server on RHEL as storage backend for the container registry. This includes the OpenShift Container Registry and Quay. Therefore, using NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift core components.

2.6.4. Pull of Internally Managed Image Fails with "not found" Error

This error occurs when the pulled image is pushed to an image stream different from the one it is being pulled from. This is caused by re-tagging a built image into an arbitrary image stream:

```
$ oc tag srcimagestream:latest anyproject/pullimagestream:latest
```

And subsequently pulling from it, using an image reference such as:

```
internal.registry.url:5000/anyproject/pullimagestream:latest
```

During a manual Docker pull, this will produce a similar error:

```
Error: image anyproject/pullimagestream:latest not found
```

To prevent this, avoid the tagging of internally managed images completely, or re-push the built image to the desired namespace [manually](#).

2.6.5. Image Push Fails with "500 Internal Server Error" on S3 Storage

There are problems reported happening when the registry runs on S3 storage back-end. Pushing to a Docker registry occasionally fails with the following error:

Received unexpected HTTP status: 500 Internal Server Error

To debug this, you need to [view the registry logs](#). In there, look for similar error messages occurring at the time of the failed push:

```
time="2016-03-30T15:01:21.22287816-04:00" level=error msg="unknown error
completing upload: driver.Error{DriverName:\"s3\", Enclosed:(*url.Error)
(0xc20901cea0)}" http.request.method=PUT
...
time="2016-03-30T15:01:21.493067808-04:00" level=error msg="response
completed with error" err.code=UNKNOWN err.detail="s3: Put
https://s3.amazonaws.com/oso-tsi-
docker/registry/docker/registry/v2/blobs/sha256/ab/abe5af443833d60cf672e2a
c57589410dddec060ed725d3e676f1865af63d2e2/data: EOF" err.message="unknown
error" http.request.method=PUT
...
time="2016-04-02T07:01:46.056520049-04:00" level=error msg="error putting
into main store: s3: The request signature we calculated does not match
the signature you provided. Check your key and signing method."
http.request.method=PUT
atest
```

If you see such errors, contact your Amazon S3 support. There may be a problem in your region or with your particular bucket.

2.6.6. Image Pruning Fails

If you encounter the following error when pruning images:

```
BLOB
sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c
error deleting blob
```

And your [registry log](#) contains the following information:

```
error deleting blob
\"sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c\
\": operation unsupported
```

It means that your [custom configuration file](#) lacks mandatory entries in the [storage section](#), namely **storage:delete:enabled** set to **true**. Add them, re-deploy the registry, and repeat your image pruning operation.

CHAPTER 3. SETTING UP A ROUTER

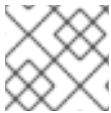
3.1. ROUTER OVERVIEW

3.1.1. About Routers

There are many ways to [get traffic into the cluster](#). The most common approach is to use the OpenShift Container Platform [router](#) as the ingress point for external traffic destined for [services](#) in your OpenShift Container Platform installation.

OpenShift Container Platform provides and supports the following router plug-ins:

- The [HAProxy template router](#) is the default plug-in. It uses the **openshift3/ose-haproxy-router** image to run an HAProxy instance alongside the template router plug-in inside a container on OpenShift Container Platform. It currently supports HTTP(S) traffic and TLS-enabled traffic via SNI. The router's container listens on the host network interface, unlike most containers that listen only on private IPs. The router proxies external requests for route names to the IPs of actual pods identified by the service associated with the route.
- The [F5 router](#) integrates with an existing **F5 BIG-IP®** system in your environment to synchronize routes. **F5 BIG-IP®** version 11.4 or newer is required in order to have the F5 iControl REST API.



NOTE

The F5 router plug-in is available starting in OpenShift Container Platform 3.0.2.

- [Deploying a Default HAProxy Router](#)
- [Deploying a Custom HAProxy Router](#)
- [Deploying a F5 BIG-IP® Router](#)
- [Configuring the HAProxy Router to Use PROXY Protocol](#)
- [Configuring Route Timeouts](#)

3.1.2. Router Service Account

Before deploying an OpenShift Container Platform cluster, you must have a [service account](#) for the router, which is automatically created during cluster installation. This service account has permissions to a [security context constraint](#) (SCC) that allows it to specify host ports.

3.1.2.1. Permission to Access Labels

When [namespace labels](#) are used, for example in creating [router shards](#), the service account for the router must have **cluster-reader** permission.

```
$ oc adm policy add-cluster-role-to-user \
    cluster-reader \
    system:serviceaccount:default:router
```

With a service account in place, you can proceed to installing [a default HAProxy Router](#), [a customized HAProxy Router](#) or [F5 Router](#).

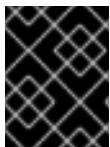
3.2. USING THE DEFAULT HAPROXY ROUTER

3.2.1. Overview

The `oc adm router` command is provided with the administrator CLI to simplify the tasks of setting up routers in a new installation. The `oc adm router` command creates the service and deployment configuration objects. Use the `--service-account` option to specify the service account the router will use to contact the master.

The [router service account](#) can be created in advance or created by the `oc adm router --service-account` command.

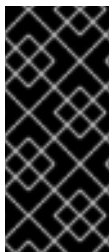
Every form of communication between OpenShift Container Platform components is secured by TLS and uses various certificates and authentication methods. The `--default-certificate` .pem format file can be supplied or one is created by the `oc adm router` command. When [routes](#) are created, the user can provide route certificates that the router will use when handling the route.



IMPORTANT

When deleting a router, ensure the deployment configuration, service, and secret are deleted as well.

Routers are deployed on specific nodes. This makes it easier for the cluster administrator and external network manager to coordinate which IP address will run a router and which traffic the router will handle. The routers are deployed on specific nodes by using [node selectors](#).



IMPORTANT

Routers use host networking by default, and they directly attach to port 80 and 443 on all interfaces on a host. Restrict routers to hosts where ports 80/443 are available and not being consumed by another service, and set this using [node selectors](#) and the [scheduler configuration](#). As an example, you can achieve this by dedicating infrastructure nodes to run services such as routers.

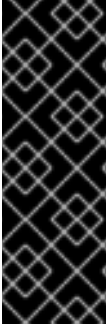


IMPORTANT

It is recommended to use separate distinct **openshift-router** service account with your router. This can be provided using the `--service-account` flag to the `oc adm router` command.

```
$ oc adm router --dry-run --service-account=router 1
```

1 `--service-account` is the name of a [service account](#) for the **openshift-router**.



IMPORTANT

Router pods created using **oc adm router** have default resource requests that a node must satisfy for the router pod to be deployed. In an effort to increase the reliability of infrastructure components, the default resource requests are used to increase the QoS tier of the router pods above pods without resource requests. The default values represent the observed minimum resources required for a basic router to be deployed and can be edited in the routers deployment configuration and you may want to increase them based on the load of the router.

3.2.2. Creating a Router

If the router does not exist, run the following to create a router:

```
$ oc adm router <router_name> --replicas=<number> --service-account=router
```

--replicas is usually **1** unless a [high availability](#) configuration is being created.

To find the host IP address of the router:

```
$ oc get po <router-pod> --template={{.status.hostIP}}
```

You can also [use router shards](#) to ensure that the router is filtered to specific namespaces or routes, or [set any environment variables](#) after router creation. In this case create a router for each shard.

3.2.3. Other Basic Router Commands

Checking the Default Router

The default router service account, named **router**, is automatically created during cluster installations. To verify that this account already exists:

```
$ oc adm router --dry-run --service-account=router
```

Viewing the Default Router

To see what the default router would look like if created:

```
$ oc adm router --dry-run -o yaml --service-account=router
```

Deploying the Router to a Labeled Node

To deploy the router to any node(s) that match a specified [node label](#):

```
$ oc adm router <router_name> --replicas=<number> --selector=<label> \
  --service-account=router
```

For example, if you want to create a router named **router** and have it placed on a node labeled with **node-role.kubernetes.io/infra=true**:

```
$ oc adm router router --replicas=1 --selector='node-
role.kubernetes.io/infra=true' \
  --service-account=router
```

During cluster installation, the **openshift_router_selector** and

openshift_registry_selector Ansible settings are set to **node-role.kubernetes.io/infra=true** by default. The default router and registry will only be automatically deployed if a node exists that matches the **node-role.kubernetes.io/infra=true** label.

For information on updating labels, see [Updating Labels on Nodes](#).

Multiple instances are created on different hosts according to the [scheduler policy](#).

Using a Different Router Image

To use a different router image and view the router configuration that would be used:

```
$ oc adm router <router_name> -o <format> --images=<image> \
  --service-account=router
```

For example:

```
$ oc adm router region-west -o yaml --images=myrepo/somerouter:mytag \
  --service-account=router
```

3.2.4. Filtering Routes to Specific Routers

Using the **ROUTE_LABELS** environment variable, you can filter routes so that they are used only by specific routers.

For example, if you have multiple routers, and 100 routes, you can attach labels to the routes so that a portion of them are handled by one router, whereas the rest are handled by another.

1. After [creating a router](#), use the **ROUTE_LABELS** environment variable to tag the router:

```
$ oc env dc/<router=name> ROUTE_LABELS="key=value"
```

2. Add the label to the desired routes:

```
oc label route <route=name> key=value
```

3. To verify that the label has been attached to the route, check the route configuration:

```
$ oc describe route/<route_name>
```

Setting the Maximum Number of Concurrent Connections

The router can handle a maximum number of 20000 connections by default. You can change that limit depending on your needs. Having too few connections prevents the health check from working, which causes unnecessary restarts. You need to configure the system to support the maximum number of connections. The limits shown in '**sysctl fs.nr_open**' and '**sysctl fs.file-max**' must be large enough. Otherwise, HAproxy will not start.

When the router is created, the **--max-connections=** option sets the desired limit:

```
$ oc adm router --max-connections=10000 ....
```

Edit the `ROUTER_MAX_CONNECTIONS` environment variable in the router's deployment configuration to change the value. The router pods are restarted with the new value. If `ROUTER_MAX_CONNECTIONS` is not present, the default value of 20000, is used.



NOTE

A connection includes the frontend and internal backend. This counts as two connections. Be sure to set `ROUTER_MAX_CONNECTIONS` to double than the number of connections you intend to create.

3.2.5. HAProxy Strict SNI

The `HAProxy strict-sni` can be controlled through the `ROUTER_STRICT_SNI` environment variable in the router's deployment configuration. It can also be set when the router is created by using the `--strict-sni` command line option.

```
$ oc adm router --strict-sni
```

3.2.6. TLS Cipher Suites

Set the router `cipher suite` using the `--ciphers` option when creating a router:

```
$ oc adm router --ciphers=modern ....
```

The values are: `modern`, `intermediate`, or `old`, with `intermediate` as the default. Alternatively, a set of ":" separated ciphers can be provided. The ciphers must be from the set displayed by:

```
$ openssl ciphers
```

Alternatively, use the `ROUTER_CIPHERS` environment variable for an existing router.

3.2.7. Highly-Available Routers

You can [set up a highly-available router](#) on your OpenShift Container Platform cluster using IP failover. This setup has multiple replicas on different nodes so the failover software can switch to another replica if the current one fails.

3.2.8. Customizing the Router Service Ports

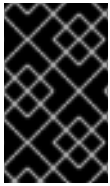
You can customize the service ports that a template router binds to by setting the environment variables `ROUTER_SERVICE_HTTP_PORT` and `ROUTER_SERVICE_HTTPS_PORT`. This can be done by creating a template router, then editing its deployment configuration.

The following example creates a router deployment with `0` replicas and customizes the router service HTTP and HTTPS ports, then scales it appropriately (to `1` replica).

```
$ oc adm router --replicas=0 --ports='10080:10080,10443:10443'
$ oc set env dc/router ROUTER_SERVICE_HTTP_PORT=10080 \
    ROUTER_SERVICE_HTTPS_PORT=10443
$ oc scale dc/router --replicas=1
```

1

- 1 Ensures exposed ports are appropriately set for routers that use the container networking mode -- **host-network=false**.



IMPORTANT

If you do customize the template router service ports, you will also need to ensure that the nodes where the router pods run have those custom ports opened in the firewall (either via Ansible or **iptables**, or any other custom method that you use via **firewall-cmd**).

The following is an example using **iptables** to open the custom router service ports.

```
$ iptables -A INPUT -p tcp --dport 10080 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 10443 -j ACCEPT
```

3.2.9. Working With Multiple Routers

An administrator can create multiple routers with the same definition to serve the same set of routes. Each router will be on a different [node](#) and will have a different IP address. The network administrator will need to get the desired traffic to each node.

Multiple routers can be grouped to distribute routing load in the cluster and separate tenants to different routers or [shards](#). Each router or shard in the group admits routes based on the selectors in the router. An administrator can create shards over the whole cluster using [ROUTE_LABELS](#). A user can create shards over a namespace (project) by using [NAMESPACE_LABELS](#).

3.2.10. Adding a Node Selector to a Deployment Configuration

Making specific routers deploy on specific nodes requires two steps:

1. Add a [label](#) to the desired node:

```
$ oc label node 10.254.254.28 "router=first"
```

2. Add a node selector to the router deployment configuration:

```
$ oc edit dc <deploymentConfigName>
```

Add the **template.spec.nodeSelector** field with a key and value corresponding to the label:

```
...
template:
  metadata:
    creationTimestamp: null
    labels:
      router: router1
  spec:
    nodeSelector:
      router: "first"
...

```

1

- 1 The key and value are **router** and **first**, respectively, corresponding to the **router=first** label.

3.2.11. Using Router Shards

Router sharding uses **NAMESPACE_LABELS** and **ROUTE_LABELS**, to filter router namespaces and routes. This enables you to distribute subsets of routes over multiple router deployments. By using non-overlapping subsets, you can effectively partition the set of routes. Alternatively, you can define shards comprising overlapping subsets of routes.

By default, a router selects all routes from all **projects (namespaces)**. Sharding involves adding labels to routes or namespaces and label selectors to routers. Each router shard comprises the routes that are selected by a specific set of label selectors or belong to the namespaces that are selected by a specific set of label selectors.



NOTE

The router service account must have the [**cluster reader**] permission set to allow access to labels in other namespaces.

Router Sharding and DNS

Because an external DNS server is needed to route requests to the desired shard, the administrator is responsible for making a separate DNS entry for each router in a project. A router will not forward unknown routes to another router.

Consider the following example:

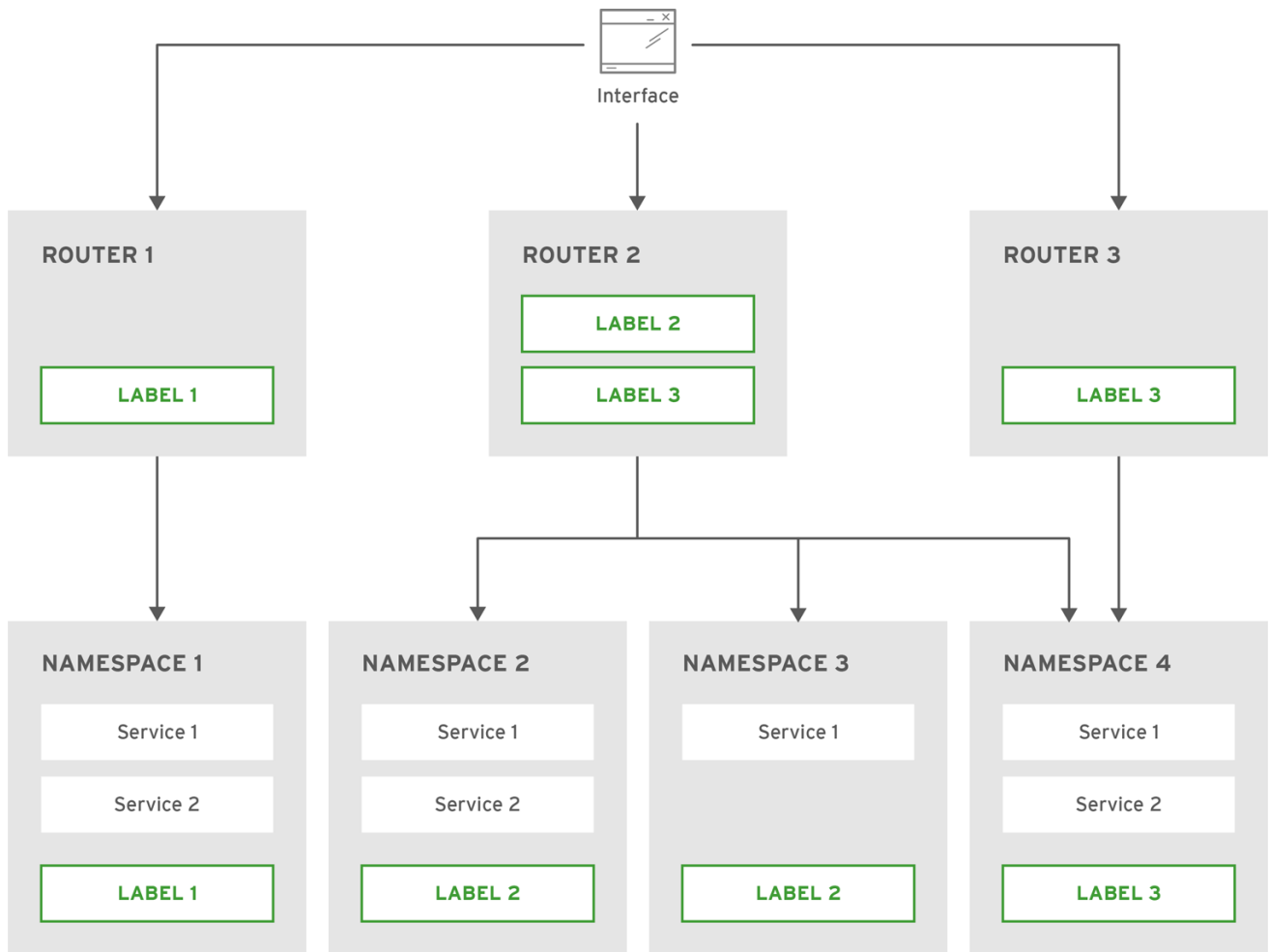
- Router A lives on host 192.168.0.5 and has routes with ***.foo.com**.
- Router B lives on host 192.168.1.9 and has routes with ***.example.com**.

Separate DNS entries must resolve *.foo.com to the node hosting Router A and *.example.com to the node hosting Router B:

- ***.foo.com A IN 192.168.0.5**
- ***.example.com A IN 192.168.1.9**

Router Sharding Examples

This section describes router sharding using namespace and route labels.

Figure 3.1. Router Sharding Based on Namespace Labels

OPENSIFT_415490_0217

1. Configure a router with a namespace label selector:

```
$ oc set env dc/router NAMESPACE_LABELS="router=r1"
```

2. Because the router has a selector on the namespace, the router will handle routes only for matching namespaces. In order to make this selector match a namespace, label the namespace accordingly:

```
$ oc label namespace default "router=r1"
```

3. Now, if you create a route in the default namespace, the route is available in the default router:

```
$ oc create -f route1.yaml
```

4. Create a new project (namespace) and create a route, **route2**:

```
$ oc new-project p1
$ oc create -f route2.yaml
```

Notice the route is not available in your router.

5. Label namespace **p1** with **router=r1**

```
$ oc label namespace p1 "router=r1"
```

Adding this label makes the route available in the router.

Example

A router deployment **finops-router** is configured with the label selector **NAMESPACE_LABELS="name in (finance, ops)"**, and a router deployment **dev-router** is configured with the label selector **NAMESPACE_LABELS="name=dev"**.

If all routes are in namespaces labeled **name=finance**, **name=ops**, and **name=dev**, then this configuration effectively distributes your routes between the two router deployments.

In the above scenario, sharding becomes a special case of partitioning, with no overlapping subsets. Routes are divided between router shards.

The criteria for route selection govern how the routes are distributed. It is possible to have overlapping subsets of routes across router deployments.

Example

In addition to **finops-router** and **dev-router** in the example above, you also have **devops-router**, which is configured with a label selector **NAMESPACE_LABELS="name in (dev, ops)"**. The routes in namespaces labeled **name=dev** or **name=ops** now are serviced by two different router deployments. This becomes a case in which you have defined overlapping subsets of routes, as illustrated in the procedure in [Router Sharding Based on Namespace Labels](#).

In addition, this enables you to create more complex routing rules, allowing the diversion of higher priority traffic to the dedicated **finops-router** while sending lower priority traffic to **devops-router**.

Router Sharding Based on Route Labels

NAMESPACE_LABELS allows filtering of the projects to service and selecting all the routes from those projects, but you may want to partition routes based on other criteria associated with the routes themselves. The **ROUTE_LABELS** selector allows you to slice-and-dice the routes themselves.

Example

A router deployment **prod-router** is configured with the label selector **ROUTE_LABELS="mydeployment=prod"**, and a router deployment **devtest-router** is configured with the label selector **ROUTE_LABELS="mydeployment in (dev, test)"**. This configuration partitions routes between the two router deployments according to the routes' labels, irrespective of their namespaces.

The example assumes you have all the routes you want to be serviced tagged with a label **"mydeployment=<tag>"**.

3.2.11.1. Creating Router Shards

This section describes an advanced example of router sharding. Suppose there are 26 routes, named **a** — **z**, with various labels:

Possible labels on routes

sla=high	geo=east	hw=modest	dept=finance
sla=medium	geo=west	hw=strong	dept=dev
sla=low			dept=ops

These labels express the concepts including service level agreement, geographical location, hardware requirements, and department. The routes can have at most one label from each column. Some routes may have other labels or no labels at all.

Name(s)	SLA	Geo	HW	Dept	Other Labels
a	high	east	modest	finance	type=static
b		west	strong		type=dynamic
c, d, e	low		modest		type=static
g—k	medium		strong	dev	
l—s	high		modest	ops	
t—z		west			type=dynamic

Here is a convenience script **mkshard** that illustrates how **oc adm router**, **oc set env**, and **oc scale** can be used together to make a router shard.

```
#!/bin/bash
# Usage: mkshard ID SELECTION-EXPRESSION
id=$1
sel="$2"
router=router-shard-$id
oc adm router $router --replicas=0
dc=dc/router-shard-$id
oc set env $dc ROUTE_LABELS="$sel"
oc scale $dc --replicas=3
```

- 1 The created router has name **router-shard-<id>**.
- 2 Specify no scaling for now.
- 3 The deployment configuration for the router.
- 4 Set the selection expression using **oc set env**. The selection expression is the value of the **ROUTE_LABELS** environment variable.
- 5 Scale it up.

Running **mkshard** several times creates several routers:

Router	Selection Expression	Routes
router-shard-1	sla=high	a, l — s
router-shard-2	geo=west	b, t — z
router-shard-3	dept=dev	g — k

3.2.11.2. Modifying Router Shards

Because a router shard is a construct [based on labels](#), you can modify either the labels (via **oc label**) or the selection expression (via **oc set env**).

This section extends the example started in the [Creating Router Shards](#) section, demonstrating how to change the selection expression.

Here is a convenience script **modshard** that modifies an existing router to use a new selection expression:

```
#!/bin/bash
# Usage: modshard ID SELECTION-EXPRESSION...
id=$1
shift
router=router-shard-$id
dc=dc/$router
oc scale $dc --replicas=0
oc set env $dc "$@"
oc scale $dc --replicas=3
```

- 1 The modified router has name **router-shard-<id>**.
- 2 The deployment configuration where the modifications occur.
- 3 Scale it down.
- 4 Set the new selection expression using **oc set env**. Unlike **mkshard** from the [Creating Router Shards](#) section, the selection expression specified as the non-**ID** arguments to **modshard** must include the environment variable name as well as its value.
- 5 Scale it back up.



NOTE

In **modshard**, the **oc scale** commands are not necessary if the [deployment strategy](#) for **router-shard-<id>** is **Rolling**.

For example, to expand the department for **router-shard-3** to include **ops** as well as **dev**:

```
$ modshard 3 ROUTE_LABELS='dept in (dev, ops)'
```

The result is that **router-shard-3** now selects routes **g — s** (the combined sets of **g — k** and **l — s**).

This example takes into account that there are only three departments in this example scenario, and specifies a department to leave out of the shard, thus achieving the same result as the preceding example:

```
$ modshard 3 ROUTE_LABELS='dept != finance'
```

This example specifies three comma-separated qualities, and results in only route **b** being selected:

```
$ modshard 3 ROUTE_LABELS='hw=strong,type=dynamic,geo=west'
```

Similarly to **ROUTE_LABELS**, which involves a route's labels, you can select routes based on the labels of the route's namespace using the **NAMESPACE_LABELS** environment variable. This example modifies **router-shard-3** to serve routes whose namespace has the label **frequency=weekly**:

```
$ modshard 3 NAMESPACE_LABELS='frequency=weekly'
```

The last example combines **ROUTE_LABELS** and **NAMESPACE_LABELS** to select routes with label **sla=low** and whose namespace has the label **frequency=weekly**:

```
$ modshard 3 \
  NAMESPACE_LABELS='frequency=weekly' \
  ROUTE_LABELS='sla=low'
```

3.2.12. Finding the Host Name of the Router

When exposing a service, a user can use the same route from the DNS name that external users use to access the application. The network administrator of the external network must make sure the host name resolves to the name of a router that has admitted the route. The user can set up their DNS with a CNAME that points to this host name. However, the user may not know the host name of the router. When it is not known, the cluster administrator can provide it.

The cluster administrator can use the **--router-canonical-hostname** option with the router's canonical host name when creating the router. For example:

```
# oc adm router myrouter --router-canonical-hostname="rtr.example.com"
```

This creates the **ROUTER_CANONICAL_HOSTNAME** environment variable in the router's deployment configuration containing the host name of the router.

For routers that already exist, the cluster administrator can edit the router's deployment configuration and add the **ROUTER_CANONICAL_HOSTNAME** environment variable:

```
spec:
  template:
    spec:
      containers:
      - env:
        - name: ROUTER_CANONICAL_HOSTNAME
          value: rtr.example.com
```

The **ROUTER_CANONICAL_HOSTNAME** value is displayed in the route status for all routers that have

admitted the route. The route status is refreshed every time the router is reloaded.

When a user creates a route, all of the active routers evaluate the route and, if conditions are met, admit it. When a router that defines the **ROUTER_CANONICAL_HOSTNAME** environment variable admits the route, the router places the value in the **routerCanonicalHostname** field in the route status. The user can examine the route status to determine which, if any, routers have admitted the route, select a router from the list, and find the host name of the router to pass along to the network administrator.

```
status:
  ingress:
    conditions:
      lastTransitionTime: 2016-12-07T15:20:57Z
      status: "True"
      type: Admitted
      host: hello.in.mycloud.com
      routerCanonicalHostname: rtr.example.com
      routerName: myrouter
      wildcardPolicy: None
```

oc describe includes the host name when available:

```
$ oc describe route/hello-route3
...
Requested Host: hello.in.mycloud.com exposed on router myroute (host
rtr.example.com) 12 minutes ago
```

Using the above information, the user can ask the DNS administrator to set up a CNAME from the route's host, **hello.in.mycloud.com**, to the router's canonical hostname, **rtr.example.com**. This results in any traffic to **hello.in.mycloud.com** reaching the user's application.

3.2.13. Customizing the Default Routing Subdomain

You can customize the suffix used as the default routing subdomain for your environment by modifying the [master configuration file](#) (the **/etc/origin/master/master-config.yaml** file by default). Routes that do not specify a host name would have one generated using this default routing subdomain.

The following example shows how you can set the configured suffix to **v3.openshift.test**:

```
routingConfig:
  subdomain: v3.openshift.test
```



NOTE

This change requires a restart of the master if it is running.

With the OpenShift Container Platform master(s) running the above configuration, the [generated host name](#) for the example of a route named **no-route-hostname** without a host name added to a namespace **mynamespace** would be:

```
no-route-hostname-mynamespace.v3.openshift.test
```

3.2.14. Forcing Route Host Names to a Custom Routing Subdomain

If an administrator wants to restrict all routes to a specific routing subdomain, they can pass the **--force-subdomain** option to the **oc adm router** command. This forces the router to override any host names specified in a route and generate one based on the template provided to the **--force-subdomain** option.

The following example runs a router, which overrides the route host names using a custom subdomain template **\${name}-\${namespace}.apps.example.com**.

```
$ oc adm router --force-subdomain='${name}-${namespace}.apps.example.com'
```

3.2.15. Using Wildcard Certificates

A TLS-enabled route that does not include a certificate uses the router's default certificate instead. In most cases, this certificate should be provided by a trusted certificate authority, but for convenience you can use the OpenShift Container Platform CA to create the certificate. For example:

```
$ CA=/etc/origin/master
$ oc adm ca create-server-cert --signer-cert=$CA/ca.crt \
  --signer-key=$CA/ca.key --signer-serial=$CA/ca.serial.txt \
  --hostnames='*.cloudapps.example.com' \
  --cert=cloudapps.crt --key=cloudapps.key
```

NOTE

The **oc adm ca create-server-cert** command generates a certificate that is valid for two years. This can be altered with the **--expire-days** option, but for security reasons, it is recommended to not make it greater than this value.

Run **oc adm** commands only from the first master listed in the Ansible host inventory file, by default **/etc/ansible/hosts**.

The router expects the certificate and key to be in PEM format in a single file:

```
$ cat cloudapps.crt cloudapps.key $CA/ca.crt > cloudapps.router.pem
```

From there you can use the **--default-cert** flag:

```
$ oc adm router --default-cert=cloudapps.router.pem --service-account=router
```

NOTE

Browsers only consider wildcards valid for subdomains one level deep. So in this example, the certificate would be valid for *a.cloudapps.example.com* but not for *a.b.cloudapps.example.com*.

3.2.16. Manually Redeploy Certificates

To manually redeploy the router certificates:

1. Check to see if a secret containing the default router certificate was added to the router:

■


```
$ oc volumes dc/router
deploymentconfigs/router
secret/router-certs as server-certificate
mounted at /etc/pki/tls/private
```

If the certificate is added, skip the following step and overwrite the secret.

2. Make sure that you have a default certificate directory set for the following variable **DEFAULT_CERTIFICATE_DIR**:

```
$ oc env dc/router --list
DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

If not, create the directory using the following command:

```
$ oc env dc/router DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

3. Export the certificate to PEM format:

```
$ cat custom-router.key custom-router.crt custom-ca.crt > custom-
router.crt
```

4. Overwrite or create a router certificate secret:

If the certificate secret was added to the router, overwrite the secret. If not, create a new secret.

To overwrite the secret, run the following command:

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-
router.crt --from-file=tls.key=custom-router.key --
type=kubernetes.io/tls -o json --dry-run | oc replace -f -
```

To create a new secret, run the following commands:

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-
router.crt --from-file=tls.key=custom-router.key --
type=kubernetes.io/tls

$ oc volume dc/router --add --mount-path=/etc/pki/tls/private --
secret-name='router-certs' --name router-certs
```

5. Deploy the router.

```
$ oc rollout latest dc/router
```

3.2.17. Using Secured Routes

Currently, password protected key files are not supported. HAProxy prompts for a password upon starting and does not have a way to automate this process. To remove a passphrase from a keyfile, you can run:

```
# openssl rsa -in <passwordProtectedKey.key> -out <new.key>
```

Here is an example of how to use a secure edge terminated route with TLS termination occurring on the router before traffic is proxied to the destination. The secure edge terminated route specifies the TLS certificate and key information. The TLS certificate is served by the router front end.

First, start up a router instance:

```
# oc adm router --replicas=1 --service-account=router
```

Next, create a private key, csr and certificate for our edge secured route. The instructions on how to do that would be specific to your certificate authority and provider. For a simple self-signed certificate for a domain named **www.example.test**, see the example shown below:

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=www.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr \
  -signkey example-test.key -out example-test.crt
```

Generate a route using the above certificate and key.

```
$ oc create route edge --service=my-service \
  --hostname=www.example.test \
  --key=example-test.key --cert=example-test.crt
route "my-service" created
```

Look at its definition.

```
$ oc get route/my-service -o yaml
apiVersion: v1
kind: Route
metadata:
  name: my-service
spec:
  host: www.example.test
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: |
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

Make sure your DNS entry for **www.example.test** points to your router instance(s) and the route to your domain should be available. The example below uses curl along with a local resolver to simulate the DNS lookup:

```
# routerip="4.1.1.1" # replace with IP address of one of your router
instances.
# curl -k --resolve www.example.test:443:$routerip
https://www.example.test/
```

3.2.18. Using Wildcard Routes (for a Subdomain)

The HAProxy router has support for wildcard routes, which are enabled by setting the **ROUTER_ALLOW_WILDCARD_ROUTES** environment variable to **true**. Any routes with a wildcard policy of **Subdomain** that pass the router admission checks will be serviced by the HAProxy router. Then, the HAProxy router exposes the associated service (for the route) per the route's wildcard policy.



IMPORTANT

To change a route's wildcard policy, you must remove the route and recreate it with the updated wildcard policy. Editing only the route's wildcard policy in a route's **.yaml** file does not work.

```
$ oc adm router --replicas=0 ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

[Learn how to configure the web console for wildcard routes.](#)

Using a Secure Wildcard Edge Terminated Route

This example reflects TLS termination occurring on the router before traffic is proxied to the destination. Traffic sent to any hosts in the subdomain **example.org** (***.example.org**) is proxied to the exposed service.

The secure edge terminated route specifies the TLS certificate and key information. The TLS certificate is served by the router front end for all hosts that match the subdomain (***.example.org**).

1. Start up a router instance:

```
$ oc adm router --replicas=0 --service-account=router
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

2. Create a private key, certificate signing request (CSR), and certificate for the edge secured route.

The instructions on how to do this are specific to your certificate authority and provider. For a simple self-signed certificate for a domain named ***.example.test**, see this example:

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr \
    -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=*.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr \
    -signkey example-test.key -out example-test.crt
```

3. Generate a wildcard route using the above certificate and key:

```
$ cat > route.yaml <<EOF
apiVersion: v1
kind: Route
metadata:
  name: my-service
spec:
  host: www.example.test
  wildcardPolicy: Subdomain
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: "$(perl -pe 's/\n/\\n/' example-test.key)"
    certificate: "$(perl -pe 's/\n/\\n/' example-test.cert)"
EOF
$ oc create -f route.yaml
```

Ensure your DNS entry for ***.example.test** points to your router instance(s) and the route to your domain is available.

This example uses **curl** with a local resolver to simulate the DNS lookup:

```
# routerip="4.1.1.1" # replace with IP address of one of your
router instances.
# curl -k --resolve www.example.test:443:$routerip
https://www.example.test/
# curl -k --resolve abc.example.test:443:$routerip
https://abc.example.test/
# curl -k --resolve anyname.example.test:443:$routerip
https://anyname.example.test/
```

For routers that allow wildcard routes (**ROUTER_ALLOW_WILDCARD_ROUTES** set to **true**), there are some caveats to the ownership of a subdomain associated with a wildcard route.

Prior to wildcard routes, ownership was based on the claims made for a host name with the namespace with the oldest route winning against any other claimants. For example, route **r1** in namespace **ns1** with a claim for **one.example.test** would win over another route **r2** in namespace **ns2** for the same host name **one.example.test** if route **r1** was older than route **r2**.

In addition, routes in other namespaces were allowed to claim non-overlapping hostnames. For example, route **rone** in namespace **ns1** could claim **www.example.test** and another route **rtwo** in namespace **ns2** could claim **c3po.example.test**.

This is still the case if there are *no* wildcard routes claiming that same subdomain (**example.test** in the above example).

However, a wildcard route needs to claim all of the host names within a subdomain (host names of the form ***.example.test**). A wildcard route's claim is allowed or denied based on whether or not the oldest route for that subdomain (**example.test**) is in the same namespace as the wildcard route. The oldest route can be either a regular route or a wildcard route.

For example, if there is already a route **eldest** that exists in the **ns1** namespace that claimed a host

named **owner.example.test** and, if at a later point in time, a new wildcard route **wildthing** requesting for routes in that subdomain (**example.test**) is added, the claim by the wildcard route will *only* be allowed if it is the same namespace (**ns1**) as the owning route.

The following examples illustrate various scenarios in which claims for wildcard routes will succeed or fail.

In the example below, a router that allows wildcard routes will allow non-overlapping claims for hosts in the subdomain **example.test** as long as a wildcard route has not claimed a subdomain.

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test
$ oc expose service myservice --hostname=bname.example.test

$ oc project ns2
$ oc expose service anotherservice --hostname=second.example.test
$ oc expose service anotherservice --hostname=cname.example.test

$ oc project othersns
$ oc expose service thirdservice --hostname=emmy.example.test
$ oc expose service thirdservice --hostname=webby.example.test
```

In the example below, a router that allows wildcard routes will not allow the claim for **owner.example.test** or **aname.example.test** to succeed since the owning namespace is **ns1**.

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test

$ oc project ns2
$ oc expose service secondservice --hostname=bname.example.test
$ oc expose service secondservice --hostname=cname.example.test

$ # Router will not allow this claim with a different path name `/p1` as
$ # namespace `ns1` has an older route claiming host `aname.example.test`.
$ oc expose service secondservice --hostname=aname.example.test --
path="/p1"

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `owner.example.test`.
$ oc expose service secondservice --hostname=owner.example.test

$ oc project othersns

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `aname.example.test`.
$ oc expose service thirdservice --hostname=aname.example.test
```

In the example below, a router that allows wildcard routes will allow the claim for ``*.example.test`` to succeed since the owning namespace is **ns1** and the wildcard route belongs to that same namespace.

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml # router will allow this claim.
```

In the example below, a router that allows wildcard routes will not allow the claim for ``*.example.test`` to succeed since the owning namespace is **ns1** and the wildcard route belongs to another namespace **cyclone**.

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Switch to a different namespace/project.
$ oc project cyclone

$ # Reusing the route.yaml from a prior example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml # router will deny (_NOT_ allow) this claim.
```

Similarly, once a namespace with a wildcard route claims a subdomain, only routes within that namespace can claim any hosts in that same subdomain.

In the example below, once a route in namespace **ns1** with a wildcard route claims subdomain **example.test**, only routes in the namespace **ns1** are allowed to claim any hosts in that same subdomain.

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ oc project othersns

$ # namespace `othersns` is allowed to claim for other.example.test
$ oc expose service otherservice --hostname=other.example.test
```

```

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml # Router will allow this claim.

$ # In addition, route in namespace othersns will lose its claim to host
$ # `other.example.test` due to the wildcard route claiming the
subdomain.

$ # namespace `ns1` is allowed to claim for deux.example.test
$ oc expose service mysecondservice --hostname=deux.example.test

$ # namespace `ns1` is allowed to claim for deux.example.test with path
/p1
$ oc expose service mythirdservice --hostname=deux.example.test --
path="/p1"

$ oc project othersns

$ # namespace `othersns` is not allowed to claim for deux.example.test
$ # with a different path '/otherpath'
$ oc expose service otherservice --hostname=deux.example.test --
path="/otherpath"

$ # namespace `othersns` is not allowed to claim for owner.example.test
$ oc expose service yetanotherservice --hostname=owner.example.test

$ # namespace `othersns` is not allowed to claim for unclaimed.example.test
$ oc expose service yetanotherservice --hostname=unclaimed.example.test

```

In the example below, different scenarios are shown, in which the owner routes are deleted and ownership is passed within and across namespaces. While a route claiming host **eldest.example.test** in the namespace **ns1** exists, wildcard routes in that namespace can claim subdomain **example.test**. When the route for host **eldest.example.test** is deleted, the next oldest route **senior.example.test** would become the oldest route and would not affect any other routes. Once the route for host **senior.example.test** is deleted, the next oldest route **junior.example.test** becomes the oldest route and block the wildcard route claimant.

```

$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=eldest.example.test
$ oc expose service seniorservice --hostname=senior.example.test

$ oc project othersns

$ # namespace `othersns` is allowed to claim for other.example.test
$ oc expose service juniorservice --hostname=junior.example.test

$ oc project ns1

```

```

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml # Router will allow this claim.

$ # In addition, route in namespace othersns will lose its claim to host
$ # `junior.example.test` due to the wildcard route claiming the
$ # subdomain.

$ # namespace `ns1` is allowed to claim for dos.example.test
$ oc expose service mysecondservice --hostname=dos.example.test

$ # Delete route for host `eldest.example.test`, the next oldest route is
$ # the one claiming `senior.example.test`, so route claims are
$ # unaffected.
$ oc delete route myservice

$ # Delete route for host `senior.example.test`, the next oldest route is
$ # the one claiming `junior.example.test` in another namespace, so claims
$ # for a wildcard route would be affected. The route for the host
$ # `dos.example.test` would be unaffected as there are no other wildcard
$ # claimants blocking it.
$ oc delete route seniorservice

```

3.2.19. Using the Container Network Stack

The OpenShift Container Platform router runs inside a container and the default behavior is to use the network stack of the host (i.e., the node where the router container runs). This default behavior benefits performance because network traffic from remote clients does not need to take multiple hops through user space to reach the target service and container.

Additionally, this default behavior enables the router to get the actual source IP address of the remote connection rather than getting the node's IP address. This is useful for defining ingress rules based on the originating IP, supporting sticky sessions, and monitoring traffic, among other uses.

This host network behavior is controlled by the **--host-network** router command line option, and the default behaviour is the equivalent of using **--host-network=true**. If you wish to run the router with the container network stack, use the **--host-network=false** option when creating the router. For example:

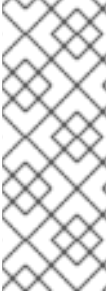
```
$ oc adm router --service-account=router --host-network=false
```

Internally, this means the router container must publish the 80 and 443 ports in order for the external network to communicate with the router.



NOTE

Running with the container network stack means that the router sees the source IP address of a connection to be the NATed IP address of the node, rather than the actual remote IP address.



NOTE

On OpenShift Container Platform clusters using [multi-tenant network isolation](#), routers on a non-default namespace with the `--host-network=false` option will load all routes in the cluster, but routes across the namespaces will not be reachable due to network isolation. With the `--host-network=true` option, routes bypass the container network and it can access any pod in the cluster. If isolation is needed in this case, then do not add routes across the namespaces.

3.2.20. Exposing Router Metrics

The [HAProxy router metrics](#) are, by default, exposed or published in [Prometheus format](#) for consumption by external metrics collection and aggregation systems (e.g. Prometheus, statsd). Metrics are also available directly from the [HAProxy router](#) in its own HTML format for viewing in a browser or CSV download. These metrics include the HAProxy native metrics and some controller metrics.

When you create a router using the following command, OpenShift Container Platform makes metrics available in Prometheus format on the stats port, by default 1936.

```
$ oc adm router --service-account=router
```

- To extract the raw statistics in Prometheus format run the following command:

```
curl <user>:<password>@<router_IP>:<STATS_PORT>
```

For example:

```
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
```

You can get the information you need to access the metrics from the router service annotations:

```
$ oc edit router service <router-service-name>

apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "1936"
    prometheus.io/scrape: "true"
    prometheus.openshift.io/password: IImoDqON02
    prometheus.openshift.io/username: admin
```

The **prometheus.io/port** is the stats port, by default 1936. You might need to configure your firewall to permit access. Use the previous user name and password to access the metrics. The path is **/metrics**.

```
$ curl <user>:<password>@<router_IP>:<STATS_PORT>
for example:
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
...
# HELP haproxy_backend_connections_total Total number of
connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default"
```

```
,route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default"
,route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default"
,route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked
and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming
bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming
bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="f
e_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="f
e_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-
registry-5-nk5fz",route="docker-
registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-
vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-
svc-1"} 0
...
```

- To get metrics in a browser:

1. Delete the following [environment variables](#) from the router deployment configuration file:

```
$ oc edit service router

- name: ROUTER_LISTEN_ADDR
  value: 0.0.0.0:1936
- name: ROUTER_METRICS_TYPE
  value: haproxy
```

2. Launch the stats window using the following URL in a browser, where the **STATS_PORT** value is **1936** by default:

```
http://admin:<Password>@<router_IP>:<STATS_PORT>
```

You can get the stats in CSV format by adding **;csv** to the URL:

For example:

```
http://admin:<Password>@<router_IP>:1936;csv
```

To get the router IP, admin name, and password:

```
oc describe pod <router_pod>
```

- To suppress metrics collection:

```
$ oc adm router --service-account=router --stats-port=0
```

== ARP Cache Tuning for Large-scale Clusters

In OpenShift Container Platform clusters with large numbers of routes (greater than the value of **net.ipv4.neigh.default.gc_thresh3**, which is **65536** by default), you must increase the default values of sysctl variables on each node in the cluster running the router pod to allow more entries in the ARP cache.

When the problem is occurring, the kernel messages would be similar to the following:

```
[ 1738.811139] net_ratelimit: 1045 callbacks suppressed
[ 1743.823136] net_ratelimit: 293 callbacks suppressed
```

When this issue occurs, the **oc** commands might start to fail with the following error:

```
Unable to connect to the server: dial tcp: lookup <hostname> on <ip>:
<port>: write udp <ip>:<port>-><ip>:<port>: write: invalid argument
```

To verify the actual amount of ARP entries for IPv4, run the following:

```
# ip -4 neigh show nud all | wc -l
```

If the number begins to approach the **net.ipv4.neigh.default.gc_thresh3** threshold, increase the values. Get the current value by running:

```
# sysctl net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh1 = 128
# sysctl net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh2 = 512
# sysctl net.ipv4.neigh.default.gc_thresh3
net.ipv4.neigh.default.gc_thresh3 = 1024
```

The following sysctl sets the variables to the OpenShift Container Platform current default values.

```
# sysctl net.ipv4.neigh.default.gc_thresh1=8192
# sysctl net.ipv4.neigh.default.gc_thresh2=32768
# sysctl net.ipv4.neigh.default.gc_thresh3=65536
```

To make these settings permanent, [see this document](#).

3.2.21. Protecting Against DDoS Attacks

Add **timeout http-request** to the default HAProxy router image to protect the deployment against distributed denial-of-service (DDoS) attacks (for example, slowloris):

```
# and the haproxy stats socket is available at /var/run/haproxy.stats
```

```

global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s
  timeout server 10s
  timeout client 30s

```

- 1** **timeout http-request** is set up to 5 seconds. HAProxy gives a client 5 seconds *to send its whole HTTP request. Otherwise, HAProxy shuts the connection with *an error.

Also, when the environment variable **ROUTER_SLOWLORIS_TIMEOUT** is set, it limits the amount of time a client has to send the whole HTTP request. Otherwise, HAProxy will shut down the connection.

Setting the environment variable allows information to be captured as part of the router's deployment configuration and does not require manual modification of the template, whereas manually adding the HAProxy setting requires you to rebuild the router pod and maintain your router template file.

Using annotations implements basic DDoS protections in the HAProxy template router, including the ability to limit the:

- number of concurrent TCP connections
- rate at which a client can request TCP connections
- rate at which HTTP requests can be made

These are enabled on a per route basis because applications can have extremely different traffic patterns.

Table 3.1. HAProxy Template Router Settings

Setting	Description
haproxy.router.openshift.io/rate-limit-connections	Enables the settings be configured (when set to true , for example).
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	The number of concurrent TCP connections that can be made by the same IP address on this route.
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	The number of TCP connections that can be opened by a client IP.
haproxy.router.openshift.io/rate-limit-connections.rate-http	The number of HTTP requests that a client IP can make in a 3-second period.

3.3. DEPLOYING A CUSTOMIZED HAPROXY ROUTER

3.3.1. Overview

The default HAProxy router is intended to satisfy the needs of most users. However, it does not expose all of the capability of HAProxy. Therefore, users may need to modify the router for their own needs.

You may need to implement new features within the application back-ends, or modify the current operation. The router plug-in provides all the facilities necessary to make this customization.

The router pod uses a template file to create the needed HAProxy configuration file. The template file is a [golang template](#). When processing the template, the router has access to OpenShift Container Platform information, including the router's deployment configuration, the set of admitted routes, and some helper functions.

When the router pod starts, and every time it reloads, it creates an HAProxy configuration file, and then it starts HAProxy. The [HAProxy configuration manual](#) describes all of the features of HAProxy and how to construct a valid configuration file.

A [configMap](#) can be used to add the new template to the router pod. With this approach, the router deployment configuration is modified to mount the **configMap** as a volume in the router pod. The **TEMPLATE_FILE** environment variable is set to the full path name of the template file in the router pod.

Alternatively, you can build a custom router image and use it when deploying some or all of your routers. There is no need for all routers to run the same image. To do this, modify the **haproxy-template.config** file, and [rebuild](#) the router image. The new image is pushed to the cluster's Docker repository, and the router's deployment configuration **image:** field is updated with the new name. When the cluster is updated, the image needs to be rebuilt and pushed.

In either case, the router pod starts with the template file.

3.3.2. Obtaining the Router Configuration Template

The HAProxy template file is fairly large and complex. For some changes, it may be easier to modify the existing template rather than writing a complete replacement. You can obtain a **haproxy-config.template** file from a running router by running this on master, referencing the router pod:

```
# oc get po
NAME                                READY    STATUS    RESTARTS   AGE
router-2-40fc3                      1/1      Running   0           11d
# oc rsh router-2-40fc3 cat haproxy-config.template > haproxy-
config.template
# oc rsh router-2-40fc3 cat haproxy.config > haproxy.config
```

Alternatively, you can log onto the node that is running the router:

```
# docker run --rm --interactive=true --tty --entrypoint=cat \
    registry.access.redhat.com/openshift3/ose-haproxy-router:v3.7 haproxy-
config.template
```

The image name is from **docker images**.

Save this content to a file for use as the basis of your customized template. The saved **haproxy.config** shows what is actually running.

3.3.3. Modifying the Router Configuration Template

3.3.3.1. Background

The template is based on the [golang template](#). It can reference any of the environment variables in the router's deployment configuration, any configuration information that is described below, and router provided helper functions.

The structure of the template file mirrors the resulting HAProxy configuration file. As the template is processed, anything not surrounded by `{{" something "}}` is directly copied to the configuration file. Passages that are surrounded by `{{" something "}}` are evaluated. The resulting text, if any, is copied to the configuration file.

3.3.3.2. Go Template Actions

The **define** action names the file that will contain the processed template.

```
{{define "/var/lib/haproxy/conf/haproxy.config"}}pipeline{{end}}
```

Table 3.2. Template Router Functions

Function	Meaning
processEndpointsForAlias(alias ServiceAliasConfig, svc ServiceUnit, action string) []Endpoint	Returns the list of valid endpoints. When action is "shuffle", the order of endpoints is randomized.
env(variable, default ...string) string	Tries to get the named environment variable from the pod. If it is not defined or empty, it returns the optional second argument. Otherwise, it returns an empty string.
matchPattern(pattern, s string) bool	The first argument is a string that contains the regular expression, the second argument is the variable to test. Returns a Boolean value indicating whether the regular expression provided as the first argument matches the string provided as the second argument.
isInteger(s string) bool	Determines if a given variable is an integer.
firstMatch(s string, allowedValues ...string) bool	Compares a given string to a list of allowed strings. Returns first match scanning left to right through the list.
matchValues(s string, allowedValues ...string) bool	Compares a given string to a list of allowed strings. Returns "true" if the string is an allowed value, otherwise returns false.
generateRouteRegexp(hostname, path string, wildcard bool) string	Generates a regular expression matching the route hosts (and paths). The first argument is the host name, the second is the path, and the third is a wildcard Boolean.
genCertificateHostName(hostname string, wildcard bool) string	Generates host name to use for serving/matching certificates. First argument is the host name and the second is the wildcard Boolean.

Function	Meaning
<code>isTrue(s string) bool</code>	Determines if a given variable contains "true".

These functions are provided by the HAProxy template router plug-in.

3.3.3.3. Router Provided Information

This section reviews the OpenShift Container Platform information that the router makes available to the template. The router configuration parameters are the set of data that the HAProxy router plug-in is given. The fields are accessed by **(dot)** `.Fieldname`.

The tables below the Router Configuration Parameters expand on the definitions of the various fields. In particular, `.State` has the set of admitted routes.

Table 3.3. Router Configuration Parameters

Field	Type	Description
<code>WorkingDir</code>	string	The directory that files will be written to, defaults to <code>/var/lib/containers/router</code>
<code>State</code>	<code>map[string](ServiceAliasConfig)`</code>	The routes.
<code>ServiceUnits</code>	<code>map[string]ServiceUnit</code>	The service lookup.
<code>DefaultCertificate</code>	string	Full path name to the default certificate in pem format.
<code>PeerEndpoints</code>	<code>`[]Endpoint</code>	Peers.
<code>StatsUser</code>	string	User name to expose stats with (if the template supports it).
<code>StatsPassword</code>	string	Password to expose stats with (if the template supports it).
<code>StatsPort</code>	int	Port to expose stats with (if the template supports it).
<code>BindPorts</code>	bool	Whether the router should bind the default ports.

Table 3.4. Router ServiceAliasConfig (A Route)

Field	Type	Description
Name	string	The user-specified name of the route.
Namespace	string	The namespace of the route.
Host	string	The host name. For example, www.example.com .
Path	string	Optional path. For example, www.example.com/myservice where myservice is the path.
TLSTermination	routeapi.TLSTerminationType	The termination policy for this back-end; drives the mapping files and router configuration.
Certificates	map[string]Certificate	Certificates used for securing this back-end. Keyed by the certificate ID.
Status	ServiceAliasConfigStatus	Indicates the status of configuration that needs to be persisted.
PreferPort	string	Indicates the port the user wants to expose. If empty, a port will be selected for the service.
InsecureEdgeTerminationPolicy	routeapi.InsecureEdgeTerminationPolicyType	Indicates desired behavior for insecure connections to an edge-terminated route: none (or disable), allow , or redirect .
RoutingKeyName	string	Hash of the route + namespace name used to obscure the cookie ID.
IsWildcard	bool	Indicates this service unit needing wildcard support.
Annotations	map[string]string	Annotations attached to this route.

Field	Type	Description
ServiceUnitNames	<code>map[string]int32</code>	Collection of services that support this route, keyed by service name and valued on the weight attached to it with respect to other entries in the map.
ActiveServiceUnits	<code>int</code>	Count of the ServiceUnitNames with a non-zero weight.

The **ServiceAliasConfig** is a route for a service. Uniquely identified by host + path. The default template iterates over routes using `{{range $cfgIdx, $cfg := .State }}`. Within such a `{{range}}` block, the template can refer to any field of the current **ServiceAliasConfig** using `$cfg.Field`.

Table 3.5. Router ServiceUnit

Field	Type	Description
Name	<code>string</code>	Name corresponds to a service name + namespace. Uniquely identifies the ServiceUnit .
EndpointTable	<code>[]Endpoint</code>	Endpoints that back the service. This translates into a final back-end implementation for routers.

ServiceUnit is an encapsulation of a service, the endpoints that back that service, and the routes that point to the service. This is the data that drives the creation of the router configuration files

Table 3.6. Router Endpoint

Field	Type
ID	<code>string</code>
IP	<code>string</code>
Port	<code>string</code>
TargetName	<code>string</code>
PortName	<code>string</code>
IdHash	<code>string</code>

Field	Type
NoHealthCheck	bool

Endpoint is an internal representation of a Kubernetes endpoint.

Table 3.7. Router Certificate, ServiceAliasConfigStatus

Field	Type	Description
Certificate	string	Represents a public/private key pair. It is identified by an ID, which will become the file name. A CA certificate will not have a PrivateKey set.
ServiceAliasConfigStatus	string	Indicates that the necessary files for this configuration have been persisted to disk. Valid values: "saved", "".

Table 3.8. Router Certificate Type

Field	Type	Description
ID	string	
Contents	string	The certificate.
PrivateKey	string	The private key.

Table 3.9. Router TLSTerminationType

Field	Type	Description
TLSTerminationType	string	Dictates where the secure communication will stop.
InsecureEdgeTerminationPolicyType	string	Indicates the desired behavior for insecure connections to a route. While each router may make its own decisions on which ports to expose, this is normally port 80.

TLSTerminationType and **InsecureEdgeTerminationPolicyType** dictate where the secure communication will stop.

Table 3.10. Router TLSTerminationType Values

Constant	Value	Meaning
TLSTerminationEdge	edge	Terminate encryption at the edge router.
TLSTerminationPassthrough	passthrough	Terminate encryption at the destination, the destination is responsible for decrypting traffic.
TLSTerminationReencrypt	reencrypt	Terminate encryption at the edge router and re-encrypt it with a new certificate supplied by the destination.

Table 3.11. Router InsecureEdgeTerminationPolicyType Values

Type	Meaning
Allow	Traffic is sent to the server on the insecure port (default).
Disable	No traffic is allowed on the insecure port.
Redirect	Clients are redirected to the secure port.

None ("") is the same as **Disable**.

3.3.3.4. Annotations

Each route can have annotations attached. Each annotation is just a name and a value.

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms
[...]
```

The name can be anything that does not conflict with existing Annotations. The value is any string. The string can have multiple tokens separated by a space. For example, **aa bb cc**. The template uses **{{index}}** to extract the value of an annotation. For example:

```
{{ $balanceAlgo := index $cfg.Annotations
"haproxy.router.openshift.io/balance" }}
```

This is an example of how this could be used for mutual client authorization.

```
{{ with $cnList := index $cfg.Annotations "whiteListCertCommonName" }}
{{ if ne $cnList "" }}
  acl test ssl_c_s_dn(CN) -m str {{ $cnList }}
```

```
    http-request deny if !test
  {{ end }}
{{ end }}
```

Then, you can handle the white-listed CNs with this command.

```
$ oc annotate route <route-name> --overwrite whiteListCertCommonName="CN1
CN2 CN3"
```

See [Route-specific Annotations](#) for more information.

3.3.3.5. Environment Variables

The template can use any environment variables that exist in the router pod. The environment variables can be set in the deployment configuration. New environment variables can be added.

They are referenced by the **env** function:

```
{{env "ROUTER_MAX_CONNECTIONS" "20000"}}
```

The first string is the variable, and the second string is the default when the variable is missing or **nil**. When **ROUTER_MAX_CONNECTIONS** is not set or is **nil**, 20000 is used. Environment variables are a map where the key is the environment variable name and the content is the value of the variable.

See [Route-specific Environment variables](#) for more information.

3.3.3.6. Example Usage

Here is a simple template based on the HAProxy template file.

Start with a comment:

```
{{/*
  Here is a small example of how to work with templates
  taken from the HAProxy template file.
*/}}
```

The template can create any number of output files. Use a define construct to create an output file. The file name is specified as an argument to define, and everything inside the define block up to the matching end is written as the contents of that file.

```
{{ define "/var/lib/haproxy/conf/haproxy.config" }}
global
{{ end }}
```

The above will copy **global** to the **/var/lib/haproxy/conf/haproxy.config** file, and then close the file.

Set up logging based on environment variables.

```
{{ with (env "ROUTER_SYSLOG_ADDRESS" "") }}
  log {{.}} {{env "ROUTER_LOG_FACILITY" "local1"}} {{env
"ROUTER_LOG_LEVEL" "warning"}}
{{ end }}
```

The **env** function extracts the value for the environment variable. If the environment variable is not defined or **nil**, the second argument is returned.

The **with** construct sets the value of "." (dot) within the **with** block to whatever value is provided as an argument to **with**. The **with** action tests **Dot** for **nil**. If not **nil**, the clause is processed up to the **end**. In the above, assume **ROUTER_SYSLOG_ADDRESS** contains **/var/log/msg**, **ROUTER_LOG_FACILITY** is not defined, and **ROUTER_LOG_LEVEL** contains **info**. The following will be copied to the output file:

```
log /var/log/msg local1 info
```

Each admitted route ends up generating lines in the configuration file. Use **range** to go through the admitted routes:

```
{{ range $cfgIdx, $cfg := .State }}
    backend be_http_{{ $cfgIdx }}
{{ end }}
```

.State is a map of **ServiceAliasConfig**, where the key is the route name. **range** steps through the map and, for each pass, it sets **\$cfgIdx** with the **key**, and sets **`\$cfg** to point to the **ServiceAliasConfig** that describes the route. If there are two routes named **myroute** and **hisroute**, the above will copy the following to the output file:

```
backend be_http_myroute
backend be_http_hisroute
```

Route Annotations, **\$cfg.Annotations**, is also a map with the annotation name as the key and the content string as the value. The route can have as many annotations as desired and the use is defined by the template author. The user codes the annotation into the route and the template author customized the HAProxy template to handle the annotation.

The common usage is to index the annotation to get the value.

```
{{ $balanceAlgo := index $cfg.Annotations
    "haproxy.router.openshift.io/balance" }}
```

The **index** extracts the value for the given annotation, if any. Therefore, **`\$balanceAlgo** will contain the string associated with the annotation or **nil**. As above, you can test for a non-**nil** string and act on it with the **with** construct.

```
{{ with $balanceAlgo }}
    balance $balanceAlgo
{{ end }}
```

Here when **\$balanceAlgo** is not **nil**, **balance \$balanceAlgo** is copied to the output file.

In a second example, you want to set a server timeout based on a timeout value set in an annotation.

```
$value := index $cfg.Annotations "haproxy.router.openshift.io/timeout"
```

The **\$value** can now be evaluated to make sure it contains a properly constructed string. The **matchPattern** function accepts a regular expression and returns **true** if the argument satisfies the expression.

```
matchPattern "[1-9][0-9]*(us|ms|s|m|h|d)?" $value
```

This would accept **5000ms** but not **7y**. The results can be used in a test.

```
{{if (matchPattern "[1-9][0-9]*(us|ms|s|m|h|d)?" $value) }}
  timeout server {{$value}}
{{ end }}
```

It can also be used to match tokens:

```
matchPattern "roundrobin|leastconn|source" $balanceAlgo
```

Alternatively **matchValues** can be used to match tokens:

```
matchValues $balanceAlgo "roundrobin" "leastconn" "source"
```

3.3.4. Using a ConfigMap to Replace the Router Configuration Template

You can use a [ConfigMap](#) to customize the router instance without rebuilding the router image. The **haproxy-config.template**, **reload-haproxy**, and other scripts can be modified as well as creating and modifying router environment variables.

1. Copy the **haproxy-config.template** that you want to modify as [described above](#). Modify it as desired.
2. Create a ConfigMap:

```
$ oc create configmap customrouter --from-file=haproxy-
config.template
```

The **customrouter** ConfigMap now contains a copy of the modified **haproxy-config.template** file.

3. Modify the router deployment configuration to mount the ConfigMap as a file and point the **TEMPLATE_FILE** environment variable to it. This can be done via **oc set env** and **oc volume** commands, or alternatively by editing the router deployment configuration.

Using oc commands

```
$ oc volume dc/router --add --overwrite \
  --name=config-volume \
  --mount-path=/var/lib/haproxy/conf/custom \
  --source='{"configMap": { "name": "customrouter"}}'
$ oc set env dc/router \
  TEMPLATE_FILE=/var/lib/haproxy/conf/custom/haproxy-
config.template
```

Editing the Router Deployment Configuration

Use **oc edit dc router** to edit the router deployment configuration with a text editor.

```
...
- name: STATS_USERNAME
  value: admin
```

```

        - name: TEMPLATE_FILE 1
          value: /var/lib/haproxy/conf/custom/haproxy-
config.template
        image: openshift/origin-haproxy-routerp
    ...
    terminationMessagePath: /dev/termination-log
    volumeMounts: 2
    - mountPath: /var/lib/haproxy/conf/custom
      name: config-volume
    dnsPolicy: ClusterFirst
    ...
    terminationGracePeriodSeconds: 30
    volumes: 3
    - configMap:
      name: customrouter
      name: config-volume
    ...

```

- 1** In the `spec.container.env` field, add the `TEMPLATE_FILE` environment variable to point to the mounted *haproxy-config.template* file.
- 2** Add the `spec.container.volumeMounts` field to create the mount point.
- 3** Add a new `spec.volumes` field to mention the ConfigMap.

Save the changes and exit the editor. This restarts the router.

3.3.5. Using Stick Tables

The following example customization can be used in a [highly-available routing setup](#) to use stick-tables that synchronize between peers.

Adding a Peer Section

In order to synchronize stick-tables amongst peers you must define a peers section in your HAProxy configuration. This section determines how HAProxy will identify and connect to peers. The plug-in provides data to the template under the `.PeerEndpoints` variable to allow you to easily identify members of the router service. You may add a peer section to the *haproxy-config.template* file inside the router image by adding:

```

{{ if (len .PeerEndpoints) gt 0 }}
peers openshift_peers
  {{ range $endpointID, $endpoint := .PeerEndpoints }}
    peer {{$endpoint.TargetName}} {{$endpoint.IP}}:1937
  {{ end }}
{{ end }}

```

Changing the Reload Script

When using stick-tables, you have the option of telling HAProxy what it should consider the name of the local host in the peer section. When creating endpoints, the plug-in attempts to set the `TargetName` to the value of the endpoint's `TargetRef.Name`. If `TargetRef` is not set, it will set the `TargetName` to the

IP address. The **TargetRef.Name** corresponds with the Kubernetes host name, therefore you can add the **-L** option to the **reload-haproxy** script to identify the local host in the peer section.

```
peer_name=$HOSTNAME 1

if [ -n "$old_pid" ]; then
    /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name -sf
    $old_pid
else
    /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name
fi
```

1 Must match an endpoint target name that is used in the peer section.

Modifying Back Ends

Finally, to use the stick-tables within back ends, you can modify the HAProxy configuration to use the stick-tables and peer set. The following is an example of changing the existing back end for TCP connections to use stick-tables:

```
        {{ if eq $cfg.TLS Termination "passthrough" }}
backend be_tcp_{{ $cfgIdx }}
    balance leastconn
    timeout check 5000ms
    stick-table type ip size 1m expire 5m{{ if (len $.PeerEndpoints) gt 0 }}
peers openshift_peers {{ end }}
    stick on src
        {{ range $endpointID, $endpoint :=
$serviceUnit.EndpointTable }}
    server {{ $endpointID }} {{ $endpoint.IP }}:{{ $endpoint.Port }} check inter
5000ms
        {{ end }}
    {{ end }}
```

After this modification, you can [rebuild your router](#).

3.3.6. Rebuilding Your Router

In order to rebuild the router, you need copies of several files that are present on a running router. Make a work directory and copy the files from the router:

```
# mkdir - myrouter/conf
# cd myrouter
# oc get po
NAME                                READY    STATUS    RESTARTS   AGE
router-2-40fc3                      1/1      Running   0           11d
# oc rsh router-2-40fc3 cat haproxy-config.template > conf/haproxy-
config.template
# oc rsh router-2-40fc3 cat error-page-503.http > conf/error-page-503.http
# oc rsh router-2-40fc3 cat default_pub_keys.pem >
conf/default_pub_keys.pem
# oc rsh router-2-40fc3 cat ../Dockerfile > Dockerfile
# oc rsh router-2-40fc3 cat ../reload-haproxy > reload-haproxy
```


You can edit or replace any of these files. However, ***conf/haproxy-config.template*** and ***reload-haproxy*** are the most likely to be modified.

After updating the files:

```
# docker build -t openshift/origin-haproxy-router-myversion .
# docker tag openshift/origin-haproxy-router-myversion
172.30.243.98:5000/openshift/haproxy-router-myversion 1
# docker push 172.30.243.98:5000/openshift/origin-haproxy-router-pc:latest
2
```

1 Tag the version with the repository. In this case the repository is **172.30.243.98:5000**.

2 Push the tagged version to the repository. It may be necessary to **docker login** to the repository first.

To use the new router, edit the router deployment configuration either by changing the **image:** string or by adding the **--images=<repo>/<image>:<tag>** flag to the **oc adm router** command.

When debugging the changes, it is helpful to set **imagePullPolicy: Always** in the deployment configuration to force an image pull on each pod creation. When debugging is complete, you can change it back to **imagePullPolicy: IfNotPresent** to avoid the pull on each pod start.

3.4. CONFIGURING THE HAPROXY ROUTER TO USE THE PROXY PROTOCOL

3.4.1. Overview

By default, the HAProxy router expects incoming connections to unsecure, edge, and re-encrypt routes to use HTTP. However, you can configure the router to expect incoming requests by using [the PROXY protocol](#) instead. This topic describes how to configure the HAProxy router and an external load balancer to use the PROXY protocol.

3.4.2. Why Use the PROXY Protocol?

When an intermediary service such as a proxy server or load balancer forwards an HTTP request, it appends the source address of the connection to the request's "Forwarded" header in order to provide this information to subsequent intermediaries and to the back-end service to which the request is ultimately forwarded. However, if the connection is encrypted, intermediaries cannot modify the "Forwarded" header. In this case, the HTTP header will not accurately communicate the original source address when the request is forwarded.

To solve this problem, some load balancers encapsulate HTTP requests using the PROXY protocol as an alternative to simply forwarding HTTP. Encapsulation enables the load balancer to add information to the request without modifying the forwarded request itself. In particular, this means that the load balancer can communicate the source address even when forwarding an encrypted connection.

The HAProxy router can be configured to accept the PROXY protocol and decapsulate the HTTP request. Because the router terminates encryption for edge and re-encrypt routes, the router can then update the "Forwarded" HTTP header (and related HTTP headers) in the request, appending any source address that is communicated using the PROXY protocol.

**WARNING**

The PROXY protocol and HTTP are incompatible and cannot be mixed. If you use a load balancer in front of the router, both must use either the PROXY protocol or HTTP. Configuring one to use one protocol and the other to use the other protocol will cause routing to fail.

3.4.3. Using the PROXY Protocol

By default, the HAProxy router does not use the PROXY protocol. The router can be configured using the **ROUTER_USE_PROXY_PROTOCOL** environment variable to expect the PROXY protocol for incoming connections:

Enable the PROXY Protocol

```
$ oc env dc/router ROUTER_USE_PROXY_PROTOCOL=true
```

Set the variable to any value other than **true** or **TRUE** to disable the PROXY protocol:

Disable the PROXY Protocol

```
$ oc env dc/router ROUTER_USE_PROXY_PROTOCOL=false
```

If you enable the PROXY protocol in the router, you must configure your load balancer in front of the router to use the PROXY protocol as well. Following is an example of configuring Amazon's Elastic Load Balancer (ELB) service to use the PROXY protocol. This example assumes that ELB is forwarding ports 80 (HTTP), 443 (HTTPS), and 5000 (for the image registry) to the router running on one or more EC2 instances.

Configure Amazon ELB to Use the PROXY Protocol

1. To simplify subsequent steps, first set some shell variables:

```
$ lb='infra-lb' 1
$ instances=( 'i-079b4096c654f563c' ) 2
$ secgroups=( 'sg-e1760186' ) 3
$ subnets=( 'subnet-cf57c596' ) 4
```

- 1** The name of your ELB.
- 2** The instance or instances on which the router is running.
- 3** The security group or groups for this ELB.
- 4** The subnet or subnets for this ELB.

2. Next, create the ELB with the appropriate listeners, security groups, and subnets.

**NOTE**

You must configure all listeners to use the TCP protocol, not the HTTP protocol.

```
$ aws elb create-load-balancer --load-balancer-name "$lb" \
  --listeners \

'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=
80' \

'Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort
=443' \

'Protocol=TCP,LoadBalancerPort=5000,InstanceProtocol=TCP,InstancePor
t=5000' \
  --security-groups $secgroups \
  --subnets $subnets
{
  "DNSName": "infra-lb-2006263232.us-east-1.elb.amazonaws.com"
}
```

3. Register your router instance or instances with the ELB:

```
$ aws elb register-instances-with-load-balancer --load-balancer-name
"$lb" \
  --instances $instances
{
  "Instances": [
    {
      "InstanceId": "i-079b4096c654f563c"
    }
  ]
}
```

4. Configure the ELB's health check:

```
$ aws elb configure-health-check --load-balancer-name "$lb" \
  --health-check
'Target=HTTP:1936/healthz,Interval=30,UnhealthyThreshold=2,HealthyTh
reshold=2,Timeout=5'
{
  "HealthCheck": {
    "HealthyThreshold": 2,
    "Interval": 30,
    "Target": "HTTP:1936/healthz",
    "Timeout": 5,
    "UnhealthyThreshold": 2
  }
}
```

5. Finally, create a load-balancer policy with the **ProxyProtocol** attribute enabled, and configure it on the ELB's TCP ports 80 and 443:

```
$ aws elb create-load-balancer-policy --load-balancer-name "$lb" \
```

```

--policy-name "${lb}-ProxyProtocol-policy" \
--policy-type-name 'ProxyProtocolPolicyType' \
--policy-attributes
'AttributeName=ProxyProtocol,AttributeValue=true'
$ for port in 80 443
do
    aws elb set-load-balancer-policies-for-backend-server \
        --load-balancer-name "$lb" \
        --instance-port "$port" \
        --policy-names "${lb}-ProxyProtocol-policy"
done

```

Verify the Configuration

You can examine the load balancer as follows to verify that the configuration is correct:

```

$ aws elb describe-load-balancers --load-balancer-name "$lb" |
jq '.LoadBalancerDescriptions| [.]|.ListenerDescriptions|'
[
  [
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] ❶
    },
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] ❷
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": [] ❸
    }
  ]
]

```

- ❶ The listener for TCP port 80 should have the policy for using the PROXY protocol.
- ❷ The listener for TCP port 443 should have the same policy.
- ❸ The listener for TCP port 5000 should **not** have the policy.

Alternatively, if you already have an ELB configured, but it is not configured to use the PROXY protocol, you will need to change the existing listener for TCP port 80 to use the TCP protocol instead of HTTP (TCP port 443 should already be using the TCP protocol):

```
$ aws elb delete-load-balancer-listeners --load-balancer-name "$lb" \
    --load-balancer-ports 80
$ aws elb create-load-balancer-listeners --load-balancer-name "$lb" \
    --listeners
'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80'
```

Verify the Protocol Updates

Verify that the protocol has been updated as follows:

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
jq '[.LoadBalancerDescriptions[]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP", ❶
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    }
  ]
]
```

❶ All listeners, including the listener for TCP port 80, should be using the TCP protocol.

Then, create a load-balancer policy and add it to the ELB as described in Step 5 above.

3.5. USING THE F5 ROUTER PLUG-IN

3.5.1. Overview



NOTE

The F5 router plug-in is available starting in OpenShift Container Platform 3.0.2.



WARNING

The F5 router plug-in will be deprecated in OpenShift Container Platform version 3.11. The functionality of the F5 router plug-in is replaced in the F5 BIG-IP® Controller for OpenShift. For more information, see [F5 BIG-IP Controller for OpenShift](#). For information about migrating existing deployments from the F5 router plug-in to the BIG-IP Controller for OpenShift, see [Replace the F5 Router with the F5 BIG-IP Controller for OpenShift](#).

The F5 router plug-in is provided as a container image and run as a pod, just like the [default HAProxy router](#).



IMPORTANT

Support relationships between F5 and Red Hat provide a full scope of support for both models of F5 integration, F5 router plug-in and the F5 BIG-IP Controller for OpenShift. If you are currently using the F5 router plug-in, Red Hat support will provide the initial support and work with F5 support if necessary. If you are currently using the F5 BIG-IP Controller for OpenShift, F5 will provide the initial support and work with Red Hat if necessary.

3.5.2. Prerequisites and Supportability

When deploying the F5 router plug-in, ensure you meet the following requirements:

- A F5 host IP with:
 - Credentials for API access
 - SSH access via a private key
- An F5 user with Advanced Shell access
- A virtual server for HTTP routes:
 - *HTTP profile* must be *http*.
- A virtual server with HTTP profile routes:
 - *HTTP profile* must be *http*
 - *SSL Profile (client)* must be *clientssl*
 - *SSL Profile (server)* must be *serverssl*

- For edge integration (not recommended):
 - A working ramp node
 - A working tunnel to the ramp node
- For native integration:
 - A host-internal IP capable of communicating with all nodes on the port 4789/UDP
 - The sdn-services add-on license installed on the F5 host.

The F5 router plug-in for OpenShift Container Platform supports only the following **F5 BIG-IP** versions:

- 11.x
- 12.x

The F5 BIG-IP Controller for OpenShift supports the OpenShift Container Platform versions found in the [F5 BIG-IP Controller for OpenShift releases and versioning](#) page in the F5 documentation.

IMPORTANT

The following features are not supported with **F5 BIG-IP** using the F5 router plug-in:

- Wildcard routes together with re-encrypt routes - you must supply a certificate and a key in the route. If you provide a certificate, a key, and a certificate authority (CA), the CA is never used.
- A pool is created for all services, even for the ones with no associated route.
- [Idling applications](#)
- Unencrypted HTTP traffic in *redirect* mode, with edge TLS termination. (**insecureEdgeTerminationPolicy: Redirect**)
- Sharding, that is, having multiple **vservers** on the F5.
- SSL cipher (**ROUTER_CIPHERS=modern/old**)
- Customizing the endpoint health checks for time-intervals and the type of checks.
- Serving F5 metrics by using a metrics server.
- Specifying a different target port (**PreferPort/TargetPort**) rather than the ones specified in the service.
- Customizing the source IP whitelists, that is, allowing traffic for a route only from specific IP addresses.
- Customizing timeout values, such as **max connect time**, or **tcp FIN timeout**.
- HA mode for the **F5 BIG-IP**.

3.5.2.1. Configuring the Virtual Servers

As a prerequisite to working with the F5 router plug-in, two virtual servers (one virtual server each for HTTP and HTTPS profiles, respectively) need to be set up in the **F5 BIG-IP** appliance.

To set up a virtual server in the **F5 BIG-IP** appliance, follow the [instructions from F5](#).

While creating the virtual server, ensure the following settings are in place:

- For the HTTP server, set the **ServicePort** to **'http'/80**.
- For the HTTPS server, set the **ServicePort** to **'https'/443**.
- In the basic configuration, set the HTTP profile to **/Common/http** for both of the virtual servers.
- For the HTTPS server, create a default **client-ssl** profile and select it for the **SSL Profile (Client)**.
 - To create the default client SSL profile, follow the [instructions from F5](#), especially the **Configuring the fallback (default) client SSL profile** section, which discusses that the certificate/key pair is the default that will be served in the case that custom certificates are not provided for a route or server name.

3.5.3. Deploying the F5 Router Plug-in



IMPORTANT

The F5 router must be run in privileged mode, because route certificates are copied using the **scp** command:

```
$ oc adm policy remove-scc-from-user hostnetwork -z router
$ oc adm policy add-scc-to-user privileged -z router
```

Deploy the F5 router plug-in with the **oc adm router** command, but provide additional flags (or environment variables) specifying the following parameters for the **F5 BIG-IP** host:

Flag	Description
--type=f5-router	Specifies to launch an F5 router plug-in instead of the default haproxy-router. (the default --type is haproxy-router).
--external-host	Specifies the F5 BIG-IP host's management interface's host name or IP address.
--external-host-username	Specifies the F5 BIG-IP user name (typically admin). The F5 BIG-IP user account must have access to the Advanced Shell (Bash) on the F5 BIG-IP system.
--external-host-password	Specifies the F5 BIG-IP password.

Flag	Description
--external-host-http-vserver	Specifies the name of the F5 virtual server for HTTP connections. This must be configured by the user prior to launching the router pod.
--external-host-https-vserver	Specifies the name of the F5 virtual server for HTTPS connections. This must be configured by the user prior to launching the router pod.
--external-host-private-key	Specifies the path to the SSH private key file for the F5 BIG-IP host. Required to upload and delete key and certificate files for routes.
--external-host-insecure	A Boolean flag that indicates that the F5 router plug-in does not use strict certificate verification with the F5 BIG-IP host.
--external-host-partition-path	Specifies the F5 BIG-IP® partition path (the default is /Common).

For example:

```
$ oc adm router \
  --type=f5-router \
  --external-host=10.0.0.2 \
  --external-host-username=admin \
  --external-host-password=myspassword \
  --external-host-http-vserver=ose-vserver \
  --external-host-https-vserver=https-ose-vserver \
  --external-host-private-key=/path/to/key \
  --host-network=false \
  --service-account=router
```

As with the HAProxy router, the **oc adm router** command creates the service and deployment configuration objects, and thus the replication controllers and pod(s) in which the F5 router plug-in itself runs. The replication controller restarts the F5 router plug-in in case of crashes. Because the F5 router plug-in is watching routes, endpoints, and nodes and configuring **F5 BIG-IP** accordingly, running the F5 router in this way, along with an appropriately configured **F5 BIG-IP** deployment, satisfies high-availability requirements.

3.5.4. F5 Router Plug-in Partition Paths

Partition paths allow you to store your OpenShift Container Platform routing configuration in a custom **F5 BIG-IP** administrative partition, instead of the default **/Common** partition. You can use custom administrative partitions to secure **F5 BIG-IP** environments. This means that an OpenShift Container Platform-specific configuration stored in **F5 BIG-IP** system objects reside within a logical container, allowing administrators to define access control policies on that specific administrative partition.

See the [F5 BIG-IP documentation](#) for more information about administrative partitions.

To configure your OpenShift Container Platform for partition paths:

1. Optionally, perform some cleaning steps:
 - a. Ensure F5 is configured to be able to switch to the **/Common** and **/Custom** paths.
 - b. Delete the static FDB of **vxlan5000**. See the [F5 BIG-IP® documentation](#) for more information.
2. [Configure a virtual server](#) for the custom partition.
3. To specify a partition path, deploy the F5 router plug-in using the **--external-host-partition-path** flag:

```
$ oc adm router --external-host-partition-path=/OpenShift/zone1 ...
```

3.5.5. Setting Up F5 Router Plug-in



NOTE

This section reviews how to set up F5 native integration with OpenShift Container Platform. The concepts of the F5 appliance and OpenShift Container Platform connection and data flow of the F5 router plug-in are discussed in the [F5 Router Plug-in](#) section of the Routes topic.



NOTE

Only **F5 BIG-IP** appliance versions 11.x and 12.x work with the F5 router plug-in presented in this section. You also need sdn-services add-on license for the integration to work properly. For version 11.x, follow the instructions to set up a [ramp node](#).

With F5 router plug-in for OpenShift Container Platform, you do not need to configure a ramp node for F5 to be able to reach the pods on the overlay network as created by OpenShift SDN.

The F5 router plug-in pod needs to be launched with enough information so that it can successfully directly connect to pods.

1. Create a ghost **hostsubnet** on the OpenShift Container Platform cluster:

```
$ cat > f5-hostsubnet.yaml << EOF
{
  "kind": "HostSubnet",
  "apiVersion": "v1",
  "metadata": {
    "name": "openshift-f5-node",
    "annotations": {
      "pod.network.openshift.io/assign-subnet": "true",
      "pod.network.openshift.io/fixed-vnid-host": "0" 1
    }
  },
  "host": "openshift-f5-node",
  "hostIP": "10.3.89.213" 2
} EOF
$ oc create -f f5-hostsubnet.yaml
```

- 1 Make F5 global.
- 2 The internal IP of the F5 appliance.

2. Determine the subnet allocated for the ghost **hostsubnet** just created:

```
$ oc get hostsubnets
NAME                                HOST                                HOST IP
SUBNET
openshift-f5-node                   openshift-f5-node                   10.3.89.213
10.131.0.0/23
openshift-master-node               openshift-master-node               172.17.0.2
10.129.0.0/23
openshift-node-1                    openshift-node-1                    172.17.0.3
10.128.0.0/23
openshift-node-2                    openshift-node-2                    172.17.0.4
10.130.0.0/23
```

3. Check the **SUBNET** for the newly created **hostsubnet**. In this example, **10.131.0.0/23**.

4. Get the entire pod network's CIDR:

```
$ oc get clusternetwork
```

This value will be something like **10.128.0.0/14**, noting the mask (**14** in this example).

5. To construct the gateway address, pick any IP address from the **hostsubnet** (for example, **10.131.0.5**). Use the mask of the pod network (**14**). The gateway address becomes: **10.131.0.5/14**.

6. Launch the F5 router plug-in pod, following [these instructions](#). Additionally, allow the access to 'node' cluster resource for the service account and use the two new additional options for VXLAN native integration.

```
$ # Add policy to allow router to access nodes using the sdn-reader
role
$ oc adm policy add-cluster-role-to-user system:sdn-reader
system:serviceaccount:default:router
$ # Launch the F5 router plug-in pod with vxlan-gw and F5's internal
IP as extra arguments
$ #--external-host-internal-ip=10.3.89.213
$ #--external-host-vxlan-gw=10.131.0.5/14
$ oc adm router \
  --type=f5-router \
  --external-host=10.3.89.90 \
  --external-host-username=admin \
  --external-host-password=mypassword \
  --external-host-http-vserver=ose-vserver \
  --external-host-https-vserver=https-ose-vserver \
  --external-host-private-key=/path/to/key \
  --service-account=router \
  --host-network=false \
  --external-host-internal-ip=10.3.89.213 \
  --external-host-vxlan-gw=10.131.0.5/14
```

**NOTE**

The **external1-host-username** is a **F5 BIG-IP** user account with access to the Advanced Shell (Bash) on the F5 BIG-IP system.

CHAPTER 4. DEPLOYING RED HAT CLOUDFORMS

4.1. DEPLOYING RED HAT CLOUDFORMS ON OPENSIFT CONTAINER PLATFORM

4.1.1. Introduction

The OpenShift Container Platform installer includes the Ansible role **openshift-management** and playbooks for deploying Red Hat CloudForms 4.6 (CloudForms Management Engine 5.9, or CFME) on OpenShift Container Platform.



WARNING

The current implementation is incompatible with the Technology Preview deployment process of Red Hat CloudForms 4.5 as described in [OpenShift Container Platform 3.6 documentation](#).

When deploying Red Hat CloudForms on OpenShift Container Platform, there are two major decisions to make:

1. Do you want an external or a containerized (also referred to as *podified*) PostgreSQL database?
2. Which storage class will back your persistent volumes (PVs)?

For the first decision, you can deploy Red Hat CloudForms in one of two ways, depending on the location of the PostgreSQL database to be used by Red Hat CloudForms:

Deployment Variant	Description
Fully containerized	All application services and the PostgreSQL database are run as pods on OpenShift Container Platform.
External database	The application utilizes an externally-hosted PostgreSQL database server, while all other services are ran as pods on OpenShift Container Platform.

For the second decision, the **openshift-management** role provides customization options for overriding many default deployment parameters. This includes the following storage class options to back your PVs:

Storage Class	Description
NFS (default)	Local, on cluster
NFS External	NFS somewhere else, like a storage appliance

Storage Class	Description
Cloud Provider	Use automatic storage provisioning from your cloud provider (Google Cloud Engine, Amazon Web Services, or Microsoft Azure)
Preconfigured (advanced)	Assumes you created everything ahead of time

Topics in this guide include the requirements for running Red Hat CloudForms on OpenShift Container Platform, descriptions of the available configuration variables, and instructions on running the installer either during your initial OpenShift Container Platform installation or after your cluster has been provisioned.

4.2. REQUIREMENTS FOR RED HAT CLOUDFORMS ON OPENSIFT CONTAINER PLATFORM

The default requirements are listed in the table below. These can be overridden by [customizing template parameters](#).



IMPORTANT

The application performance will suffer, or possibly even fail to deploy, if these requirements are not satisfied.

Table 4.1. Default Requirements

Item	Requirement	Description	Customization Parameter
Application Memory	≥ 4.0 Gi	Minimum required memory for the application	APPLICATION_MEM_REQ
Application Storage	≥ 5.0 Gi	Minimum PV size required for the application	APPLICATION_VOLUME_CAPACITY
PostgreSQL Memory	≥ 6.0 Gi	Minimum required memory for the database	POSTGRESQL_MEM_REQ
PostgreSQL Storage	≥ 15.0 Gi	Minimum PV size required for the database	DATABASE_VOLUME_CAPACITY
Cluster Hosts	≥ 3	Number of hosts in your cluster	N/A

To sum up these requirements:

- You must have several cluster nodes.
- Your cluster nodes must have lots of memory available.
- You must have several GiB's of storage available, either locally or on your cloud provider.
- PV sizes can be changed by providing override values to template parameters.

4.3. CONFIGURING ROLE VARIABLES

4.3.1. Overview

The following sections describe role variables that may be used in your [Ansible inventory file](#), which is used to control the behavior of the Red Hat CloudForms installation when [running the installer](#).

4.3.2. General Variables

Variable	Required	Default	Description
<code>openshift_management_install_management</code>	No	false	Boolean, set to true to install the application.
<code>openshift_management_app_template</code>	Yes	cfme-template	The deployment variant of Red Hat CloudForms to install. Set cfme-template for a containerized database or cfme-template-ext-db for an external database.
<code>openshift_management_project</code>	No	openshift-t-management	Namespace (project) for the Red Hat CloudForms installation.
<code>openshift_management_project_description</code>	No	CloudForms Management Engine	Namespace (project) description.
<code>openshift_management_username</code>	No	admin	Default management user name. Changing this value does not change the user name; only change this value if you have changed the name already and are running integration scripts (such as the script to add container providers).

Variable	Required	Default	Description
openshift_management_password	No	smartvm	Default management password. Changing this value does not change the password; only change this value if you have changed the password already and are running integration scripts (such as the script to add container providers).

4.3.3. Customizing Template Parameters

You can use the **openshift_management_template_parameters** Ansible role variable to specify any template parameters you want to override in the application or PV templates.

For example, if you wanted to reduce the memory requirement of the PostgreSQL pod, then you could set the following:

```
openshift_management_template_parameters={'POSTGRESQL_MEM_REQ': '1Gi'}
```

When the Red Hat CloudForms template is processed, **1Gi** will be used for the value of the **POSTGRESQL_MEM_REQ** template parameter.

Not all template parameters are present in *both* template variants (containerized or external database). For example, while the podified database template has a **POSTGRESQL_MEM_REQ** parameter, no such parameter is present in the external db template, as there is no need for this information due to there being no databases that require pods.

Therefore, be very careful if you are overriding template parameters. Including parameters not defined in a template will cause errors. If you do receive an error during the **Ensure the Management App is created** task, run the [uninstall scripts](#) first before running the installer again.

4.3.4. Database Variables

4.3.4.1. Containerized (Podified) Database

Any **POSTGRES_*** or **DATABASE_*** template parameters in the *cfme-template.yaml* file may be customized through the **openshift_management_template_parameters** hash in your inventory file..

4.3.4.2. External Database

Any **POSTGRES_*** or **DATABASE_*** template parameters in the *cfme-template-ext-db.yaml* file may be customized through the **openshift_management_template_parameters** hash in your inventory file..

External PostgreSQL databases require you to provide database connection parameters. You must set the required connection keys in the **openshift_management_template_parameters** parameter in your inventory. The following keys are required:

- **DATABASE_USER**
- **DATABASE_PASSWORD**
- **DATABASE_IP**

- **DATABASE_PORT** (Most PostgreSQL servers run on port **5432**)
- **DATABASE_NAME**



NOTE

Ensure your external database is running PostgreSQL 9.5 or you may not be able to deploy the CloudForms application successfully.

Your inventory would contain a line similar to:

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db 1
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10',
'DATABASE_PORT': '5432', 'DATABASE_NAME': 'cfme'}
```

- 1** Set **openshift_management_app_template** parameter to **cfme-template-ext-db**.

4.3.5. Storage Class Variables

Variable	Required	Default	Description
openshift_management_storage_class	No	nfs	Storage type to use. Options are nfs , nfs_external , preconfigured , or cloudprovider .
openshift_management_storage_nfs_external_hostname	No	false	If you are using an external NFS server, such as a NetApp appliance, then you must set the host name here. Leave the value as false if you are not using external NFS. Additionally, external NFS requires that you create the NFS exports that will back the application PV and optionally the database PV.

Variable	Required	Default	Description
openshift_management_storage_nfs_base_dir	No	/exports/	If you are using external NFS, then you can set the base path to the exports location here. For local NFS, you can also change this value if you want to change the default path used for local NFS exports.
openshift_management_storage_nfs_local_hostname	No	false	If you do not have an [nfs] group in your inventory, or want to simply manually define the local NFS host in your cluster, set this parameter to the host name of the preferred NFS server. The server must be a part of your OpenShift Container Platform cluster.

4.3.5.1. NFS (Default)

The NFS storage class is best suited for proof-of-concept and test deployments. It is also the default storage class for deployments. No additional configuration is required for this choice.

This storage class configures NFS on a cluster host (by default, the first master in the inventory file) to back the required PVs. The application requires a PV, and the database (which may be hosted externally) may require a second. PV minimum required sizes are 5GiB for the Red Hat CloudForms application, and 15GiB for the PostgreSQL database (20GiB minimum available space on a volume or partition if used specifically for NFS purposes).

Customization is provided through the following role variables:

- **openshift_management_storage_nfs_base_dir**
- **openshift_management_storage_nfs_local_hostname**

4.3.5.2. NFS External

External NFS leans on pre-configured NFS servers to provide exports for the required PVs. For external NFS you must have a **cfme-app** and optionally a **cfme-db** (for containerized database) exports.

Configuration is provided through the following role variables:

- **openshift_management_storage_nfs_external_hostname**
- **openshift_management_storage_nfs_base_dir**

The **openshift_management_storage_nfs_external_hostname** parameter must be set to the host name or IP of your external NFS server.

If **/exports** is not the parent directory to your exports then you must set the base directory via the **openshift_management_storage_nfs_base_dir** parameter.

For example, if your server export is **/exports/hosted/prod/cfme-app**, then you must set **openshift_management_storage_nfs_base_dir=/exports/hosted/prod**.

4.3.5.3. Cloud Provider

If you are using OpenShift Container Platform cloud provider integration for your storage class, Red Hat CloudForms can also use the cloud provider storage to back its required PVs. For this functionality to work, you must have configured the **openshift_cloudprovider_kind** variable (for AWS or GCE) and all associated parameters specific to your chosen cloud provider.

When the application is created using this storage class, the required PVs are automatically provisioned using the configured cloud provider storage integration.

There are no additional variables to configure the behavior of this storage class.

4.3.5.4. Preconfigured (Advanced)

The **preconfigured** storage class implies that you know exactly what you are doing and that all storage requirements have been taken care ahead of time. Typically this means that you have already created the correctly sized PVs. The installer will do nothing to modify any storage settings.

There are no additional variables to configure the behavior of this storage class.

4.4. RUNNING THE INSTALLER

4.4.1. Deploying Red Hat CloudForms During or After OpenShift Container Platform Installation

You can choose to deploy Red Hat CloudForms either during initial OpenShift Container Platform installation or after the cluster has been provisioned:

1. Ensure that **openshift_management_install_management** is set to **true** in your inventory file under the **[OSEv3:vars]** section:

```
[OSEv3:vars]
openshift_management_install_management=true
```

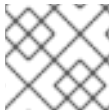
2. Set any other Red Hat CloudForms role variables in your inventory file as described in [Configuring Role Variables](#). Resources to assist in this are provided in [Example Inventory Files](#).
3. Choose which playbook to run depending on whether OpenShift Container Platform is already provisioned:
 - a. If you want to install Red Hat CloudForms at the same time you install your OpenShift Container Platform cluster, call the standard **config.yml** playbook as described in [Running the Installation Playbooks](#) to begin the OpenShift Container Platform cluster and Red Hat CloudForms installation.

- b. If you want to install Red Hat CloudForms on an already provisioned OpenShift Container Platform cluster, call the Red Hat CloudForms playbook directly to begin the installation:

```
# ansible-playbook -v [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    management/config.yml
```

4.4.2. Example Inventory Files

The following sections show example snippets of inventory files showing various configurations of Red Hat CloudForms on OpenShift Container Platform that can help you get started.



NOTE

See [Configuring Role Variables](#) for complete variable descriptions.

4.4.2.1. All Defaults

This example is the simplest, using all of the default values and choices. This results in a fully-containerized (podified) Red Hat CloudForms installation. All application components, as well as the PostgreSQL database, are created as pods in OpenShift Container Platform:

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
```

4.4.2.2. External NFS Storage

This is as the previous example, except that instead of using local NFS services in the cluster, it uses an existing, external NFS server (such as a storage appliance). Note the two new parameters:

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_storage_class=nfs_external ①
openshift_management_storage_nfs_external_hostname=nfs.example.com ②
```

① Set to **nfs_external**.

② Set to the host name of the NFS server.

If the external NFS host exports directories under a different parent directory, such as **/exports/hosted/prod**, add the following additional variable:

```
openshift_management_storage_nfs_base_dir=/exports/hosted/prod
```

4.4.2.3. Override PV Sizes

This example overrides the persistent volume (PV) sizes. PV sizes must be set via **openshift_management_template_parameters**, which ensures that the application and database are able to make claims on created PVs without interfering with each other:

```
[OSEv3:vars]
```

```
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_VOLUME_CAPACITY':
'10Gi', 'DATABASE_VOLUME_CAPACITY': '25Gi'}
```

4.4.2.4. Override Memory Requirements

In a test or proof-of-concept installation, you may need to reduce the application and database memory requirements to fit within your capacity. Note that reducing memory limits can result in reduced performance or a complete failure to initialize the application:

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_MEM_REQ': '3000Mi',
'POSTGRESQL_MEM_REQ': '1Gi', 'ANSIBLE_MEM_REQ': '512Mi'}
```

This example instructs the installer to process the application template with the parameter **APPLICATION_MEM_REQ** set to **3000Mi**, **POSTGRESQL_MEM_REQ** set to **1Gi**, and **ANSIBLE_MEM_REQ** set to **512Mi**.

These parameters can be combined with the parameters displayed in the previous example [Override PV Sizes](#).

4.4.2.5. External PostgreSQL Database

To use an external database, you must change the **openshift_management_app_template** parameter value to **cfme-template-ext-db**.

Additionally, database connection information must be supplied using the **openshift_management_template_parameters** variable. See [Configuring Role Variables](#) for more details.

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10',
'DATABASE_PORT': '5432', 'DATABASE_NAME': 'cfme'}
```



IMPORTANT

Ensure you are running PostgreSQL 9.5 or you may not be able to deploy the application successfully.

4.5. ENABLING CONTAINER PROVIDER INTEGRATION

4.5.1. Adding a Single Container Provider

After deploying Red Hat CloudForms on OpenShift Container Platform as described in [Running the Installer](#), there are two methods for enabling container provider integration. You can manually add OpenShift Container Platform as a container provider, or you can try the playbooks included with this role.

4.5.1.1. Adding Manually

See the following Red Hat CloudForms documentation for steps on manually adding your OpenShift Container Platform cluster as a container provider:

- [Integration with OpenShift Container Platform](#)

4.5.1.2. Adding Automatically

Automated container provider integration can be accomplished using the playbooks included with this role.

This playbook:

1. Gathers the necessary authentication secrets.
2. Finds the public routes to the Red Hat CloudForms application and the cluster API.
3. Makes a REST call to add the OpenShift Container Platform cluster as a container provider.

To run the container provider playbook:

```
# ansible-playbook -v [-i /path/to/inventory] \  
    /usr/share/ansible/openshift-ansible/playbooks/openshift-  
management/add_container_provider.yml
```

4.5.2. Multiple Container Providers

As well as providing playbooks to integrate your current OpenShift Container Platform cluster into your Red Hat CloudForms deployment, this role includes a script which allows you to add multiple container platforms as container providers in any arbitrary Red Hat CloudForms server. The container platforms can be OpenShift Container Platform or OpenShift Origin.

Using the multiple provider script requires manual configuration and setting an **EXTRA_VARS** parameter on the CLI when running the playbook.

4.5.2.1. Preparing the Script

To prepare the multiple provider script, complete the following manual configuration:

1. Copy the **/usr/share/ansible/openshift-ansible/roles/openshift_management/files/examples/container_providers.yml** example somewhere, such as **/tmp/cp.yml**. You will be modifying this file.
2. If you changed your Red Hat CloudForms name or password, update the **hostname**, **user**, and **password** parameters in the **management_server** key in the **container_providers.yml** file that you copied.
3. Fill in an entry under the **container_providers** key for each container platform cluster you want to add as container providers.
 - a. The following parameters must be configured:
 - **auth_key** - This is the token of a service account that has **cluster-admin** privileges.
 - **hostname** - This is the host name that points to the cluster API. Each container provider must have a unique host name.

- **name** - This is the name of the cluster to be displayed in the Red Hat CloudForms server container providers overview page. This must be unique.

TIP

To obtain the **auth_key** bearer token from your clusters:

```
$ oc serviceaccounts get-token -n management-infra management-admin
```

b. The following parameters may be optionally configured:

- **port** - Update this key if your container platform cluster runs the API on a port other than **8443**.
- **endpoint** - You may enable SSL verification (**verify_ssl**) or change the validation setting to **ssl-with-validation**. Support for custom trusted CA certificates is not currently available.

4.5.2.1.1. Example

As an example, consider the following scenario:

- You copied the **container_providers.yml** file to **/tmp/cp.yml**.
- You want to add two OpenShift Container Platform clusters.
- Your Red Hat CloudForms server runs on **mgmt.example.com**

For this scenario, you would customize **/tmp/cp.yml** as follows:

```
container_providers:
  - connection_configurations:
    - authentication: {auth_key: "<token>", authtype: bearer, type:
AuthToken} ❶
      endpoint: {role: default, security_protocol: ssl-without-
validation, verify_ssl: 0}
      hostname: "<provider_hostname1>"
      name: <display_name1>
      port: 8443
      type: "ManageIQ::Providers::Openshift::ContainerManager"
  - connection_configurations:
    - authentication: {auth_key: "<token>", authtype: bearer, type:
AuthToken} ❷
      endpoint: {role: default, security_protocol: ssl-without-
validation, verify_ssl: 0}
      hostname: "<provider_hostname2>"
      name: <display_name2>
      port: 8443
      type: "ManageIQ::Providers::Openshift::ContainerManager"
management_server:
  hostname: "<hostname>"
  user: <user_name>
  password: <password>
```

- 1 2 Replace **<token>** with the management token for this cluster.

4.5.2.2. Running the Playbook

To run the multiple-providers integration script, you must provide the path to the container providers configuration file as an **EXTRA_VARS** parameter to the **ansible-playbook** command. Use the **-e** (or **-extra-vars**) parameter to set **container_providers_config** to the configuration file path:

```
# ansible-playbook -v [-i /path/to/inventory] \
    -e container_providers_config=/tmp/cp.yml \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/add_many_container_providers.yml
```

After the playbook completes, you should find two new container providers in your Red Hat CloudForms service. Navigate to the **Compute** → **Containers** → **Providers** page to see an overview.

4.5.3. Refreshing Providers

After adding either a single or multiple container providers, the new provider(s) must be refreshed in Red Hat CloudForms to get all the latest data about the container provider and the containers being managed. This involves navigating to each provider in the Red Hat CloudForms web console and clicking a refresh button for each.

See the following Red Hat CloudForms documentation for steps:

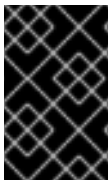
- [Managing Providers](#)

4.6. UNINSTALLING RED HAT CLOUDFORMS

4.6.1. Running the Uninstall Playbook

To uninstall and erase a deployed Red Hat CloudForms installation from OpenShift Container Platform, run the following playbook:

```
# ansible-playbook -v [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/uninstall.yml
```



IMPORTANT

NFS export definitions and data stored on NFS exports are not automatically removed. You are urged to manually erase any data from old application or database deployments before attempting to initialize a new deployment.

4.6.2. Troubleshooting

Failure to erase old PostgreSQL data can result in cascading errors, causing the **postgresql** pod to enter a **crashloopbackoff** state. This blocks the **cfme** pod from ever starting. The cause of the **crashloopbackoff** is due to incorrect file permissions on the database NFS export created during a previous deployment.

To continue, erase all data from the PostgreSQL export and delete the pod (not the deployer pod). For example, if you had the following pods:

```
$ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
httpd-1-cx7fk	1/1	Running	1	21h
cfme-0	0/1	Running	1	21h
memcached-1-vkc7p	1/1	Running	1	21h
postgresql-1-deploy	1/1	Running	1	21h
postgresql-1-6w2t4	0/1	CrashLoopBackOff	1	21h

Then you would:

1. Erase the data from the database NFS export.
2. Run:

```
$ oc delete postgresql-1-6w2t4
```

The PostgreSQL deployer pod will try to scale up a new **postgresql** pod to replace the one you deleted. After the **postgresql** pod is running, the **cfme** pod will stop blocking and begin application initialization.

CHAPTER 5. MASTER AND NODE CONFIGURATION

The **openshift start** command and its subcommands (**master** to launch a [master server](#) and **node** to launch a [node server](#)) take a limited set of arguments that are sufficient for launching servers in a development or experimental environment.

However, these arguments are insufficient to describe and control the full set of configuration and security options that are necessary in a production environment. You must provide those options in the [Master host files](#), at `/etc/origin/master/master-config.yaml` and the [node configuration maps](#):

These files define options including overriding the default plug-ins, connecting to etcd, automatically creating service accounts, building image names, customizing project requests, configuring volume plug-ins, and much more.

This topic covers the available options for customizing your OpenShift Container Platform master and node hosts, and shows you how to make changes to the configuration after installation.

These files are fully specified with no default values. Therefore, an empty value indicates that you want to start up with an empty value for that parameter. This makes it easy to reason about exactly what your configuration is, but it also makes it difficult to remember all of the options to specify. To make this easier, the configuration files can be created with the **--write-config** option and then used with the **--config** option.

5.1. INSTALLATION DEPENDENCIES

Production environments should be installed using the standard [cluster installation](#) process. In production environments, it is a good idea to use [multiple masters](#) for the purposes of [high availability](#) (HA). A cluster architecture of three masters is recommended, and [HAproxy](#) is the recommended solution for this.

CAUTION

If etcd is installed on the *master hosts*, you must configure your cluster to use at least three masters, because etcd would not be able to decide which one is authoritative. The only way to successfully run only two masters is if you install etcd on hosts other than the masters.

5.2. CONFIGURING MASTERS AND NODES

The method you use to configure your master and node configuration files must match the method that was used to install your OpenShift Container Platform cluster. If you followed the standard [cluster installation](#) process, then make your configuration changes in the Ansible inventory file.

5.3. MAKING CONFIGURATION CHANGES USING ANSIBLE

For this section, familiarity with Ansible is assumed.

Only a portion of the available host configuration options are [exposed to Ansible](#). After an OpenShift Container Platform install, Ansible creates an inventory file with some substituted values. Modifying this inventory file and re-running the Ansible installer playbook is how you customize your OpenShift Container Platform cluster.

While OpenShift Container Platform supports using Ansible for cluster installation, using an Ansible playbook and inventory file, you can also use other management tools, such as [Puppet](#), [Chef](#), [Salt](#)).

Use Case: Configuring the cluster to use HTTPasswd authentication



NOTE

- This use case assumes you have already set up [SSH keys](#) to all the nodes referenced in the playbook.
- The **htpasswd** utility is in the **httpd-tools** package:

```
# yum install httpd-tools
```

To modify the Ansible inventory and make configuration changes:

1. Open the **./hosts** inventory file.
2. Add the following new variables to the **[OSEv3:vars]** section of the file:

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'<name>': '<hashed-password>',
'<name>': '<hashed-password>'}
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>
```

For HTPasswd authentication, you can use either the **openshift_master_htpasswd_users** variable to create the specified user(s) and password(s) or the **openshift_master_htpasswd_file** variable to specify a pre-generated flat file (the *htpasswd* file) with the users and passwords already created.

Because OpenShift Container Platform requires a hashed password to configure HTPasswd authentication, you can use the **htpasswd** command, [as shown in the following section](#), to generate the hashed password(s) for your user(s) or to create the flat file with the users and associated hashed passwords.

The following example changes the authentication method from the default **deny all** setting to **htpasswd** and use the specified file to generate user IDs and passwords for the **jsmith** and **bloblaw** users.

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'jsmith':
'$apr1$wIwXkFLI$bAygTKGmPOqaJftB', 'bloblaw':
'7IRJ$20DmeLoxf4I6sUEKfiA$2aDJqLJe'}
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>
```

3. Re-run the ansible playbook for these modifications to take effect:

```
$ ansible-playbook -b -i ./hosts ~/src/openshift-
ansible/playbooks/deploy_cluster.yml
```

The playbook updates the configuration, and restarts the OpenShift Container Platform master service to apply the changes.

You have now modified the master and node configuration files using Ansible, but this is just a simple use case. From here you can see which [master](#) and [node configuration](#) options are [exposed to Ansible](#) and customize your own Ansible inventory.

5.3.1. Using the `htpasswd` command

To configure the OpenShift Container Platform cluster to use HTTPasswd authentication, you need at least one user with a hashed password to include in the [inventory file](#).

You can:

- [Generate the username and password](#) to add directly to the `./hosts` inventory file.
- [Create a flat file](#) to pass the credentials to the `./hosts` inventory file.

To create a user and hashed password:

1. Run the following command to add the specified user:

```
$ htpasswd -n <user_name>
```



NOTE

You can include the `-b` option to supply the password on the command line:

```
$ htpasswd -nb <user_name> <password>
```

2. Enter and confirm a clear-text password for the user.
For example:

```
$ htpasswd -n myuser
New password:
Re-type new password:
myuser:$apr1$vdW.cI3j$WSKIOzUPs6Q
```

The command generates a hashed version of the password.

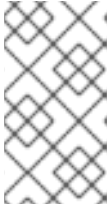
You can then use the hashed password when configuring [HTTPasswd authentication](#). The hashed password is the string after the `:`. In the above example, you would enter:

```
openshift_master_htpasswd_users={'myuser':
'$apr1$wIwXkFLI$bAygTISk2eKGmqAJftB'}
```

To create a flat file with a user name and hashed password:

1. Execute the following command:

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```



NOTE

You can include the **-b** option to supply the password on the command line:

```
$ htpasswd -c -b <user_name> <password>
```

2. Enter and confirm a clear-text password for the user.
For example:

```
htpasswd -c users.htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

The command generates a file that includes the user name and a hashed version of the user's password.

You can then use the password file when configuring [HTPasswd authentication](#).



NOTE

For more information on the **htpasswd** command, see [HTPasswd Identity Provider](#).

5.4. MAKING MANUAL CONFIGURATION CHANGES

Use Case: Configure the cluster to use HTPasswd authentication

To manually modify a configuration file:

1. Open the configuration file you want to modify, which in this case is the ***/etc/origin/master/master-config.yaml*** file:
2. Add the following new variables to the **identityProviders** stanza of the file:

```
oauthConfig:
  ...
identityProviders:
- name: my_htpasswd_provider
  challenge: true
  login: true
  mappingMethod: claim
  provider:
    apiVersion: v1
    kind: HTPasswdPasswordIdentityProvider
    file: /path/to/users.htpasswd
```

3. Save your changes and close the file.
4. Restart the master for the changes to take effect:

```
$ master-restart api
$ master-restart controllers
```

You have now manually modified the master and node configuration files, but this is just a simple use case. From here you can see all the [master](#) and [node configuration](#) options, and further customize your own cluster by making further modifications.



NOTE

To modify a node in your cluster, update the [node configuration maps](#) as needed. Do not manually edit the **node-config.yaml** file.

5.5. MASTER CONFIGURATION FILES

This section reviews parameters mentioned in the **master-config.yaml** file.

You can [create a new master configuration file](#) to see the valid options for your installed version of OpenShift Container Platform.



IMPORTANT

Whenever you modify the **master-config.yaml** file, you must restart the master for the changes to take effect. See [Restarting OpenShift Container Platform services](#).

5.5.1. Admission Control Configuration

Table 5.1. Admission Control Configuration Parameters

Parameter Name	Description
AdmissionConfig	Contains the admission control plug-in configuration. OpenShift Container Platform has a configurable list of admission controller plug-ins that are triggered whenever API objects are created or modified. This option allows you to override the default list of plug-ins; for example, disabling some plug-ins, adding others, changing the ordering, and specifying configuration. Both the list of plug-ins and their configuration can be controlled from Ansible.
APIServerArguments	Key-value pairs that will be passed directly to the Kube API server that match the API servers' command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in KubernetesMasterConfig , which may cause invalid configurations. Use APIServerArguments with the event-ttl value to store events in etcd. The default is 2h , but it can be set to less to prevent memory growth: <pre>apiServerArguments: event-ttl: - "15m"</pre>

Parameter Name	Description
ControllerArguments	Key-value pairs that will be passed directly to the Kube controller manager that match the controller manager's command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in KubernetesMasterConfig , which may cause invalid configurations.
DefaultAdmissionConfig	Used to enable or disable various admission plug-ins. When this type is present as the configuration object under pluginConfig and if the admission plug-in supports it, this will cause an off by default admission plug-in to be enabled.
PluginConfig	Allows specifying a configuration file per admission control plug-in.
PluginOrderOverride	A list of admission control plug-in names that will be installed on the master. Order is significant. If empty, a default list of plug-ins is used.
SchedulerArguments	Key-value pairs that will be passed directly to the Kube scheduler that match the scheduler's command line arguments. These are not migrated, but if you reference a value that does not exist the server will not start. These values may override other settings in KubernetesMasterConfig , which may cause invalid configurations.

5.5.2. Asset Configuration

Table 5.2. Asset Configuration Parameters

Parameter Name	Description
AssetConfig	<p>If present, then the asset server starts based on the defined parameters. For example:</p> <pre> assetConfig: logoutURL: "" masterPublicURL: https://master.ose32.example.com:8443 publicURL: https://master.ose32.example.com:8443/console/ servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: "" keyFile: master.server.key maxRequestsInFlight: 0 requestTimeoutSeconds: 0 </pre>

Parameter Name	Description
corsAllowedOrigins	To access the API server from a web application using a different host name, you must whitelist that host name by specifying corsAllowedOrigins in the configuration field or by specifying the --cors-allowed-origins option on openshift start . No pinning or escaping is done to the value. See Web Console for example usage.
DisabledFeatures	A list of features that should not be started. You will likely want to set this as null . It is very unlikely that anyone will want to manually disable features and that is not encouraged.
Extensions	Files to serve from the asset server file system under a subcontext.
ExtensionDevelopment	When set to true , tells the asset server to reload extension scripts and stylesheets for every request rather than only at startup. It lets you develop extensions without having to restart the server for every change.
ExtensionProperties	Key- (string) and value- (string) pairs that will be injected into the console under the global variable OPENSIFT_EXTENSION_PROPERTIES .
ExtensionScripts	File paths on the asset server files to load as scripts when the web console loads.
ExtensionStylesheets	File paths on the asset server files to load as style sheets when the web console loads.
LoggingPublicURL	The public endpoint for logging (optional).
LogoutURL	An optional, absolute URL to redirect web browsers to after logging out of the web console. If not specified, the built-in logout page is shown.
MasterPublicURL	How the web console can access the OpenShift Container Platform server.
MetricsPublicURL	The public endpoint for metrics (optional).
PublicURL	URL of the asset server.

5.5.3. Authentication and Authorization Configuration

Table 5.3. Authentication and Authorization Parameters

Parameter Name	Description
authConfig	Holds authentication and authorization configuration options.

Parameter Name	Description
AuthenticationCacheSize	Indicates how many authentication results should be cached. If 0, the default cache size is used.
AuthorizationCacheTTL	Indicates how long an authorization result should be cached. It takes a valid time duration string (e.g. "5m"). If empty, you get the default timeout. If zero (e.g. "0m"), caching is disabled.

5.5.4. Controller Configuration

Table 5.4. Controller Configuration Parameters

Parameter Name	Description
Controllers	List of the controllers that should be started. If set to none , no controllers will start automatically. The default value is * which will start all controllers. When using * , you may exclude controllers by prepending a - in front of their name. No other values are recognized at this time.
ControllerLeaseTTL	Enables controller election, instructing the master to attempt to acquire a lease before controllers start and renewing it within a number of seconds defined by this value. Setting this value non-negative forces pauseControllers=true . This value defaults off (0, or omitted) and controller election can be disabled with -1.
PauseControllers	Instructs the master to not automatically start controllers, but instead to wait until a notification to the server is received before launching them.

5.5.5. etcd Configuration

Table 5.5. etcd Configuration Parameters

Parameter Name	Description
Address	The advertised host:port for client connections to etcd.
etcdClientInfo	<p>Contains information about how to connect to etcd. Specifies if etcd is run as embedded or non-embedded, and the hosts. The rest of the configuration is handled by the Ansible inventory. For example:</p> <pre>etcdClientInfo: ca: ca.crt certFile: master.etcd-client.crt keyFile: master.etcd-client.key urls: - https://m1.aos.example.com:4001</pre>

Parameter Name	Description
etcdConfig	<p>If present, then etcd starts based on the defined parameters. For example:</p> <pre> etcdConfig: address: master.ose32.example.com:4001 peerAddress: master.ose32.example.com:7001 peerServingInfo: bindAddress: 0.0.0.0:7001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key servingInfo: bindAddress: 0.0.0.0:4001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key storageDirectory: /var/lib/origin/openshift.local.etcd </pre>
etcdStorageConfig	Contains information about how API resources are stored in etcd. These values are only relevant when etcd is the backing store for the cluster.
KubernetesStoragePrefix	The path within etcd that the Kubernetes resources will be rooted under. This value, if changed, will mean existing objects in etcd will no longer be located. The default value is kubernetes.io .
KubernetesStorageVersion	The API version that Kubernetes resources in etcd should be serialized to. This value should not be advanced until all clients in the cluster that read from etcd have code that allows them to read the new version.
OpenShiftStoragePrefix	The path within etcd that the OpenShift Container Platform resources will be rooted under. This value, if changed, will mean existing objects in etcd will no longer be located. The default value is openshift.io .
OpenShiftStorageVersion	API version that OS resources in etcd should be serialized to. This value should not be advanced until all clients in the cluster that read from etcd have code that allows them to read the new version.
PeerAddress	The advertised host:port for peer connections to etcd .
PeerServingInfo	Describes how to start serving the etcd peer.

Parameter Name	Description
ServingInfo	Describes how to start serving. For example: <pre>servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: ca.crt keyFile: master.server.key maxRequestsInFlight: 500 requestTimeoutSeconds: 3600</pre>
StorageDir	The path to the <i>etcd</i> storage directory.

5.5.6. Grant Configuration

Table 5.6. Grant Configuration Parameters

Parameter Name	Description
GrantConfig	Describes how to handle grants.
GrantHandlerAuto	Auto-approves client authorization grant requests.
GrantHandlerDeny	Auto-denies client authorization grant requests.
GrantHandlerPrompt	Prompts the user to approve new client authorization grant requests.
Method	Determines the default strategy to use when an OAuth client requests a grant. This method will be used only if the specific OAuth client does not provide a strategy of their own. Valid grant handling methods are: <ul style="list-style-type: none"> • auto: always approves grant requests, useful for trusted clients • prompt: prompts the end user for approval of grant requests, useful for third-party clients • deny: always denies grant requests, useful for black-listed clients

5.5.7. Image Configuration

Table 5.7. Image Configuration Parameters

Parameter Name	Description
Format	The format of the name to be built for the system component.

Parameter Name	Description
Latest	Determines if the latest tag will be pulled from the registry.

5.5.8. Image Policy Configuration

Table 5.8. Image Policy Configuration Parameters

Parameter Name	Description
DisableScheduledImport	Allows scheduled background import of images to be disabled.
MaxImagesBulkImportedPerRepository	Controls the number of images that are imported when a user does a bulk import of a Docker repository. This number defaults to 5 to prevent users from importing large numbers of images accidentally. Set -1 for no limit.
MaxScheduledImageImportsPerMinute	The maximum number of scheduled image streams that will be imported in the background per minute. The default value is 60.
ScheduledImageImportMinimumIntervalSeconds	The minimum number of seconds that can elapse between when image streams scheduled for background import are checked against the upstream repository. The default value is 15 minutes.
AllowedRegistriesForImport	Limits the docker registries that normal users may import images from. Set this list to the registries that you trust to contain valid Docker images and that you want applications to be able to import from. Users with permission to create Images or ImageStreamMappings via the API are not affected by this policy - typically only administrators or system integrations will have those permissions.
InternalRegistryHostname	Sets the hostname for the default internal image registry. The value must be in hostname[:port] format. For backward compatibility, users can still use OPENSIFT_DEFAULT_REGISTRY environment variable but this setting overrides the environment variable. When this is set, the internal registry must have its hostname set as well. See setting the registry hostname for more details.
ExternalRegistryHostname	ExternalRegistryHostname sets the hostname for the default external image registry. The external hostname should be set only when the image registry is exposed externally. The value is used in publicDockerImageRepository field in ImageStreams. The value must be in hostname[:port] format.

5.5.9. Kubernetes Master Configuration

Table 5.9. Kubernetes Master Configuration Parameters

Parameter Name	Description
APILevels	A list of API levels that should be enabled on startup, v1 as examples.
DisabledAPIGroupVersions	A map of groups to the versions (or *) that should be disabled.
KubeletClientInfo	Contains information about how to connect to kubelets.
KubernetesMasterConfig	Contains information about how to connect to kubelet's KubernetesMasterConfig. If present, then start the kubernetes master with this process.
MasterCount	The number of expected masters that should be running. This value defaults to 1 and may be set to a positive integer, or if set to -1, indicates this is part of a cluster.
MasterIP	The public IP address of Kubernetes resources. If empty, the first result from net.InterfaceAddrs will be used.
MasterKubeConfig	File name for the <i>.kubeconfig</i> file that describes how to connect this node to the master.
ServicesNodePortRange	The range to use for assigning service public ports on a host. Default 30000-32767.
ServicesSubnet	The subnet to use for assigning service IPs.
StaticNodeNames	The list of nodes that are statically known.

5.5.10. Network Configuration

Choose the CIDRs in the following parameters carefully, because the IPv4 address space is shared by all users of the nodes. OpenShift Container Platform reserves CIDRs from the IPv4 address space for its own use, and reserves CIDRs from the IPv4 address space for addresses that are shared between the external user and the cluster.

Table 5.10. Network Configuration Parameters

Parameter Name	Description
ClusterNetworkCIDR	The CIDR string to specify the global overlay network's L3 space. This is reserved for the internal use of the cluster networking.

Parameter Name	Description
externalIPNetworkCIDRs	Controls what values are acceptable for the service external IP field. If empty, no externalIP may be set. It may contain a list of CIDRs which are checked for access. If a CIDR is prefixed with ! , IPs in that CIDR will be rejected. Rejections will be applied first, then the IP checked against one of the allowed CIDRs. You must ensure this range does not overlap with your nodes, pods, or service CIDRs for security reasons.
HostSubnetLength	The number of bits to allocate to each host's subnet. For example, 8 would mean a /24 network on the host.
ingressIPNetworkCIDR	Controls the range to assign ingress IPs from for services of type LoadBalancer on bare metal. It may contain a single CIDR that it will be allocated from. By default 172.17.0.0/16 is configured. For security reasons, you should ensure that this range does not overlap with the CIDRs reserved for external IPs, nodes, pods, or services.
HostSubnetLength	The number of bits to allocate to each host's subnet. For example, 8 would mean a /24 network on the host.

Parameter Name	Description
NetworkConfig	<p>To be passed to the compiled-in-network plug-in. Many of the options here can be controlled in the Ansible inventory.</p> <ul style="list-style-type: none"> • NetworkPluginName (string) • ClusterNetworkCIDR (string) • HostSubnetLength (unsigned integer) • ServiceNetworkCIDR (string) • externalIPNetworkCIDRs (string array): Controls which values are acceptable for the service external IP field. If empty, no external IP may be set. It can contain a list of CIDRs which are checked for access. If a CIDR is prefixed with !, then IPs in that CIDR are rejected. Rejections are applied first, then the IP is checked against one of the allowed CIDRs. For security purposes, you should ensure this range does not overlap with your nodes, pods, or service CIDRs. <p>For Example:</p> <pre>networkConfig: clusterNetworks - cidr: 10.3.0.0/16 hostSubnetLength: 8 networkPluginName: example/openshift-ovs-subnet # serviceNetworkCIDR must match kubernetesMasterConfig.servicesSubnet serviceNetworkCIDR: 179.29.0.0/16</pre>
NetworkPluginName	The name of the network plug-in to use.
ServiceNetwork	The CIDR string to specify the service networks.

5.5.11. OAuth Authentication Configuration

Table 5.11. OAuth Configuration Parameters

Parameter Name	Description
AlwaysShowProviderSelection	Forces the provider selection page to render even when there is only a single provider.
AssetPublicURL	Used for building valid client redirect URLs for external access.
Error	A path to a file containing a go template used to render error pages during the authentication or grant flow. If unspecified, the default error page is used.

Parameter Name	Description
IdentityProviders	Ordered list of ways for a user to identify themselves.
Login	A path to a file containing a go template used to render the login page. If unspecified, the default login page is used.
MasterCA	CA for verifying the TLS connection back to the MasterURL .
MasterPublicURL	Used for building valid client redirect URLs for external access.
MasterURL	Used for making server-to-server calls to exchange authorization codes for access tokens.
OAuthConfig	<p>If present, then the /oauth endpoint starts based on the defined parameters. For example:</p> <pre> oauthConfig: assetPublicURL: https://master.ose32.example.com:8443/console/ grantConfig: method: auto identityProviders: - challenge: true login: true mappingMethod: claim name: htpasswd_all provider: apiVersion: v1 kind: HTTPasswdPasswordIdentityProvider file: /etc/origin/openshift-passwd masterCA: ca.crt masterPublicURL: https://master.ose32.example.com:8443 masterURL: https://master.ose32.example.com:8443 sessionConfig: sessionMaxAgeSeconds: 3600 sessionName: ssn sessionSecretsFile: /etc/origin/master/session-secrets.yaml tokenConfig: accessTokenMaxAgeSeconds: 86400 authorizeTokenMaxAgeSeconds: 500 </pre>
OAuthTemplates	Allows for customization of pages like the login page.
ProviderSelection	A path to a file containing a go template used to render the provider selection page. If unspecified, the default provider selection page is used.

Parameter Name	Description
SessionConfig	Holds information about configuring sessions.
Templates	Allows you to customize pages like the login page.
TokenConfig	Contains options for authorization and access tokens.

5.5.12. Project Configuration

Table 5.12. Project Configuration Parameters

Parameter Name	Description
DefaultNodeSelector	Holds default project node label selector.

Parameter Name	Description
ProjectConfig	<p>Holds information about project creation and defaults:</p> <ul style="list-style-type: none"> • DefaultNodeSelector (string): Holds the default project node label selector. • ProjectRequestMessage (string): The string presented to a user if they are unable to request a project via the projectrequest API endpoint. • ProjectRequestTemplate (string): The template to use for creating projects in response to projectrequest. It is in the format <code><namespace>/<template></code>. It is optional, and if it is not specified, a default template is used. • SecurityAllocator: Controls the automatic allocation of UIDs and MCS labels to a project. If nil, allocation is disabled: <ul style="list-style-type: none"> ◦ mcsAllocatorRange (string): Defines the range of MCS categories that will be assigned to namespaces. The format is <code><prefix>/<numberOfLabels>[,<maxCategory>]</code>. The default is <code>s0/2</code> and will allocate from <code>c0 → c1023</code>, which means a total of 535k labels are available. If this value is changed after startup, new projects may receive labels that are already allocated to other projects. The prefix may be any valid SELinux set of terms (including user, role, and type). However, leaving the prefix at its default allows the server to set them automatically. For example, <code>s0:/2</code> would allocate labels from <code>s0:c0,c0</code> to <code>s0:c511,c511</code> whereas <code>s0:/2,512</code> would allocate labels from <code>s0:c0,c0,c0</code> to <code>s0:c511,c511,511</code>. ◦ mcsLabelsPerProject (integer): Defines the number of labels to reserve per project. The default is <code>5</code> to match the default UID and MCS ranges. ◦ uidAllocatorRange (string): Defines the total set of Unix user IDs (UIDs) automatically allocated to projects, and the size of the block each namespace gets. For example, <code>1000-1999/10</code> would allocate ten UIDs per namespace, and would be able to allocate up to 100 blocks before running out of space. The default is to allocate from 1 billion to 2 billion in 10k blocks, which is the expected size of ranges for container images when user namespaces are started.
ProjectRequestMessage	The string presented to a user if they are unable to request a project via the project request API endpoint.
ProjectRequestTemplate	The template to use for creating projects in response to projectrequest . It is in the format <code>namespace/template</code> and it is optional. If it is not specified, a default template is used.

5.5.13. Scheduler Configuration

Table 5.13. Scheduler Configuration Parameters

Parameter Name	Description
SchedulerConfigFile	Points to a file that describes how to set up the scheduler. If empty, you get the default scheduling rules

5.5.14. Security Allocator Configuration

Table 5.14. Security Allocator Parameters

Parameter Name	Description
MCSAllocatorRange	Defines the range of MCS categories that will be assigned to namespaces. The format is <prefix>/<numberOfLabels>[, <maxCategory>] . The default is s0/2 and will allocate from c0 to c1023, which means a total of 535k labels are available (1024 choose 2 ~ 535k). If this value is changed after startup, new projects may receive labels that are already allocated to other projects. Prefix may be any valid SELinux set of terms (including user, role, and type), although leaving them as the default will allow the server to set them automatically.
SecurityAllocator	Controls the automatic allocation of UIDs and MCS labels to a project. If nil, allocation is disabled.
UIDAllocatorRange	Defines the total set of Unix user IDs (UIDs) that will be allocated to projects automatically, and the size of the block each namespace gets. For example, 1000-1999/10 will allocate ten UIDs per namespace, and will be able to allocate up to 100 blocks before running out of space. The default is to allocate from 1 billion to 2 billion in 10k blocks (which is the expected size of the ranges container images will use once user namespaces are started).

5.5.15. Service Account Configuration

Table 5.15. Service Account Configuration Parameters

Parameter Name	Description
LimitSecretReferences	Controls whether or not to allow a service account to reference any secret in a namespace without explicitly referencing them.
ManagedNames	A list of service account names that will be auto-created in every namespace. If no names are specified, the ServiceAccountsController will not be started.
MasterCA	The CA for verifying the TLS connection back to the master. The service account controller will automatically inject the contents of this file into pods so they can verify connections to the master.

Parameter Name	Description
PrivateKeyFile	A file containing a PEM-encoded private RSA key, used to sign service account tokens. If no private key is specified, the service account TokensController will not be started.
PublicKeyFiles	A list of files, each containing a PEM-encoded public RSA key. If any file contains a private key, the public portion of the key is used. The list of public keys is used to verify presented service account tokens. Each key is tried in order until the list is exhausted or verification succeeds. If no keys are specified, no service account authentication will be available.
ServiceAccountConfig	<p>Holds options related to service accounts:</p> <ul style="list-style-type: none"> • LimitSecretReferences (boolean): Controls whether or not to allow a service account to reference any secret in a namespace without explicitly referencing them. • ManagedNames (string): A list of service account names that will be auto-created in every namespace. If no names are specified, then the ServiceAccountsController will not be started. • MasterCA (string): The certificate authority for verifying the TLS connection back to the master. The service account controller will automatically inject the contents of this file into pods so that they can verify connections to the master. • PrivateKeyFile (string): Contains a PEM-encoded private RSA key, used to sign service account tokens. If no private key is specified, then the service account TokensController will not be started. • PublicKeyFiles (string): A list of files, each containing a PEM-encoded public RSA key. If any file contains a private key, then OpenShift Container Platform uses the public portion of the key. The list of public keys is used to verify service account tokens; each key is tried in order until either the list is exhausted or verification succeeds. If no keys are specified, then service account authentication will not be available.

5.5.16. Serving Information Configuration

Table 5.16. Serving Information Configuration Parameters

Parameter Name	Description
AllowRecursiveQueries	Allows the DNS server on the master to answer queries recursively. Note that open resolvers can be used for DNS amplification attacks and the master DNS should not be made accessible to public networks.
BindAddress	The ip:port to serve on.

Parameter Name	Description
BindNetwork	Controls limits and behavior for importing images.
CertFile	A file containing a PEM-encoded certificate.
CertInfo	TLS cert information for serving secure traffic.
ClientCA	The certificate bundle for all the signers that you recognize for incoming client certificates.
dnsConfig	<p>If present, then start the DNS server based on the defined parameters. For example:</p> <pre>dnsConfig: bindAddress: 0.0.0.0:8053 bindNetwork: tcp4</pre>
DNSDomain	Holds the domain suffix.
DNSIP	Holds the IP.
KeyFile	A file containing a PEM-encoded private key for the certificate specified by CertFile .
MasterClientConnection Overrides	Provides overrides to the client connection used to connect to the master.
MaxRequestsInFlight	The number of concurrent requests allowed to the server. If zero, no limit.
NamedCertificates	A list of certificates to use to secure requests to specific host names.
RequestTimeoutSecond	The number of seconds before requests are timed out. The default is 60 minutes. If -1, there is no limit on requests.
ServingInfo	The HTTP serving information for the assets.

5.5.17. Volume Configuration

Table 5.17. Volume Configuration Parameters

Parameter Name	Description
DynamicProvisioningEnabled	A boolean to enable or disable dynamic provisioning. Default is true .

Parameter Name	Description
FSGroup	Enables local storage quotas on each node for each FSGroup. At present this is only implemented for emptyDir volumes, and if the underlying volumeDirectory is on an XFS filesystem.
MasterVolumeConfig	Contains options for configuring volume plug-ins in the master node.
NodeVolumeConfig	Contains options for configuring volumes on the node.
VolumeConfig	Contains options for configuring volume plug-ins in the node: <ul style="list-style-type: none"> • DynamicProvisioningEnabled (boolean): Default value is true, and toggles dynamic provisioning off when false.
VolumeDirectory	The directory that volumes are stored under.

5.5.18. Basic Audit

Audit provides a security-relevant chronological set of records documenting the sequence of activities that have affected system by individual users, administrators, or other components of the system.

Audit works at the API server level, logging all requests coming to the server. Each audit log contains two entries:

1. The request line containing:
 - a. A Unique ID allowing to match the response line (see #2)
 - b. The source IP of the request
 - c. The HTTP method being invoked
 - d. The original user invoking the operation
 - e. The impersonated user for the operation (**self** meaning himself)
 - f. The impersonated group for the operation (**lookup** meaning user's group)
 - g. The namespace of the request or <none>
 - h. The URI as requested
2. The response line containing:
 - a. The unique ID from #1
 - b. The response code

Example output for user **admin** asking for a list of pods:

```
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" ip="127.0.0.1"
```

```
method="GET" user="admin" as="<self>" asgroups="<lookup>"
namespace="default" uri="/api/v1/namespaces/default/pods"
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" response="200"
```

The **openshift_master_audit_config** variable enables API service auditing. It takes an array of the following options:

Table 5.18. Audit Configuration Parameters

Parameter Name	Description
enabled	A boolean to enable or disable audit logs. Default is false .
auditFilePath	File path where the requests should be logged to. If not set, logs are printed to master logs.
maximumFileRetentionDays	Specifies maximum number of days to retain old audit log files based on the time stamp encoded in their filename.
maximumRetainedFiles	Specifies the maximum number of old audit log files to retain.
maximumFileSizeMegabytes	Specifies maximum size in megabytes of the log file before it gets rotated. Defaults to 100MB.

Example Audit Configuration

```
auditConfig:
  auditFilePath: "/var/log/audit-ocp.log"
  enabled: true
  maximumFileRetentionDays: 10
  maximumFileSizeMegabytes: 10
  maximumRetainedFiles: 10
```

Advanced Setup for the Audit Log

If you want more advanced setup for the audit log, you can use:

```
openshift_master_audit_config={"enabled": true}
```

The directory in **auditFilePath** will be created if it does not exist.

```
openshift_master_audit_config={"enabled": true, "auditFilePath":
"/var/log/openpaas-oscp-audit/openpaas-oscp-audit.log",
"maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500,
"maximumRetainedFiles": 5}
```

5.5.19. Advanced Audit

The advanced audit feature provides several improvements over the [basic audit functionality](#), including fine-grained events filtering and multiple output back ends.

To enable the advanced audit feature, provide the following values in the `openshift_master_audit_config` parameter

```
openshift_master_audit_config={"enabled": true, "auditFilePath":
"/var/log/oscp-audit/-oscp-audit.log", "maximumFileRetentionDays": 14,
"maximumFileSizeMegabytes": 500, "maximumRetainedFiles": 5, "policyFile":
"/etc/security/adv-audit.yaml", "logFormat":"json"}
```



IMPORTANT

The policy file `/etc/security/adv-audit.yaml` must be available on each master node.

The following table contains additional options you can use.

Table 5.19. Advanced Audit Configuration Parameters

Parameter Name	Description
<code>policyFile</code>	Path to the file that defines the audit policy configuration.
<code>policyConfiguration</code>	An embedded audit policy configuration.
<code>logFormat</code>	Specifies the format of the saved audit logs. Allowed values are legacy (the format used in basic audit), and json .
<code>webHookKubeConfig</code>	Path to a .kubeconfig -formatted file that defines the audit webhook configuration, where the events are sent to.
<code>webHookMode</code>	Specifies the strategy for sending audit events. Allowed values are block (blocks processing another event until the previous has fully processed) and batch (buffers events and delivers in batches).



IMPORTANT

To enable the advanced audit feature, you must provide either `policyFile` or `policyConfiguration` describing the audit policy rules:

Sample Audit Policy Configuration

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  # Do not log watch requests by the "system:kube-proxy" on endpoints or
  # services
  - level: None ①
    users: ["system:kube-proxy"] ②
    verbs: ["watch"] ③
    resources: ④
      - group: ""
```



```

    resources: ["endpoints", "services"]

# Do not log authenticated requests to certain non-resource URL paths.
- level: None
  userGroups: ["system:authenticated"] 5
  nonResourceURLs: 6
  - "/api*" # Wildcard matching.
  - "/version"

# Log the request body of configmap changes in kube-system.
- level: Request
  resources:
  - group: "" # core API group
    resources: ["configmaps"]
  # This rule only applies to resources in the "kube-system" namespace.
  # The empty string "" can be used to select non-namespaced resources.
  namespaces: ["kube-system"] 7

# Log configmap and secret changes in all other namespaces at the
metadata level.
- level: Metadata
  resources:
  - group: "" # core API group
    resources: ["secrets", "configmaps"]

# Log all other resources in core and extensions at the request level.
- level: Request
  resources:
  - group: "" # core API group
  - group: "extensions" # Version of group should NOT be included.

# A catch-all rule to log all other requests at the Metadata level.
- level: Metadata 8

# Log login failures from the web console or CLI. Review the logs and
refine your policies.
- level: Metadata
  nonResourceURLs:
  - /login* 9
  - /oauth* 10

```

1 8 There are four possible levels every event can be logged at:

- **None** - Do not log events that match this rule.
- **Metadata** - Log request metadata (requesting user, time stamp, resource, verb, etc.), but not request or response body. This is the same level as the one used in basic audit.
- **Request** - Log event metadata and request body, but not response body.
- **RequestResponse** - Log event metadata, request, and response bodies.

2 A list of users the rule applies to. An empty list implies every user.

3

A list of verbs this rule applies to. An empty list implies every verb. This is Kubernetes verb associated with API requests (including **get**, **list**, **watch**, **create**, **update**, **patch**, **delete**,

- 4 A list of resources the rule applies to. An empty list implies every resource. Each resource is specified as a group it is assigned to (for example, an empty for Kubernetes core API, batch, build.openshift.io, etc.), and a resource list from that group.
- 5 A list of groups the rule applies to. An empty list implies every group.
- 6 A list of non-resources URLs the rule applies to.
- 7 A list of namespaces the rule applies to. An empty list implies every namespace.
- 9 Endpoint used by the web console.
- 10 Endpoint used by the CLI.

For more information on advanced audit, see the [Kubernetes documentation](#)

5.5.20. Specifying TLS ciphers for etcd

You can specify the [supported TLS ciphers](#) to use in communication between the master and etcd servers.

1. On each etcd node, upgrade etcd:

```
# yum update etcd iptables-services
```

2. Confirm that your etcd version is 3.2.22 or later:

```
# etcd --version
etcd Version: 3.2.22
```

3. On each master host, specify the ciphers to enable in the `/etc/origin/master/master-config.yaml` file:

```
servingInfo:
  ...
  minTLSVersion: VersionTLS12
  cipherSuites:
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_256_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  ...
```

4. On each master host, restart the master service:

```
# master-restart api
# master-restart controllers
```

5. Confirm that the cipher is applied. For example, for TLSv1.2 cipher **ECDHE-RSA-AES128-GCM-SHA256**, run the following command:

```

# openssl s_client -connect etcd1.example.com:2379 1
CONNECTED(00000003)
depth=0 CN = etcd1.example.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = etcd1.example.com
verify error:num=21:unable to verify the first certificate
verify return:1
139905367488400:error:14094412:SSL routines:ssl3_read_bytes:ssl3
alert bad certificate:s3_pkt.c:1493:SSL alert number 42
139905367488400:error:140790E5:SSL routines:ssl23_write:ssl
handshake failure:s23_lib.c:177:
---
Certificate chain
 0 s:/CN=etcd1.example.com
  i:/CN=etcd-signer@1529635004
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEKjCCAnqgAwIBAgIBATANBgkqhkiG9w0BAQsFADAhMR8wHQYDVQDDBZldGNk
.....
....
eif87qttt0Sl1vS8DG1KQ01oOBlnkg==
-----END CERTIFICATE-----
subject=/CN=etcd1.example.com
issuer=/CN=etcd-signer@1529635004
---
Acceptable client certificate CA names
/CN=etcd-signer@1529635004
Client Certificate Types: RSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA384
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 1666 bytes and written 138 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1.2
    Cipher : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
    Session-ID-ctx:
    Master-Key:
1EFA00A91EE5FC5EDDCFC67C8ECD060D44FD3EB23D834EDED929E4B74536F273C0F9
299935E5504B562CD56E76ED208D
    Key-Arg : None
    Krb5 Principal: None
    PSK identity: None

```

```
PSK identity hint: None
Start Time: 1529651744
Timeout    : 300 (sec)
Verify return code: 21 (unable to verify the first certificate)
```

- 1 **etcd1.example.com** is the name of an etcd host.

5.6. NODE CONFIGURATION FILES

The following **node-config.yaml** file is a sample node configuration file that was generated with the default values as of writing.



NOTE

To modify a node in your cluster, update the [node configuration maps](#) as needed. Do not manually edit the **node-config.yaml** file.

Example 5.1. Sample Node Configuration File

```
allowDisabledDocker: false
apiVersion: v1
authConfig:
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsDomain: cluster.local
dnsIP: 0.0.0.0 1
dockerConfig:
  execHandlerName: native
imageConfig:
  format: openshift/origin-${component}:${version}
  latest: false
iptablesSyncPeriod: 5s
kind: NodeConfig
masterKubeConfig: node.kubeconfig
networkConfig:
  mtu: 1450
  networkPluginName: ""
nodeIP: ""
nodeName: node1.example.com
podManifestConfig: 2
  path: "/path/to/pod-manifest-file" 3
  fileCheckIntervalSeconds: 30 4
proxyArguments:
  proxy-mode:
    - iptables 5
servingInfo:
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  certFile: server.crt
  clientCA: node-client-ca.crt
  keyFile: server.key
```

```
namedCertificates: null
volumeDirectory: /root/openshift.local.volumes
```

- 1 Configures an IP address to be prepended to a pod's */etc/resolv.conf* by adding the address here.
- 2 Allows pods to be placed directly on certain set of nodes, or on all nodes without going through the scheduler. You can then use pods to perform the same administrative tasks and support the same services on each node.
- 3 Specifies the path for the [pod manifest file](#) or directory. If it is a directory, then it is expected to contain one or more manifest files. This is used by the Kubelet to create pods on the node.
- 4 This is the interval (in seconds) for checking the manifest file for new data. The interval must be a positive value.
- 5 The [service proxy implementation](#) to use.

The node configuration file determines the resources of a node. See the [Allocating node resources section in the Cluster Administrator guide](#) for more information.

5.6.1. Pod and Node Configuration

Table 5.20. Pod and Node Configuration Parameters

Parameter Name	Description
NodeConfig	The fully specified configuration starting an OpenShift Container Platform node.
NodeIP	Node may have multiple IPs, so this specifies the IP to use for pod traffic routing. If not specified, network parse/lookup on the nodeName is performed and the first non-loopback address is used.
NodeName	The value used to identify this particular node in the cluster. If possible, this should be your fully qualified hostname. If you are describing a set of static nodes to the master, this value must match one of the values in the list.
PodEvictionTimeout	Controls grace period for deleting pods on failed nodes. It takes valid time duration string. If empty, you get the default pod eviction timeout.
ProxyClientInfo	Specifies the client cert/key to use when proxying to pods.

5.6.2. Docker Configuration

Table 5.21. Docker Configuration Parameters

Parameter Name	Description
AllowDisabledDocker	If true, the kubelet will ignore errors from Docker. This means that a node can start on a machine that does not have docker started.
DockerConfig	Holds Docker related configuration options
ExecHandlerName	The handler to use for executing commands in Docker containers.

5.6.3. Local Storage Configuration

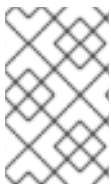
You can use the [XFS quota subsystem](#) to limit the size of **emptyDir** volumes and volumes based on an **emptyDir** volume, such as secrets and configuration maps, on each node.

To limit the size of **emptyDir** volumes in an XFS filesystem, configure local volume quota for each unique [FSGroup](#) using the **node-config-compute** configuration map in the **openshift-node** project.

```
apiVersion: kubelet.config.openshift.io/v1
kind: VolumeConfig
  localQuota: 1
  perFSGroup: 1Gi
```

- 1 Contains options for controlling local volume quota on the node.
- 2 Set this value to a resource quantity representing the desired quota per [FSGroup], per node, such as **1Gi**, **512Mi**, and so forth. Requires the **volumeDirectory** to be on an XFS filesystem mounted with the **grpquota** option. The matching security context constraint **fsGroup** type must be [set to MustRunAs](#).

If no FSGroup is specified, indicating the request matched an SCC with **RunAsAny**, the quota application is skipped.



NOTE

Do not edit the **/etc/origin/node/volume-config.yaml** file directly. The file is created from the **node-config-compute** configuration map. Use the **node-config-compute** configuration map to create or edit the parameters in the **volume-config.yaml** file.

5.6.4. Parallel Image Pulls with Docker 1.9+

If you are using Docker 1.9+, you may want to consider enabling parallel image pulling, as the default is to pull images one at a time.



NOTE

There is a potential issue with data corruption prior to Docker 1.9. However, starting with 1.9, the corruption issue is resolved and it is safe to switch to parallel pulls.

```
kubeletArguments:
  serialize-image-pulls:
    - "false" 1
```

- 1** Change to true to disable parallel pulls. (This is the default config)

5.7. PASSWORDS AND OTHER SENSITIVE DATA

For some [authentication configurations](#), an LDAP **bindPassword** or OAuth **clientSecret** value is required. Instead of specifying these values directly in the master configuration file, these values may be provided as environment variables, external files, or in encrypted files.

Environment Variable Example

```
...
bindPassword:
  env: BIND_PASSWORD_ENV_VAR_NAME
```

External File Example

```
...
bindPassword:
  file: bindPassword.txt
```

Encrypted External File Example

```
...
bindPassword:
  file: bindPassword.encrypted
  keyFile: bindPassword.key
```

To create the encrypted file and key file for the above example:

```
$ oc adm ca encrypt --genkey=bindPassword.key --out=bindPassword.encrypted
> Data to encrypt: B1ndPass0rd!
```

Run **oc adm** commands only from the first master listed in the Ansible host inventory file, by default */etc/ansible/hosts*.



WARNING

Encrypted data is only as secure as the decrypting key. Care should be taken to limit filesystem permissions and access to the key file.

5.8. CREATING NEW CONFIGURATION FILES

When defining an OpenShift Container Platform configuration from scratch, start by creating new configuration files.

For master host configuration files, use the **openshift start** command with the **--write-config** option to write the configuration files. For node hosts, use the **oc adm create-node-config** command to write the configuration files.

The following commands write the relevant launch configuration file(s), certificate files, and any other necessary files to the specified **--write-config** or **--node-dir** directory.

Generated certificate files are valid for two years, while the certification authority (CA) certificate is valid for five years. This can be altered with the **--expire-days** and **--signer-expire-days** options, but for security reasons, it is recommended to not make them greater than these values.

To create configuration files for an all-in-one server (a master and a node on the same host) in the specified directory:

```
$ openshift start --write-config=/openshift.local.config
```

To create a [master configuration file](#) and other required files in the specified directory:

```
$ openshift start master --write-config=/openshift.local.config/master
```

To create a [node configuration file](#) and other related files in the specified directory:

```
$ oc adm create-node-config \
  --node-dir=/openshift.local.config/node-<node_hostname> \
  --node=<node_hostname> \
  --hostnames=<node_hostname>,<ip_address> \
  --certificate-authority="/path/to/ca.crt" \
  --signer-cert="/path/to/ca.crt" \
  --signer-key="/path/to/ca.key" \
  --signer-serial="/path/to/ca.serial.txt" \
  --node-client-certificate-authority="/path/to/ca.crt"
```

When creating node configuration files, the **--hostnames** option accepts a comma-delimited list of every host name or IP address you want server certificates to be valid for.

5.9. LAUNCHING SERVERS USING CONFIGURATION FILES

Once you have modified the master and/or node configuration files to your specifications, you can use them when launching servers by specifying them as an argument. Keep in mind that if you specify a configuration file, none of the other command line options you pass are respected.



NOTE

To modify a node in your cluster, update the [node configuration maps](#) as needed. Do not manually edit the **node-config.yaml** file.

To launch an all-in-one server using a master configuration and a node configuration file:


```
$ openshift start --master-config=/openshift.local.config/master/master-
config.yaml --node-config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

To launch a master server using a master configuration file:

```
$ openshift start master --config=/openshift.local.config/master/master-
config.yaml
```

To launch a node server using a node configuration file:

```
$ openshift start node --config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

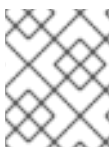
5.10. CONFIGURING LOGGING LEVELS

OpenShift Container Platform uses the **systemd-journald.service** to collect log messages for debugging, using five log message severities. The logging levels are based on Kubernetes logging conventions, as follows:

Table 5.22. Log Level Options

Option	Description
0	Errors and warnings only
2	Normal information
4	Debugging-level information
6	API-level debugging information (request / response)
8	Body-level API debugging information

You can control which INFO messages are logged by setting the `loglevel` option in the in [node configuration files](#), the `/etc/origin/master/master-config.yaml` file. Configuring the logs to collect all messages can lead to large logs that are difficult to interpret and can take up excessive space. Collecting all messages should only be used in debug situations.



NOTE

Messages with FATAL, ERROR, WARNING and some INFO severities appear in the logs regardless of the log configuration.

You can view logs for the master or the node system using the following command:

```
# journalctl -r -u <journal_name>
```

Use the `-r` option to show the newest entries first.

For example:

```
# journalctl -r -u atomic-openshift-master-controllers
# journalctl -r -u atomic-openshift-master-api
# journalctl -r -u atomic-openshift-node.service
```

To change the logging level:

1. Edit the `/etc/sysconfig/atomic-openshift-master` file for the master or `/etc/sysconfig/atomic-openshift-node` file for the nodes.
2. Enter a value from the **Log Level Options** table above in the `OPTIONS=- -loglevel=` field.
For example:

```
OPTIONS=- -loglevel=4
```

3. Restart the master or node host as appropriate. See [Restarting OpenShift Container Platform services](#).

After the restart, all new log messages will conform to the new setting. Older messages do not change.



NOTE

The default log level can be set using the standard cluster installation process. For more information, see [Cluster Variables](#).

The following examples are excerpts from a node service **journald** log at various log levels. Timestamps and system information have been removed from these examples.

Excerpt of `journalctl -u atomic-openshift-node.service` output at `loglevel=0`

```
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-system(24c32eee4307b78bb685338a>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-system(24c32eee4307b78bb685338a>
1583 kubelet_node_status.go:400] Error updating node status, will retry: failed to patch status "{\"status\":{\"setElementOrder/conditio>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-system(4d3ce7ecd6d4f423afeedd44b2f61e5c>
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-system(24c32eee4307b78bb685338a>
1583 fsHandler.go:135] du and find on following dirs took 4.659884995s: [/var/lib/docker/containers/08b2214f76605122ecd029f32c56c4302f42>
1583 fsHandler.go:135] du and find on following dirs took 4.659995453s: [/var/lib/docker/containers/285d268a5e6fe00a7cac5a70a11a011ca51c>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Readiness probe for "master-api-master.com_kube-
```

```

system(4d3ce7ecd6d4f423afeedd44b2f61e5>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5c>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 kubelet_node_status.go:400] Error updating node status, will retry:
failed to patch status "{\"status\":{\"$setElementOrder/conditio>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service

```

Excerpt of journalctl -u atomic-openshift-node.service output at loglevel=2

```

1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5c>
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-etcd-master-etcd-
master.comntdv.lab.eng.bos.redhat.com_kube-
system(e41c2a57b9d15b28cb8a240c7780597>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5c>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service

```

Excerpt of journalctl -u atomic-openshift-node.service output at loglevel=4

```

1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-etcd-master-etcd-
master.comntdv.lab.eng.bos.redhat.com_kube-
system(e41c2a57b9d15b28cb8a240c7780597>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5c>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
>>>>>> Several pages still use removed services

```

Excerpt of journalctl -u atomic-openshift-node.service output at loglevel=8

```

1583 fsHandler.go:135] du and find on following dirs took 5.173967549s: [
/var/lib/docker/containers/10aea0708b49be7e342d8337d92904256215>
1583 fsHandler.go:135] du and find on following dirs took 5.487146071s: [
/var/lib/docker/containers/5795a3717863ef48ebb962ac84e5ee407d73>
1583 fsHandler.go:135] du and find on following dirs took 5.491539774s: [
/var/lib/docker/containers/7fa2063f98e3662998a93ddd1dc0466f0a70>
1583 fsHandler.go:135] du and find on following dirs took 5.488087177s: [
/var/lib/docker/containers/3764d12be7fc2e66d31438940f89e457fdac>
1583 fsHandler.go:135] du and find on following dirs took 5.494384131s: [
/var/lib/docker/containers/c5a1f82ae892d94efa5a2a0aa4505f489c77>
1583 fsHandler.go:135] du and find on following dirs took 4.364582208s: [
/var/lib/docker/containers/d415400af55721143c7ececc4414f8635660>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Readiness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5>
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 prober.go:111] Liveness probe for "master-controlplane-master.com_kube-
system(24c32eee4307b78bb685338a>
1583 container_manager_linux.go:427] [ContainerManager]: Discovered
runtime cgroups name: /system.slice/docker.service
1583 prober.go:111] Liveness probe for "master-api-master.com_kube-
system(4d3ce7ecd6d4f423afeedd44b2f61e5c>

```

```
1583 prober.go:111] Liveness probe for "master-control-plane-master.com_kube-  
system(24c32eee4307b78bb685338a>  
1583 container_manager_linux.go:427] [ContainerManager]: Discovered  
runtime cgroups name: /system.slice/docker.service  
1583 container_manager_linux.go:427] [ContainerManager]: Discovered  
runtime cgroups name: /system.slice/docker.service
```

5.11. RESTARTING MASTER AND NODE SERVICES

To apply master or node configuration changes, you must restart the respective services.

To reload master configuration changes, restart master services running in control plane static pods using the **master-restart** command:

```
# master-restart api
```

To reload node configuration changes, restart the node service on the node host:

```
# systemctl restart atomic-openshift-node
```

CHAPTER 6. OPENSIFT ANSIBLE BROKER CONFIGURATION

6.1. OVERVIEW

When the [OpenShift Ansible broker](#) (OAB) is deployed in a cluster, its behavior is largely dictated by the broker's configuration file loaded on startup. The broker's configuration is stored as a ConfigMap object in the broker's namespace (**openshift-ansible-service-broker** by default).

Example OpenShift Ansible Broker Configuration File

```
registry: ❶
  - type: dockerhub
    name: docker
    url: https://registry.hub.docker.com
    org: <dockerhub_org>
    fail_on_error: false
  - type: rhcc
    name: rhcc
    url: https://registry.access.redhat.com
    fail_on_error: true
    white_list:
      - "^foo.*-apb$"
      - ".*-apb$"
    black_list:
      - "bar.*-apb$"
      - "^my-apb$"
  - type: local_openshift
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
dao: ❷
  etcd_host: localhost
  etcd_port: 2379
log: ❸
  logfile: /var/log/ansible-service-broker/asb.log
  stdout: true
  level: debug
  color: true
openshift: ❹
  host: ""
  ca_file: ""
  bearer_token_file: ""
  image_pull_policy: IfNotPresent
  sandbox_role: "edit"
  keep_namespace: false
  keep_namespace_on_error: true
broker: ❺
  bootstrap_on_startup: true
  dev_broker: true
  launch_apb_on_bind: false
  recovery: true
  output_request: true
```

```

ssl_cert_key: /path/to/key
ssl_cert: /path/to/cert
refresh_interval: "600s"
auth:
  - type: basic
    enabled: true
secrets: 6
  - title: Database credentials
    secret: db_creds
    apb_name: dh-rhsc1-postgresql-apb

```

1 See [Registry Configuration](#) for details.

2 See [DAO Configuration](#) for details.

3 See [Log Configuration](#) for details.

4 See [OpenShift Configuration](#) for details.

5 See [Broker Configuration](#) for details.

6 See [Secrets Configuration](#) for details.

6.2. MODIFYING THE OPENSIFT ANSIBLE BROKER CONFIGURATION

To modify the OAB's default broker configuration after it has been deployed:

1. Edit the **broker-config** ConfigMap object in the OAB's namespace as a user with **cluster-admin** privileges:

```
$ oc edit configmap broker-config -n openshift-ansible-service-broker
```

2. After saving any updates, redeploy the OAB's deployment configuration for the changes to take effect:

```
$ oc rollout latest dc/asb -n openshift-ansible-service-broker
```

6.3. REGISTRY CONFIGURATION

The **registry** section allows you to define the registries that the broker should look at for APBs.

Table 6.1. registry Section Configuration Options

Field	Description	Required
name	The name of the registry. Used by the broker to identify APBs from this registry.	Y
user	The user name for authenticating to the registry. Not used when auth_type is set to secret or file .	N

Field	Description	Required
pass	The password for authenticating to the registry. Not used when auth_type is set to secret or file .	N
auth_type	How the broker should read the registry credentials if they are not defined in the broker configuration via user and pass . Can be secret (uses a secret in the broker namespace) or file (uses a mounted file).	N
auth_name	Name of the secret or file storing the registry credentials that should be read. Used when auth_type is set to secret .	N, only required when auth_type is set to secret or file .
org	The namespace or organization that the image is contained in.	N
type	The type of registry. Available adapters are mock , rhcc , openshift , dockerhub , and local_openshift .	Y
namespaces	The list of namespaces to configure the local_openshift registry type with. By default, a user should use openshift .	N
url	The URL that is used to retrieve image information. Used extensively for RHCC while the dockerhub type uses hard-coded URLs.	N
fail_on_error	Should this registry fail, the bootstrap request if it fails. Will stop the execution of other registries loading.	N
white_list	The list of regular expressions used to define which image names should be allowed through. Must have a white list to allow APBs to be added to the catalog. The most permissive regular expression that you can use is . *-apb\$ if you would want to retrieve all APBs in a registry. See APB Filtering for more details.	N
black_list	The list of regular expressions used to define which images names should never be allowed through. See APB Filtering for more details.	N
images	The list of images to be used with an OpenShift Container Registry.	N

6.3.1. Production or Development

A *production* broker configuration is designed to be pointed at a trusted container distribution registry, such as the Red Hat Container Catalog (RHCC):

```
registry:
```



```
- name: rhcc
  type: rhcc
  url: https://registry.access.redhat.com
  tag: v3.10
  white_list:
    - ".*-apb$"
- type: local_openshift
  name: localregistry
  namespaces:
    - openshift
  white_list: []
```

However, a *development* broker configuration is primarily used by developers working on the broker. To enable developer settings, set the registry name to **dev** and the **dev_broker** field in the **broker** section to **true**:

```
registry:
  name: dev

broker:
  dev_broker: true
```

6.3.2. Storing Registry Credentials

The broker configuration determines how the broker should read any registry credentials. They can be read from the **user** and **pass** values in the **registry** section, for example:

```
registry:
- name: isv
  type: openshift
  url: https://registry.connect.redhat.com
  user: <user>
  pass: <password>
```

If you want to ensure these credentials are not publicly accessible, the **auth_type** field in the **registry** section can be set to the **secret** or **file** type. The **secret** type configures a registry to use a secret from the broker's namespace, while the **file** type configures a registry to use a secret that has been mounted as a volume.

To use the **secret** or **file** type:

1. The associated secret should have the values **username** and **password** defined. When using a secret, you must ensure that the **openshift-ansible-service-broker** namespace exists, as this is where the secret will be read from.

For example, create a **reg-creds.yaml** file:

```
$ cat reg-creds.yaml
---
username: <user_name>
password: <password>
```

2. Create a secret from this file in the **openshift-ansible-service-broker** namespace:

```
$ oc create secret generic \
  registry-credentials-secret \
  --from-file reg-creds.yaml \
  -n openshift-ansible-service-broker
```

3. Choose whether you want to use the **secret** or **file** type:

- To use the **secret** type:

- a. In the broker configuration, set **auth_type** to **secret** and **auth_name** to the name of the secret:

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: secret
    auth_name: registry-credentials-secret
```

- b. Set the namespace where the secret is located:

```
openshift:
  namespace: openshift-ansible-service-broker
```

- To use the **file** type:

- a. Edit the **asb** deployment configuration to mount your file into **/tmp/registry-credentials/reg-creds.yaml**:

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

In the **containers.volumeMounts** section, add:

```
volumeMounts:
  - mountPath: /tmp/registry-credentials
    name: reg-auth
```

In the **volumes** section, add:

```
volumes:
  - name: reg-auth
    secret:
      defaultMode: 420
      secretName: registry-credentials-secret
```

- b. In the broker configuration, set **auth_type** to **file** and **auth_type** to the location of the file:

```
registry:
  - name: isv
    type: openshift
```

```
url: https://registry.connect.redhat.com
auth_type: file
auth_name: /tmp/registry-credentials/reg-creds.yaml
```

6.3.3. Mock Registry

A mock registry is useful for reading local APB specs. Instead of going out to a registry to search for image specs, this uses a list of local specs. Set the name of the registry to **mock** to use the mock registry.

```
registry:
  - name: mock
    type: mock
```

6.3.4. Dockerhub Registry

The **dockerhub** type allows you to load APBs from a specific organization in the DockerHub. For example, the [ansibleplaybookbundle](#) organization.

```
registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
```

6.3.5. APB Filtering

APBs can be filtered out by their image name using a combination of the **white_list** or **black_list** parameters, set on a registry basis inside the broker's configuration.

Both are optional lists of regular expressions that will be run over the total set of discovered APBs for a given registry to determine matches.

Table 6.2. APB Filter Behavior

Present	Allowed	Blocked
Only whitelist	Matches a regex in list.	Any APB that does not match.
Only blacklist	All APBs that do not match.	APBs that match a regex in list.
Both present	Matches regex in whitelist but not in blacklist.	APBs that match a regex in blacklist.
None	No APBs from the registry.	All APBs from that registry.

For example:

Whitelist Only

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
```

Anything matching on **foo.*-apb\$** and only **my-apb** will be allowed through in this case. All other APBs will be rejected.

Blacklist Only

```
black_list:
  - "bar.*-apb$"
  - "^foobar-apb$"
```

Anything matching on **bar.*-apb\$** and only **foobar-apb** will be blocked in this case. All other APBs will be allowed through.

Whitelist and Blacklist

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
black_list:
  - "^foo-rootkit-apb$"
```

Here, **foo-rootkit-apb** is specifically blocked by the blacklist despite its match in the whitelist because the whitelist match is overridden.

Otherwise, only those matching on **foo.*-apb\$** and **my-apb** will be allowed through.

Example Broker Configuration registry Section:

```
registry:
  - type: dockerhub
    name: dockerhub
    url: https://registry.hub.docker.com
    user: <user>
    pass: <password>
    org: <org>
    white_list:
      - "foo.*-apb$"
      - "^my-apb$"
    black_list:
      - "bar.*-apb$"
      - "^foobar-apb$"
```

6.3.6. Local OpenShift Container Registry

Using the **local_openshift** type will allow you to load APBs from the OpenShift Container Registry that is internal to the OpenShift Container Platform cluster. You can configure the namespaces in which you want to look for published APBs.

```
registry:
  - type: local_openshift
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
```

6.3.7. Red Hat Container Catalog Registry

Using the **rhcc** type will allow you to load APBs that are published to the [Red Hat Container Catalog](#) (RHCC) registry.

```
registry:
  - name: rhcc
    type: rhcc
    url: https://registry.access.redhat.com
    white_list:
      - ".*-apb$"
```

6.3.8. Red Hat Connect Partner Registry

Third-party images in the Red Hat Container Catalog are served from the Red Hat Connect Partner Registry at <https://registry.connect.redhat.com>. The **partner_rhcc** type allows the broker to be bootstrapped from the Partner Registry to retrieve a list of APBs and load their specs. The Partner Registry requires authentication for pulling images with a valid Red Hat Customer Portal user name and password.

```
registry:
  - name: partner_reg
    type: partner_rhcc
    url: https://registry.connect.redhat.com
    user: <registry_user>
    pass: <registry_password>
    white_list:
      - ".*-apb$"
```

Because the Partner Registry requires authentication, the following manual step is also required to configure the broker to use the Partner Registry URL:

1. Run the following command on all nodes of a OpenShift Container Platform cluster:

```
# docker --config=/var/lib/origin/.docker \
  login -u <registry_user> -p <registry_password> \
  registry.connect.redhat.com
```

6.3.9. Multiple Registries

You can use more than one registry to separate APBs into logical organizations and be able to manage them from the same broker. The registries must have a unique, non-empty name. If there is no unique name, the service broker will fail to start with an error message alerting you to the problem.

```
registry:
```

```

- name: dockerhub
  type: dockerhub
  org: ansibleplaybookbundle
  user: <user>
  pass: <password>
  white_list:
    - ".*-apb$"
- name: rhcc
  type: rhcc
  url: <rhcc_url>
  white_list:
    - ".*-apb$"

```

6.4. BROKER AUTHENTICATION

The broker supports authentication, meaning when connecting to the broker, the caller must supply the Basic Auth or Bearer Auth credentials for each request. Using **curl**, it is as simple as supplying:

```
-u <user_name>:<password>
```

or

```
-h "Authorization: bearer <token>
```

to the command. The service catalog must be configured with a secret containing the user name and password combinations or the bearer token.

6.4.1. Basic Auth

To enable Basic Auth usage, set the following in the broker configuration:

```

broker:
  ...
  auth:
    - type: basic 1
      enabled: true 2

```

1 The **type** field specifies the type of authentication to use.

2 The **enabled** field allows you to disable a particular authentication type. This keeps you from having to delete the entire section of **auth** just to disable it.

6.4.1.1. Deployment Template and Secrets

Typically the broker is configured using a [ConfigMap](#) in a deployment template. You supply the authentication configuration the same way as in the file configuration.

The following is an example of the [deployment template](#):

```

auth:
  - type: basic
    enabled: ${ENABLE_BASIC_AUTH}

```

Another part to Basic Auth is the user name and password used to authenticate against the broker. While the Basic Auth implementation can be backed by different back-end services, the currently supported one is backed by a *secret*. The secret must be injected into the pod via a volume mount at the `/var/run/asb_auth` location. This is from where the broker will read the user name and password.

In the [deployment template](#), a secret must be specified. For example:

```
- apiVersion: v1
  kind: Secret
  metadata:
    name: asb-auth-secret
    namespace: openshift-ansible-service-broker
  data:
    username: ${BROKER_USER}
    password: ${BROKER_PASS}
```

The secret must contain a user name and password. The values must be base64 encoded. The easiest way to generate the values for those entries is to use the **echo** and **base64** commands:

```
$ echo -n admin | base64 ❶
YWRtaW4=
```

❶ The **-n** option is very important.

This secret must now be injected to the pod via a volume mount. This is configured in the deployment template as well:

```
spec:
  serviceAccount: asb
  containers:
  - image: ${BROKER_IMAGE}
    name: asb
    imagePullPolicy: IfNotPresent
    volumeMounts:
      ...
      - name: asb-auth-volume
        mountPath: /var/run/asb-auth
```

Then, in the **volumes** section, mount the secret:

```
volumes:
  ...
  - name: asb-auth-volume
    secret:
      secretName: asb-auth-secret
```

The above will have created a volume mount located at `/var/run/asb-auth`. This volume will have two files: a user name and password written by the **asb-auth-secret** secret.

6.4.1.2. Configuring Service Catalog and Broker Communication

Now that the broker is configured to use Basic Auth, you must tell the service catalog how to communicate with the broker. This is accomplished by the **authInfo** section of the broker resource.

The following is an example of creating a **broker** resource in the service catalog. The **spec** tells the service catalog what URL the broker is listening at. The **authInfo** tells it what secret to read to get the authentication information.

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: Broker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basicAuthSecret:
      namespace: openshift-ansible-service-broker
      name: asb-auth-secret
```

Starting with v0.0.17 of the service catalog, the broker resource configuration changes:

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basic:
      secretRef:
        namespace: openshift-ansible-service-broker
        name: asb-auth-secret
```

6.4.2. Bearer Auth

By default, if no authentication is specified the broker will use bearer token authentication (Bearer Auth). Bearer Auth uses delegated authentication from the [Kubernetes apiserver](#) library.



NOTE

Bearer Auth is only available starting in OpenShift Container Platform 3.7.

The configuration grants access, through [Kubernetes RBAC](#) roles and role bindings, to the URL prefix. The broker has added a configuration option **cluster_url** to specify the **url_prefix**. This value defaults to **openshift-ansible-service-broker**.

Example Cluster Role

```
- apiVersion: authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: access-asb-role
  rules:
    - nonResourceURLs: ["/ansible-service-broker", "/ansible-service-broker/*"]
```



```
verbs: ["get", "post", "put", "patch", "delete"]
```

6.4.2.1. Deployment Template and Secrets

The following is an example of creating a secret that the service catalog can use. This example assumes that the role, **access-asb-role**, has been created already. From the [deployment template](#):

```
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: ansible-service-broker-client
    namespace: openshift-ansible-service-broker

- apiVersion: authorization.openshift.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: ansible-service-broker-client
  subjects:
    - kind: ServiceAccount
      name: ansible-service-broker-client
      namespace: openshift-ansible-service-broker
  roleRef:
    kind: ClusterRole
    name: access-asb-role

- apiVersion: v1
  kind: Secret
  metadata:
    name: ansible-service-broker-client
    annotations:
      kubernetes.io/service-account.name: ansible-service-broker-client
  type: kubernetes.io/service-account-token
```

The above example creates a service account, granting access to **access-asb-role** and [creating a secret](#) for that service accounts token.

6.4.2.2. Configuring Service Catalog and Broker Communication

Now that the broker is configured to use Bearer Auth tokens, you must tell the service catalog how to communicate with the broker. This is accomplished by the **authInfo** section of the **broker** resource.

The following is an example of creating a **broker** resource in the service catalog. The **spec** tells the service catalog what URL the broker is listening at. The **authInfo** tells it what secret to read to get the authentication information.

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb.openshift-ansible-service-
broker.svc:1338${BROKER_URL_PREFIX}/
  authInfo:
    bearer:
```

```
secretRef:
  kind: Secret
  namespace: openshift-ansible-service-broker
  name: ansible-servicebroker-client
```

6.5. DAO CONFIGURATION

Field	Description	Required
etcd_host	The URL of the etcd host.	Y
etcd_port	The port to use when communicating with etcd_host .	Y

6.6. LOG CONFIGURATION

Field	Description	Required
logfile	Where to write the broker's logs.	Y
stdout	Write logs to stdout.	Y
level	Level of the log output.	Y
color	Color the logs.	Y

6.7. OPENSIFT CONFIGURATION

Field	Description	Required
host	OpenShift Container Platform host.	N
ca_file	Location of the certificate authority file.	N
bearer_token_file	Location of bearer token to be used.	N
image_pull_policy	When to pull the image.	Y
namespace	The namespace that the broker has been deployed to. Important for things like passing parameter values via secret.	Y
sandbox_role	Role to give to an APB sandbox environment.	Y

Field	Description	Required
keep_namespace	Always keep namespace after an APB execution.	N
keep_namespace_on_error	Keep namespace after an APB execution has an error.	N

6.8. BROKER CONFIGURATION

The **broker** section tells the broker what functionality should be enabled and disabled. It will also tell the broker where to find files on disk that will enable the full functionality.

Field	Description	Default Value	Required
dev_broker	Allow development routes to be accessible.	false	N
launch_apb_on_bind	Allow bind to be a no-op.	false	N
bootstrap_on_startup	Allow the broker attempt to bootstrap itself on start up. Will retrieve the APBs from configured registries.	false	N
recovery	Allow the broker to attempt to recover itself by dealing with pending jobs noted in etcd.	false	N
output_request	Allow the broker to output the requests to the log file as they come in for easier debugging.	false	N
ssl_cert_key	Tells the broker where to find the TLS key file. If not set, the API server will attempt to create one.	""	N
ssl_cert	Tells the broker where to find the TLS .crt file. If not set, the API server will attempt to create one.	""	N
refresh_interval	The interval to query registries for new image specs.	"600s"	N
auto_escalate	Allows the broker to escalate the permissions of a user while running the APB.	false	N
cluster_url	Sets the prefix for the URL that the broker is expecting.	openshift-ansible-service-broker	N

**NOTE**

Async bind and unbind is an experimental feature and is not supported or enabled by default. With the absence of async bind, setting **launch_apb_on_bind** to **true** can cause the bind action to timeout and will span a retry. The broker will handle this with "409 Conflicts" because it is the same bind request with different parameters.

6.9. SECRETS CONFIGURATION

The **secrets** section creates associations between secrets in the broker's namespace and APBs the broker runs. The broker uses these rules to mount secrets into running APBs, allowing the user to use secrets to pass parameters without exposing them to the catalog or users.

The section is a list where each entry has the following structure:

Field	Description	Required
title	The title of the rule. This is just for display and output purposes.	Y
apb_name	The name of the APB to associate with the specified secret. This is the fully qualified name (<registry_name>-<image_name>).	Y
secret	The name of the secret to pull parameters from.	Y

You can download and use the [create_broker_secret.py](#) file to create and format this configuration section.

```
secrets:
- title: Database credentials
  secret: db_creds
  apb_name: dh-rhsc1-postgresql-apb
```

6.10. RUNNING BEHIND A PROXY

When running the OAB inside of a proxied OpenShift Container Platform cluster, it is important to understand its core concepts and consider them within the context of a proxy used for external network access.

As an overview, the broker itself runs as a pod within the cluster. It has a requirement for external network access depending on how its registries have been configured.

6.10.1. Registry Adapter Whitelists

The broker's configured registry adapters must be able to communicate with their external registries in order to bootstrap successfully and load remote APB manifests. These requests can be made via the proxy, however, the proxy must ensure that the required remote hosts are accessible.

Example required whitelisted hosts:

Registry Adapter Type	Whitelisted Hosts
rhcc	registry.access.redhat.com, access.redhat.com
dockerhub	docker.io

6.10.2. Configuring the Broker Behind a Proxy Using Ansible

If during initial installation you configure your OpenShift Container Platform cluster to run behind a proxy (see [Configuring Global Proxy Options](#)), when the OAB is deployed it will:

- inherit those cluster-wide proxy settings automatically and
- generate the required **NO_PROXY** list, including the **cidr** fields and **serviceNetworkCIDR**,

and no further configuration is needed.

6.10.3. Configuring the Broker Behind a Proxy Manually

If your cluster's global proxy options were not configured during initial installation or prior to the broker being deployed, or if you have modified the global proxy settings, you must manually configure the broker for external access via proxy:

1. Before attempting to run the OAB behind a proxy, review [Working with HTTP Proxies](#) and ensure your cluster is configured accordingly to run behind a proxy.
In particular, the cluster must be configured to *not* proxy internal cluster requests. This is typically configured with a **NO_PROXY** setting of:

```
.cluster.local, .svc, <serviceNetworkCIDR_value>, <master_IP>,  
<master_domain>, .default
```

in addition to any other desired **NO_PROXY** settings. See [Configuring NO_PROXY](#) for more details.



NOTE

Brokers deploying unversioned, or v1 APBs *must* also add **172.30.0.1** to their **NO_PROXY** list. APBs prior to v2 extracted their credentials from running APB pods via an **exec** HTTP request, rather than a secret exchange. Unless you are running a broker with experimental proxy support in a cluster prior to OpenShift Container Platform 3.9, you probably do not have to worry about this.

2. Edit the broker's **DeploymentConfig** as a user with **cluster-admin** privileges:

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

3. Set the following environment variables:

- **HTTP_PROXY**

- **HTTPS_PROXY**
- **NO_PROXY**

**NOTE**

See [Setting Proxy Environment Variables in Pods](#) for more information.

4. After saving any updates, redeploy the OAB's deployment configuration for the changes to take effect:

```
$ oc rollout latest dc/asb -n openshift-ansible-service-broker
```

6.10.4. Setting Proxy Environment Variables in Pods

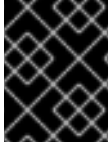
It is common that APB pods themselves may require external access via proxy as well. If the broker recognizes it has a proxy configuration, it will transparently apply these environment variables to the APB pods that it spawns. As long as the modules used within the APB respect proxy configuration via environment variable, the APB will also use these settings to perform its work.

Finally, it is possible the services spawned by the APB may also require external network access via proxy. The APB *must* be authored to set these environment variables explicitly if recognizes them in its own execution environment, or the cluster operator must manually modify the required services to inject them into their environments.

CHAPTER 7. ADDING HOSTS TO AN EXISTING CLUSTER

7.1. ADDING HOSTS

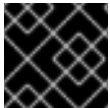
You can add new hosts to your cluster by running the ***scaleup.yml*** playbook. This playbook queries the master, generates and distributes new certificates for the new hosts, and then runs the configuration playbooks on only the new hosts. Before running the ***scaleup.yml*** playbook, complete all prerequisite [host preparation](#) steps.



IMPORTANT

The ***scaleup.yml*** playbook configures only the new host. It does not update ***NO_PROXY*** in master services, and it does not restart master services.

You must have an existing inventory file, for example ***/etc/ansible/hosts***, that is representative of your current cluster configuration in order to run the ***scaleup.yml*** playbook. If you previously used the ***atomic-openshift-installer*** command to run your installation, you can check ***~/.config/openshift/hosts*** for the last inventory file that the installer generated and use that file as your inventory file. You can modify this file as required. You must then specify the file location with ***-i*** when you run the ***ansible-playbook***.



IMPORTANT

See the [cluster limits](#) section for the recommended maximum number of nodes.

Procedure

1. Ensure you have the latest playbooks by updating the ***atomic-openshift-utils*** package:

```
# yum update atomic-openshift-utils
```

2. Edit your ***/etc/ansible/hosts*** file and add ***new_<host_type>*** to the ***[OSEv3:children]*** section: For example, to add a new node host, add ***new_nodes***:

```
[OSEv3:children]
masters
nodes
new_nodes
```

To add new master hosts, add ***new_masters***.

3. Create a ***[new_<host_type>]*** section to specify host information for the new hosts. Format this section like an existing section, as shown in the following example of adding a new node:

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'
```

```
[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

See [Configuring Host Variables](#) for more options.

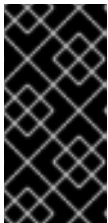
When adding new masters, add hosts to both the **[new_masters]** section and the **[new_nodes]** section to ensure that the new master host is part of the OpenShift SDN.

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
master3.example.com
```



IMPORTANT

If you label a master host with the **node-role.kubernetes.io/infra=true** label and have no other dedicated infrastructure nodes, you must also explicitly mark the host as schedulable by adding **openshift_schedulable=true** to the entry. Otherwise, the registry and router pods cannot be placed anywhere.

4. Run the **scaleup.yml** playbook. If your inventory file is located somewhere other than the default of **/etc/ansible/hosts**, specify the location with the **-i** option.

- For additional nodes:

```
# ansible-playbook [-i /path/to/file] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    node/scaleup.yml
```

- For additional masters:

```
# ansible-playbook [-i /path/to/file] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    master/scaleup.yml
```

5. After the playbook runs, [verify the installation](#).
6. Move any hosts that you defined in the **[new_<host_type>]** section to their appropriate section. By moving these hosts, subsequent playbook runs that use this inventory file treat the nodes correctly. You can keep the empty **[new_<host_type>]** section. For example, when adding new nodes:


```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-
infra'
infra-node2.example.com openshift_node_group_name='node-config-
infra'

[new_nodes]
```

7.2. ADDING ETCD HOSTS TO EXISTING CLUSTER

You can add new etcd hosts to your cluster by running the *etcd scaleup* playbook. This playbook queries the master, generates and distributes new certificates for the new hosts, and then runs the configuration playbooks on the new hosts only. Before running the etcd ***scaleup.yml*** playbook, complete all prerequisite [host preparation](#) steps.

To add an etcd host to an existing cluster:

1. Ensure you have the latest playbooks by updating the **atomic-openshift-utils** package:

```
$ yum update atomic-openshift-utils
```

2. Edit your **/etc/ansible/hosts** file, add **new_<host_type>** to the **[OSEv3:children]** group and add hosts under the **new_<host_type>** group:

For example, to add a new etcd, add **new_etcd**:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

[etcd]
etcd1.example.com
etcd2.example.com

[new_etcd]
etcd3.example.com
```

3. Run the etcd ***scaleup.yml*** playbook. If your inventory file is located somewhere other than the default of **/etc/ansible/hosts**, specify the location with the **-i** option.

```
$ ansible-playbook [-i /path/to/file] \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
  etcd/scaleup.yml
```

4. Set the node label to **logging-infra-fluentd: "true"**.

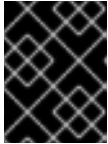
```
# oc label node/new-node.example.com logging-infra-fluentd: "true"
```

5. After the playbook completes successfully, [verify the installation](#).

7.3. REPLACING EXISTING MASTERS WITH ETCD COLOCATED

Follow these steps when you are migrating your machines to a different data center and the network and IPs assigned to it will change.

1. Back up the primary [etcd](#) and [master](#) nodes.



IMPORTANT

Ensure that you back up the `/etc/etcd/` directory, as noted in the [etcd backup](#) instructions.

2. Provision as many new machines as there are masters to replace.
3. Add or expand the cluster. for example, if you want to add 3 masters with etcd colocated, scale up 3 master nodes.



IMPORTANT

In the initial release of OpenShift Container Platform version 3.11, the ***scaleup.yml*** playbook does not scale up etcd. This will be fixed in a future release on [BZ#1628201](#).

- a. Add a [master](#). In step 3 of that process, add the host of the new data center in `[new_masters]` and `[new_nodes]` and run the master ***scaleup.yml*** playbook.
- b. Put the same host in the [etcd](#) section and run the etcd ***scaleup.yml*** playbook.
- c. Verify that the host was added:

```
# oc get nodes
```

- d. Verify that the master host IP was added:

```
# oc get ep kubernetes
```

- e. Verify that etcd was added. The value of **ETCDCTL_API** depends on the version being used:

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS
member list
```

- f. Copy `/etc/origin/master/ca.serial.txt` from the `/etc/origin/master` directory to the new master host that is listed first in your inventory file. By default, this is `/etc/ansible/hosts`.
 1. Remove the etcd hosts.
- g. Copy the `/etc/etcd/ca` directory to the new etcd host that is listed first in your inventory file. By default, this is `/etc/ansible/hosts`.

- h. Remove the old etcd clients from the **master-config.yaml** file:

```
# grep etcdClientInfo -A 11 /etc/origin/master/master-config.yaml
```

- i. Restart the masters:

```
# master-restart api
# master-restart controllers
```

- j. Remove the old etcd members from the cluster. The value of **ETCDCTL_API** depends on the version being used:

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS
member list
```

- k. Take the IDs from the output of the command above and remove the old members using the IDs:

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URL
member remove 1609b5a3a078c227
```

- l. Stop the etcd services on the old etcd hosts by removing the etcd pod definition:

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

1. Shut down old master API and controller services by moving definition files out of the static pods dir /etc/origin/node/pods:

```
# mkdir -p /etc/origin/node/pods/disabled
# mv /etc/origin/node/pods/controller.yaml
/etc/origin/node/pods/disabled/:
```

2. Remove the master nodes from the HA proxy configuration, which was installed as a load balancer by default during the native installation process.

3. Decommission the machine.

- m. Stop the node service on the master to be removed by removing the pod definition and rebooting the host:

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
# reboot
```

- n. Delete the node resource:

```
# oc delete node
```

7.4. MIGRATING THE NODES

You can migrate nodes individually or in groups (of 2, 5, 10, and so on), depending on what you are comfortable with and how the services on the node are run and scaled.

1. For the migration node or nodes, provision new VMs for the node's use in the new data center.
2. To add the new node, [scale up the infrastructure](#). Ensure the labels for the new node are set properly and that your new API servers are added to your load balancer and successfully serving traffic.
3. Evaluate and scale down.
 - a. Mark the current node (in the old data center) [unscheduled](#).
 - b. [Evacuate the node](#), so that pods on it are scheduled to other nodes.
 - c. Verify that the evacuated services are running on the new nodes.
4. Remove the node.
 - a. Verify that the node is empty and does not have running processes.
 - b. Stop the service or delete the node.

CHAPTER 8. ADDING THE DEFAULT IMAGE STREAMS AND TEMPLATES

8.1. OVERVIEW

If you installed OpenShift Container Platform on servers with x86_64 architecture, your cluster includes useful sets of Red Hat-provided [image streams](#) and [templates](#) to make it easy for developers to create new applications. By default, the [cluster installation](#) process automatically create these sets in the **openshift** project, which is a default global project to which all users have view access.

If you installed OpenShift Container Platform on servers with IBM POWER architecture, you can add [image streams](#) and [templates](#) to your cluster.

8.2. OFFERINGS BY SUBSCRIPTION TYPE

Depending on the active subscriptions on your Red Hat account, the following sets of image streams and templates are provided and supported by Red Hat. Contact your Red Hat sales representative for further subscription details.

8.2.1. OpenShift Container Platform Subscription

The core set of image streams and templates are provided and supported with an active *OpenShift Container Platform subscription*. This includes the following technologies:

Type	Technology
Languages & Frameworks	<ul style="list-style-type: none"> • .NET Core • Node.js • Perl • PHP • Python • Ruby
Databases	<ul style="list-style-type: none"> • MariaDB • MongoDB • MySQL • PostgreSQL
Middleware Services	<ul style="list-style-type: none"> • Red Hat JBoss Web Server (Tomcat) • Red Hat Single Sign-on

Type	Technology
Other Services	<ul style="list-style-type: none"> • Jenkins • Jenkins Slaves

8.2.2. xPaaS Middleware Add-on Subscriptions

Support for xPaaS middleware images are provided by *xPaaS Middleware add-on subscriptions*, which are separate subscriptions for each xPaaS product. If the relevant subscription is active on your account, image streams and templates are provided and supported for the following technologies:

Type	Technology
Middleware Services	<ul style="list-style-type: none"> • Red Hat JBoss A-MQ • Red Hat JBoss BPM Suite Intelligent Process Server • Red Hat JBoss BRMS Decision Server • Red Hat JBoss Data Grid • Red Hat JBoss EAP • Red Hat JBoss Fuse Integration Services • Red Hat JBoss Data Virtualization

8.3. BEFORE YOU BEGIN

Before you consider performing the tasks in this topic, confirm if these image streams and templates are already registered in your OpenShift Container Platform cluster by doing one of the following:

- Log into the web console and click **Add to Project**.
- List them for the **openshift** project using the CLI:

```
$ oc get is -n openshift
$ oc get templates -n openshift
```

If the default image streams and templates are ever removed or changed, you can follow this topic to create the default objects yourself. Otherwise, the following instructions are not necessary.

8.4. PREREQUISITES

Before you can create the default image streams and templates:

- The [integrated Docker registry](#) service must be deployed in your OpenShift Container Platform installation.

- You must be able to run the **oc create** command with [cluster-admin privileges](#), because they operate on the default **openshiftproject**.
- You must have installed the **atomic-openshift-utils** RPM package. See [Software Prerequisites](#) for instructions.
- Define shell variables for the directories containing image streams and templates. This significantly shortens the commands in the following sections. To do this:
 - For cloud installations and on-premise installations on x86_64 servers:

```
$ IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/image-streams"; \
  XPAASSTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/xpaas-streams"; \
  XPAASTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/xpaas-templates"; \
  DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/db-templates"; \
  QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/quickstart-
templates"
```

- For on-premise installations on IBM POWER8 or IBM POWER9 servers:

```
IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/image-streams"; \
  DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/db-templates"; \
  QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/quickstart-
templates"
```

8.5. CREATING IMAGE STREAMS FOR OPENSIFT CONTAINER PLATFORM IMAGES

If your node hosts are subscribed using Red Hat Subscription Manager and you want to use the core set of image streams that used Red Hat Enterprise Linux (RHEL) 7 based images:

```
$ oc create -f $IMAGESTREAMDIR/image-streams-rhel7.json -n openshift
```

Alternatively, to create the core set of image streams that use the CentOS 7 based images:

```
$ oc create -f $IMAGESTREAMDIR/image-streams-centos7.json -n openshift
```

Creating both the CentOS and RHEL sets of image streams is not possible, because they use the same names. To have both sets of image streams available to users, either create one set in a different project, or edit one of the files and modify the image stream names to make them unique.

8.6. CREATING IMAGE STREAMS FOR XPAAS MIDDLEWARE IMAGES

The xPaaS Middleware image streams provide images for **JBoss EAP**, **JBoss JWS**, **JBoss A-MQ**, **JBoss Fuse Integration Services**, **Decision Server**, **JBoss Data Virtualization** and **JBoss Data Grid**. They can be used to build applications for those platforms using the provided templates.

To create the xPaaS Middleware set of image streams:

```
$ oc create -f $XPAASSTREAMDIR/jboss-image-streams.json -n openshift
```



NOTE

Access to the images referenced by these image streams requires the relevant xPaaS Middleware subscriptions.

8.7. CREATING DATABASE SERVICE TEMPLATES

The database service templates make it easy to run a database image which can be utilized by other components. For each database ([MongoDB](#), [MySQL](#), and [PostgreSQL](#)), two templates are defined.

One template uses ephemeral storage in the container which means data stored will be lost if the container is restarted, for example if the pod moves. This template should be used for demonstration purposes only.

The other template defines a persistent volume for storage, however it requires your OpenShift Container Platform installation to have [persistent volumes](#) configured.

To create the core set of database templates:

```
$ oc create -f $DBTEMPLATES -n openshift
```

After creating the templates, users are able to easily instantiate the various templates, giving them quick access to a database deployment.

8.8. CREATING INSTANT APP AND QUICKSTART TEMPLATES

The Instant App and Quickstart templates define a full set of objects for a running application. These include:

- [Build configurations](#) to build the application from source located in a GitHub public repository
- [Deployment configurations](#) to deploy the application image after it is built.
- [Services](#) to provide load balancing for the application [pods](#).
- [Routes](#) to provide external access to the application.

Some of the templates also define a database deployment and service so the application can perform database operations.



NOTE

The templates which define a database use ephemeral storage for the database content. These templates should be used for demonstration purposes only as all database data will be lost if the database pod restarts for any reason.

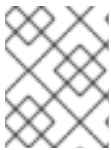
Using these templates, users are able to easily instantiate full applications using the various language images provided with OpenShift Container Platform. They can also customize the template parameters during instantiation so that it builds source from their own repository rather than the sample repository, so this provides a simple starting point for building new applications.

To create the core Instant App and Quickstart templates:

```
$ oc create -f $QSTEMPLATES -n openshift
```

There is also a set of templates for creating applications using various xPaaS Middleware products (**JBoss EAP**, **JBoss JWS**, **JBoss A-MQ**, **JBoss Fuse Integration Services**, **Decision Server**, and **JBoss Data Grid**), which can be registered by running:

```
$ oc create -f $XPAASTEMPLATES -n openshift
```



NOTE

The xPaaS Middleware templates require the [xPaaS Middleware image streams](#), which in turn require the relevant xPaaS Middleware subscriptions.



NOTE

The templates which define a database use ephemeral storage for the database content. These templates should be used for demonstration purposes only as all database data will be lost if the database pod restarts for any reason.

8.9. WHAT'S NEXT?

With these artifacts created, developers can now [log into the web console](#) and follow the flow for [creating from a template](#). Any of the database or application templates can be selected to create a running database service or application in the current project. Note that some of the application templates define their own database services as well.

The example applications are all built out of GitHub repositories which are referenced in the templates by default, as seen in the **SOURCE_REPOSITORY_URL** parameter value. Those repositories can be forked, and the fork can be provided as the **SOURCE_REPOSITORY_URL** parameter value when creating from the templates. This allows developers to experiment with creating their own applications.

You can direct your developers to the [Using the Instant App and Quickstart Templates](#) section in the Developer Guide for these instructions.

CHAPTER 9. CONFIGURING CUSTOM CERTIFICATES

9.1. OVERVIEW

Administrators can configure custom serving certificates for the public host names of the OpenShift Container Platform API and [web console](#). This can be done during a [cluster installation](#) or configured after installation.

9.2. CONFIGURING CUSTOM CERTIFICATES DURING INSTALLATION

During cluster installations, custom certificates can be configured using the `openshift_master_named_certificates` and `openshift_master_overwrite_named_certificates` parameters, which are configurable in the inventory file. More details are available about [configuring custom certificates with Ansible](#).

Custom Certificate Configuration Parameters

```
openshift_master_overwrite_named_certificates=true ❶
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-
file", "keyfile": "/path/on/host/to/key-file", "names": ["public-master-
host.com"], "cafile": "/path/on/host/to/ca-file"}] ❷
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-
crt-file", "keyfile": "/path/on/host/to/app-key-file", "cafile":
"/path/on/host/to/app-ca-file"} ❸
```

- ❶ If you provide a value for the `openshift_master_named_certificates` parameter, set this parameter to **true**.
- ❷ Provisions a [master API certificate](#).
- ❸ Provisions a [wildcard API certificate](#).

Example parameters for a master API certificate:

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"names":
["master.148.251.233.173.nip.io"], "certfile": "/home/cloud-user/master-
bundle.cert.pem", "keyfile": "/home/cloud-
user/master.148.251.233.173.nip.io.key.pem" ]
```

Example parameters for a wildcard API certificate:

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-
chain.cert.pem"}
```

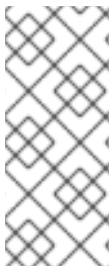
9.3. CONFIGURING CUSTOM CERTIFICATES FOR THE WEB CONSOLE OR CLI

You can specify custom certificates for the web console and for the CLI through the **servingInfo** section of the [master configuration file](#):

- The **servingInfo.namedCertificates** section serves up custom certificates for the web console.
- The **servingInfo** section serves up custom certificates for the CLI and other API calls.

You can configure multiple certificates this way, and each certificate can be associated with [multiple host names](#), [multiple routers](#), or the [OpenShift Container Platform image registry](#).

A default certificate must be configured in the **servingInfo.certFile** and **servingInfo.keyFile** configuration sections in addition to **namedCertificates**.



NOTE

The **namedCertificates** section should be configured only for the host name associated with the **masterPublicURL** and **oauthConfig.assetPublicURL** settings in the `/etc/origin/master/master-config.yaml` file. Using a custom serving certificate for the host name associated with the **masterURL** will result in TLS errors as infrastructure components will attempt to contact the master API using the internal **masterURL** host.

Custom Certificates Configuration

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt 1
  clientCA: ""
  keyFile: master.server.key 2
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt 3
      keyFile: wildcard.example.com.key 4
      names:
        - "openshift.example.com"
  metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

1 2 Path to certificate and key files for the CLI and other API calls.

3 4 Path to certificate and key files for the web console.

The **openshift_master_cluster_public_hostname** and **openshift_master_cluster_hostname** parameters in the [Ansible inventory file](#), by default `/etc/ansible/hosts`, must be different. If they are the same, the named certificates will fail and you will need to re-install them.

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=internal.paas.example.com
openshift_master_cluster_public_hostname=external.paas.example.com
```

For more information on using DNS with OpenShift Container Platform, see the [DNS installation prerequisites](#).

This approach allows you to take advantage of the self-signed certificates generated by OpenShift Container Platform and add custom trusted certificates to individual components as needed.

Note that the internal infrastructure certificates remain self-signed, which might be perceived as bad practice by some security or PKI teams. However, any risk here is minimal, as the only clients that trust these certificates are other components within the cluster. All external users and systems use custom trusted certificates.

Relative paths are resolved based on the location of the master configuration file. Restart the server to pick up the configuration changes.

9.4. CONFIGURING A CUSTOM MASTER HOST CERTIFICATE

In order to facilitate trusted connections with external users of OpenShift Container Platform, you can provision a named certificate that matches the domain name provided in the **openshift_master_cluster_public_hostname** parameter in the [Ansible inventory file](#), by default **/etc/ansible/hosts**.

You must place this certificate in a directory accessible to Ansible and add the path in the Ansible inventory file, as follows:

```
openshift_master_named_certificates=[{"certfile": "/path/to/console.ocp-
c1.myorg.com.crt", "keyfile": "/path/to/console.ocp-c1.myorg.com.key",
"names": ["console.ocp-c1.myorg.com"]}]]
```

Where the parameter values are:

- **certfile** is the path to the file that contains the OpenShift Container Platform router certificate.
- **keyfile** is the path to the file that contains the OpenShift Container Platform wildcard key.
- **names** is the cluster public hostname.

The file paths must be local to the system where Ansible runs. Certificates are copied to master hosts and are deployed within the **/etc/origin/master** directory.

When securing the registry, add the service hostnames and IP addresses to the server certificate for the registry. The Subject Alternative Names (SAN) must contain the following.

- Two service hostnames:

```
docker-registry.default.svc.cluster.local
docker-registry.default.svc
```

- Service IP address.
For example:

```
172.30.252.46
```

■

Use the following command to get the Docker registry service IP address:

```
oc get service docker-registry --template '{{.spec.clusterIP}}'
```

- Public hostname.

```
docker-registry-default.apps.example.com
```

Use the following command to get the Docker registry public hostname:

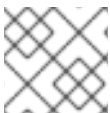
```
oc get route docker-registry --template '{{.spec.host}}'
```

For example, the server certificate should contain SAN details similar to the following:

```
X509v3 Subject Alternative Name:
      DNS:docker-registry-public.openshift.com, DNS:docker-
registry.default.svc, DNS:docker-registry.default.svc.cluster.local,
DNS:172.30.2.98, IP Address:172.30.2.98
```

9.5. CONFIGURING A CUSTOM WILDCARD CERTIFICATE FOR THE DEFAULT ROUTER

You can configure the OpenShift Container Platform default router with a default wildcard certificate. A default wildcard certificate provides a convenient way for applications that are deployed in OpenShift Container Platform to use default encryption without needing custom certificates.



NOTE

Default wildcard certificates are recommended for non-production environments only.

In order to configure a default wildcard certificate, provision a certificate that is valid for `.<app_domain>`, where `<app_domain>` is the value of `openshift_master_default_subdomain` in the [Ansible inventory file](#), by default `/etc/ansible/hosts`. Once provisioned, place the certificate, key, and ca certificate files on your Ansible host, and add the following line to your Ansible inventory file.

```
openshift_hosted_router_certificate={"certfile": "/path/to/apps.c1-
ocp.myorg.com.crt", "keyfile": "/path/to/apps.c1-ocp.myorg.com.key",
"cafile": "/path/to/apps.c1-ocp.myorg.com.ca.crt"}
```

For example:

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-
chain.cert.pem"}
```

Where the parameter values are:

- **certfile** is the path to the file that contains the OpenShift Container Platform router certificate.

- **keyfile** is the path to the file that contains the OpenShift Container Platform wildcard key.
- **cafile** is the path to the file that contains the root CA for this key and certificate. If an intermediate CA is in use, the file should contain both the intermediate and root CA.

If these certificate files are new to your OpenShift Container Platform cluster, run the Ansible **deploy_router.yml** playbook to add these files to the OpenShift Container Platform configuration files. The playbook adds the certificate files to the **/etc/origin/master/** directory.

```
# ansible-playbook [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    hosted/deploy_router.yml
```

If [the certificates are not new](#), for example, you want to change existing certificates or replace expired certificates, run the following playbook:

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-
    certificates.yml
```



NOTE

For this playbook to run, the certificate names must not change. If the certificate names change, rerun the Ansible **deploy_cluster.yml** playbook as if the certificates were new.

9.6. CONFIGURING A CUSTOM CERTIFICATE FOR THE IMAGE REGISTRY

The OpenShift Container Platform image registry is an internal service that facilitates builds and deployments. Most of the communication with the registry is handled by internal components in OpenShift Container Platform. As such, you should not need to replace the certificate used by the registry service itself.

However, by default, the registry uses routes to allow external systems and users to do pulls and pushes of images. You can use a *re-encrypt route* with a custom certificate that is presented to external users instead of using the internal, self-signed certificate.

To configure this, add the [following lines](#) of code to the **[OSEv3:vars]** section of the Ansible inventory file, by default **/etc/ansible/hosts** file. Specify the certificates to use with the registry route.

```
openshift_hosted_registry_routehost=registry.apps.c1-ocp.myorg.com 1
openshift_hosted_registry_routecertificates={"certfile":
"/path/to/registry.apps.c1-ocp.myorg.com.crt", "keyfile":
"/path/to/registry.apps.c1-ocp.myorg.com.key", "cafile":
"/path/to/registry.apps.c1-ocp.myorg.com-ca.crt"} 2
openshift_hosted_registry_routetermination=reencrypt 3
```

1 The host name of the registry.

2 The locations of the **cacert**, **root**, and **cafile** files.

- **certfile** is the path to the file that contains the OpenShift Container Platform router certificate.

- **keyfile** is the path to the file that contains the OpenShift Container Platform wildcard key.
- **cafile** is the path to the file that contains the root CA for this key and certificate. If an intermediate CA is in use, the file should contain both the intermediate and root CA.

3 Specify where encryption is performed:

- Set to **reencrypt** with a *re-encrypt route* to terminate encryption at the edge router and re-encrypt it with a new certificate supplied by the destination.
- Set to **passthrough** to terminate encryption at the destination. The destination is responsible for decrypting traffic.

9.7. CONFIGURING A CUSTOM CERTIFICATE FOR A LOAD BALANCER

If your OpenShift Container Platform cluster uses the default load balancer or an enterprise-level load balancer, you can use custom certificates to make the web console and API available externally using a publicly-signed custom certificate. leaving the existing internal certificates for the internal endpoints.

To configure OpenShift Container Platform to use custom certificates in this way:

1. Edit the **servingInfo** section of the [master configuration file](#):

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt
  clientCA: ""
  keyFile: master.server.key
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt 1
      keyFile: wildcard.example.com.key 2
      names:
        - "openshift.example.com"
  metricsPublicURL:
    "https://metrics.os.example.com/hawkular/metrics"
```

1 Path to the certificate file for the web console.

2 Path to the key file for the web console.



NOTE

Configure the **namedCertificates** section for only the host name associated with the **masterPublicURL** and **oauthConfig.assetPublicURL** settings. Using a custom serving certificate for the host name associated with the **masterURL** causes in TLS errors as infrastructure components attempt to contact the master API using the internal masterURL host.

- Specify the **openshift_master_cluster_public_hostname** and **openshift_master_cluster_hostname** parameters in the Ansible inventory file, by default **/etc/ansible/hosts**. These values must be different. If they are the same, the named certificates will fail.

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=paas.example.com 1
openshift_master_cluster_public_hostname=public.paas.example.com 2
```

- 1 The FQDN for internal load balancer configured for SSL passthrough.
- 2 The FQDN for external the load balancer with custom (public) certificate.

For information specific to your load balancer environment, refer to [the OpenShift Container Platform Reference Architecture for your provider](#) and [Custom Certificate SSL Termination \(Production\)](#).

9.8. RETROFIT CUSTOM CERTIFICATES INTO A CLUSTER

You can retrofit custom master and custom router certificates into an existing OpenShift Container Platform cluster by editing the the Ansible inventory file, by default **/etc/ansible/hosts**, and running a playbook.

9.8.1. Retrofit Custom Master Certificates into a Cluster

To retrofit custom certificates:

- Edit the Ansible inventory file to set the **openshift_master_overwrite_named_certificates=true**.
- Specify the path to the certificate using the **openshift_master_named_certificates** parameter.

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"certfile":
"/path/on/host/to/crt-file", "keyfile": "/path/on/host/to/key-file",
"names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-
file"}] 1
```

- 1 Path to a [master API certificate](#).

- Run the following playbook:

```
ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/redeploy-certificates.yml
```

9.8.2. Retrofit Custom Router Certificates into a Cluster

To retrofit custom router certificates:

- Edit the Ansible inventory file to set the **openshift_master_overwrite_named_certificates=true**.

2. Specify the path to the certificate using the **openshift_hosted_router_certificate** parameter.

```
openshift_master_overwrite_named_certificates=true
openshift_hosted_router_certificate={"certfile":
"/path/on/host/to/app-crt-file", "keyfile": "/path/on/host/to/app-
key-file", "cafile": "/path/on/host/to/app-ca-file"} ❶
```

- ❶ Path to a [wildcard API certificate](#).

3. Run the following playbook:

```
ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-hosted/redeploy-router-certificates.yml
```

9.9. USING CUSTOM CERTIFICATES WITH OTHER COMPONENTS

For information on how other components, such as Logging & Metrics, use custom certificates, see [Certificate Management](#).

CHAPTER 10. REDEPLOYING CERTIFICATES

10.1. OVERVIEW

OpenShift Container Platform uses certificates to provide secure connections for the following components:

- masters (API server and controllers)
- etcd
- nodes
- registry
- router

You can use Ansible playbooks provided with the installer to automate checking expiration dates for cluster certificates. Playbooks are also provided to automate backing up and redeploying these certificates, which can fix common certificate errors.

Possible use cases for redeploying certificates include:

- The installer detected the wrong host names and the issue was identified too late.
- The certificates are expired and you need to update them.
- You have a new CA and want to create certificates using it instead.

10.2. CHECKING CERTIFICATE EXPIRATIONS

You can use the installer to warn you about any certificates expiring within a configurable window of days and notify you about any certificates that have already expired. Certificate expiry playbooks use the Ansible role **openshift_certificate_expiry**.

Certificates examined by the role include:

- Master and node service certificates
- Router and registry service certificates from etcd secrets
- Master, node, router, registry, and *kubeconfig* files for **cluster-admin** users
- etcd certificates (including embedded)

10.2.1. Role Variables

The **openshift_certificate_expiry** role uses the following variables:

Table 10.1. Core Variables

Variable Name	Default Value	Description
<code>openshift_certificate_expiry_config_base</code>	<code>/etc/origin</code>	Base OpenShift Container Platform configuration directory.
<code>openshift_certificate_expiry_warning_days</code>	<code>30</code>	Flag certificates that will expire in this many days from now.
<code>openshift_certificate_expiry_show_all</code>	<code>no</code>	Include healthy (non-expired and non-warning) certificates in results.

Table 10.2. Optional Variables

Variable Name	Default Value	Description
<code>openshift_certificate_expiry_generate_html_report</code>	<code>no</code>	Generate an HTML report of the expiry check results.
<code>openshift_certificate_expiry_html_report_path</code>	<code>/tmp/cert-expiry-report.html</code>	The full path for saving the HTML report.
<code>openshift_certificate_expiry_save_json_results</code>	<code>no</code>	Save expiry check results as a JSON file.
<code>openshift_certificate_expiry_json_results_path</code>	<code>/tmp/cert-expiry-report.json</code>	The full path for saving the JSON report.

10.2.2. Running Certificate Expiration Playbooks

The OpenShift Container Platform installer provides a set of example certificate expiration playbooks, using different sets of configuration for the `openshift_certificate_expiry` role.

These playbooks must be used with an [inventory file](#) that is representative of the cluster. For best results, run **ansible-playbook** with the `-v` option.

Using the ***easy-mode.yaml*** example playbook, you can try the role out before tweaking it to your specifications as needed. This playbook:

- Produces JSON and stylized HTML reports in `/tmp/`.
- Sets the warning window very large, so you will almost always get results back.
- Includes all certificates (healthy or not) in the results.

easy-mode.yaml Playbook

```
- name: Check cert expirys
  hosts: nodes:masters:etcd
  become: yes
```

```
gather_facts: no
vars:
  openshift_certificate_expiry_warning_days: 1500
  openshift_certificate_expiry_save_json_results: yes
  openshift_certificate_expiry_generate_html_report: yes
  openshift_certificate_expiry_show_all: yes
roles:
  - role: openshift_certificate_expiry
```

To run the ***easy-mode.yaml*** playbook:

```
$ ansible-playbook -v -i <inventory_file> \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
  checks/certificate_expiry/easy-mode.yaml
```

Other Example Playbooks

The other example playbooks are also available to run directly out of the ***/usr/share/ansible/openshift-ansible/playbooks/certificate_expiry/*** directory.

Table 10.3. Other Example Playbooks

File Name	Usage
<i>default.yaml</i>	Produces the default behavior of the <i>openshift_certificate_expiry</i> role.
<i>html_and_json_default_paths.yaml</i>	Generates HTML and JSON artifacts in their default paths.
<i>longer_warning_period.yaml</i>	Changes the expiration warning window to 1500 days.
<i>longer-warning-period-json-results.yaml</i>	Changes the expiration warning window to 1500 days and saves the results as a JSON file.

To run any of these example playbooks:

```
$ ansible-playbook -v -i <inventory_file> \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
  checks/certificate_expiry/<playbook>
```

10.2.3. Output Formats

As noted above, there are two ways to format your check report. In JSON format for machine parsing, or as a stylized HTML page for easy skimming.

HTML Report

An example of an HTML report is provided with the installer. You can open the following file in your browser to view it:

/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.html

JSON Report

There are two top-level keys in the saved JSON results: **data** and **summary**.

The **data** key is a hash where the keys are the names of each host examined and the values are the check results for the certificates identified on each respective host.

The **summary** key is a hash that summarizes the total number of certificates:

- examined on the entire cluster
- that are OK
- expiring within the configured warning window
- already expired

For an example of the full JSON report, see */usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.json*.

The summary from the JSON data can be easily checked for warnings or expirations using a variety of command-line tools. For example, using **grep** you can look for the word **summary** and print out the two lines after the match (**-A2**):

```
$ grep -A2 summary /tmp/cert-expiry-report.json
    "summary": {
        "warning": 16,
        "expired": 0
```

If available, the **jq** tool can also be used to pick out specific values. The first two examples below show how to select just one value, either **warning** or **expired**. The third example shows how to select both values at once:

```
$ jq '.summary.warning' /tmp/cert-expiry-report.json
16

$ jq '.summary.expired' /tmp/cert-expiry-report.json
0

$ jq '.summary.warning, .summary.expired' /tmp/cert-expiry-report.json
16
0
```

10.3. REDEPLOYING CERTIFICATES

Use the following playbooks to redeploy master, etcd, node, registry, and router certificates on all relevant hosts. You can redeploy all of them at once using the current CA, redeploy certificates for specific components only, or redeploy a newly generated or custom CA on its own.

Just like the certificate expiry playbooks, these playbooks must be run with an [inventory file](#) that is representative of the cluster.

In particular, the inventory must specify or override all host names and IP addresses set via the following variables such that they match the current cluster configuration:

- **openshift_public_hostname**

- `openshift_public_ip`
- `openshift_master_cluster_hostname`
- `openshift_master_cluster_public_hostname`

The playbooks you need are provided by:

```
# yum install atomic-openshift-utils
```



NOTE

The validity (length in days until they expire) for any certificates auto-generated while redeploying can be configured via Ansible as well. See [Configuring Certificate Validity](#).



NOTE

OpenShift Container Platform CA and etcd certificates expire after five years. Signed OpenShift Container Platform certificates expire after two years.

10.3.1. Redeploying All Certificates Using the Current OpenShift Container Platform and etcd CA

The ***redeploy-certificates.yml*** playbook does *not* regenerate the OpenShift Container Platform CA certificate. New master, etcd, node, registry, and router certificates are created using the current CA certificate to sign new certificates.

This also includes serial restarts of:

- etcd
- master services
- node services

To redeploy master, etcd, and node certificates using the current OpenShift Container Platform CA, run this playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/redeploy-
    certificates.yml
```

10.3.2. Redeploying a New or Custom OpenShift Container Platform CA

The ***openshift-master/redeploy-openshift-ca.yml*** playbook redeploys the OpenShift Container Platform CA certificate by generating a new CA certificate and distributing an updated bundle to all components including client ***kubeconfig*** files and the node's database of trusted CAs (the CA-trust).

This also includes serial restarts of:

- master services
- node services

- docker

Additionally, you can specify a [custom CA certificate](#) when redeploying certificates instead of relying on a CA generated by OpenShift Container Platform.

When the master services are restarted, the registry and routers can continue to communicate with the master without being redeployed because the master's serving certificate is the same, and the CA the registry and routers have are still valid.

To redeploy a newly generated or custom CA:

1. Optionally, specify a custom CA. The **certfile** that you specify as part of the custom CA parameter, **openshift_master_ca_certificate**, must contain only the single certificate that signs the OpenShift Container Platform certificates. If you have intermediate certificates in your chain, you must bundle them into a different file.
 - a. To specify a CA without intermediate certificates, set the following variable in your inventory file:

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing
clusters.
# This option may only be specified when creating a new cluster
or
# when redeploying cluster certificates with the redeploy-
certificates
# playbook.
openshift_master_ca_certificate={'certfile': '</path/to/ca.crt>',
'keyfile': '</path/to/ca.key>'}
```

- b. To specify a CA certificate that is issued by an intermediate CA:
 - i. Create a bundled certificate that contains the full chain of intermediate and root certificates for the CA:

```
# cat intermediate/certs/<intermediate.cert.pem> \
certs/ca.cert.pem >> intermediate/certs/ca-
chain.cert.pem
```

- ii. Set the following variables in your inventory file:

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing
clusters.
# This option may only be specified when creating a new
cluster or
# when redeploying cluster certificates with the redeploy-
certificates
# playbook.
openshift_master_ca_certificate={'certfile':
'</path/to/ca.crt>', 'keyfile': '</path/to/ca.key>'}
openshift_additional_ca=intermediate/certs/ca-chain.cert.pem
```

2. Run the **openshift-master/redeploy-openshift-ca.yml** playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    master/redeploy-openshift-ca.yml
```

With the new OpenShift Container Platform CA in place, you can then use the [openshift-master/redeploy-certificates.yml](#) [playbook](#) at your discretion whenever you want to redeploy certificates signed by the new CA on all components.

10.3.3. Redeploying a New etcd CA

The **[openshift-etcd/redeploy-ca.yml](#)** [playbook](#) redeploys the etcd CA certificate by generating a new CA certificate and distributing an updated bundle to all etcd peers and master clients.

This also includes serial restarts of:

- etcd
- master services

To redeploy a newly generated etcd CA:

1. Run the **[openshift-etcd/redeploy-ca.yml](#)** [playbook](#), specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    etcd/redeploy-ca.yml
```

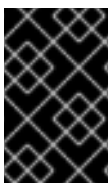
With the new etcd CA in place, you can then use the **[openshift-etcd/redeploy-certificates.yml](#)** [playbook](#) at your discretion whenever you want to redeploy certificates signed by the new etcd CA on etcd peers and master clients. Alternatively, you can use the **[redeploy-certificates.yml](#)** [playbook](#) to redeploy certificates for OpenShift Container Platform components in addition to etcd peers and master clients.

10.3.4. Redeploying Master Certificates Only

The **[openshift-master/redeploy-certificates.yml](#)** [playbook](#) only redeploys master certificates. This also includes serial restarts of master services.

To redeploy master certificates, run this [playbook](#), specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    master/redeploy-certificates.yml
```



IMPORTANT

After running this [playbook](#), you need to regenerate any [service signing certificate or key pairs](#) by deleting existing secrets that contain service serving certificates or removing and re-adding annotations to appropriate services.

10.3.5. Redeploying etcd Certificates Only

The ***openshift-etcd/redeploy-certificates.yml*** playbook only redeploys etcd certificates including master client certificates.

This also include serial restarts of:

- etcd
- master services.

To redeploy etcd certificates, run this playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    etcd/redeploy-certificates.yml
```

10.3.6. Redeploying Node Certificates Only

The ***openshift-node/redeploy-certificates.yml*** playbook only redeploys node certificates. This also include serial restarts of node services.

To redeploy node certificates, run this playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    node/redeploy-certificates.yml
```

10.3.7. Redeploying Registry or Router Certificates Only

The ***openshift-hosted/redeploy-registry-certificates.yml*** and ***openshift-hosted/redeploy-router-certificates.yml*** playbooks replace installer-created certificates for the registry and router. If custom certificates are in use for these components, see [Redeploying Custom Registry or Router Certificates](#) to replace them manually.

10.3.7.1. Redeploying Registry Certificates Only

To redeploy registry certificates, run the following playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    hosted/redeploy-registry-certificates.yml
```

10.3.7.2. Redeploying Router Certificates Only

To redeploy router certificates, run the following playbook, specifying your inventory file:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    hosted/redeploy-router-certificates.yml
```

10.3.8. Redeploying Custom Registry or Router Certificates

When nodes are evacuated due to a redeployed CA, registry and router pods are restarted. If the registry and router certificates were not also redeployed with the new CA, this can cause outages because they cannot reach the masters using their old certificates.

The playbooks for redeploying certificates cannot redeploy custom registry or router certificates, so to address this issue, you can manually redeploy the registry and router certificates.

10.3.8.1. Redeploying Registry Certificates Manually

To redeploy registry certificates manually, you must add new registry certificates to a secret named **registry-certificates**, then redeploy the registry:

1. Switch to the **default** project for the remainder of these steps:

```
$ oc project default
```

2. If your registry was initially created on OpenShift Container Platform 3.1 or earlier, it may still be using environment variables to store certificates (which has been deprecated in favor of using secrets).
 - a. Run the following and look for the **OPENSIFT_CA_DATA**, **OPENSIFT_CERT_DATA**, **OPENSIFT_KEY_DATA** environment variables:

```
$ oc env dc/docker-registry --list
```

- b. If they do not exist, skip this step. If they do, create the following **ClusterRoleBinding**:

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: registry-registry-role
roleRef:
  kind: ClusterRole
  name: system:registry
subjects:
- kind: ServiceAccount
  name: registry
  namespace: default
userNames:
- system:serviceaccount:default:registry
EOF
oc create -f -
```

Then, run the following to remove the environment variables:

```
$ oc env dc/docker-registry OPENSIFT_CA_DATA-
OPENSIFT_CERT_DATA- OPENSIFT_KEY_DATA- OPENSIFT_MASTER-
```

3. Set the following environment variables locally to make later commands less complex:

```
$ REGISTRY_IP=`oc get service docker-registry -o
```

```
jsonpath='{.spec.clusterIP}'`
$ REGISTRY_HOSTNAME=`oc get route/docker-registry -o
jsonpath='{.spec.host}'`
```

4. Create new registry certificates:

```
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --hostnames=$REGISTRY_IP,docker-registry.default.svc,docker-
registry.default.svc.cluster.local,$REGISTRY_HOSTNAME \
  --cert=/etc/origin/master/registry.crt \
  --key=/etc/origin/master/registry.key \
  --signer-serial=/etc/origin/master/ca.serial.txt
```

Run **oc adm** commands only from the first master listed in the Ansible host inventory file, by default **/etc/ansible/hosts**.

5. Update the **registry-certificates** secret with the new registry certificates:

```
$ oc create secret generic registry-certificates \
  --from-
file=/etc/origin/master/registry.crt,/etc/origin/master/registry.key
\
  -o json --dry-run | oc replace -f -
```

6. Redeploy the registry:

```
$ oc rollout latest dc/docker-registry
```

10.3.8.2. Redeploying Router Certificates Manually

When routers are initially deployed, an annotation is added to the router's service that automatically creates a [service serving certificate secret](#).

To redeploy router certificates manually, that service serving certificate can be triggered to be recreated by deleting the secret, removing and re-adding annotations to the **router** service, then redeploying the router:

1. Switch to the **default** project for the remainder of these steps:

```
$ oc project default
```

2. If your router was initially created on OpenShift Container Platform 3.1 or earlier, it might still use environment variables to store certificates, which has been deprecated in favor of using service serving certificate secret.

- a. Run the following command and look for the **OPENSHIFT_CA_DATA**, **OPENSHIFT_CERT_DATA**, **OPENSHIFT_KEY_DATA** environment variables:

```
$ oc env dc/router --list
```

- b. If those variables exist, create the following **ClusterRoleBinding**:

■

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: router-router-role
roleRef:
  kind: ClusterRole
  name: system:router
subjects:
- kind: ServiceAccount
  name: router
  namespace: default
userNames:
- system:serviceaccount:default:router
EOF
oc create -f -
```

- c. If those variables exist, run the following command to remove them:

```
$ oc env dc/router OPENSIFT_CA_DATA- OPENSIFT_CERT_DATA-
OPENSIFT_KEY_DATA- OPENSIFT_MASTER-
```

3. Obtain a certificate.

- If you use an external Certificate Authority (CA) to sign your certificates, create a new certificate and provide it to OpenShift Container Platform by following your internal processes.
- If you use the internal OpenShift Container Platform CA to sign certificates, run the following commands:



IMPORTANT

The following commands generate a certificate that is internally signed. It will be trusted by only clients that trust the OpenShift Container Platform CA.

```
$ cd /root
$ mkdir cert ; cd cert
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --signer-serial=/etc/origin/master/ca.serial.txt \
  --hostnames='*.hostnames.for.the.certificate' \
  --cert=router.crt \
  --key=router.key \
```

These commands generate the following files:

- A new certificate named ***router.crt***.
- A copy of the signing CA certificate chain, ***/etc/origin/master/ca.crt***. This chain can contain more than one certificate if you use intermediate CAs.

- o A corresponding private key named ***router.key***.

4. Create a new file that concatenates the generated certificates:

```
$ cat router.crt /etc/origin/master/ca.crt router.key > router.pem
```

5. Before you generate a new secret, back up the current one:

```
$ oc export secret router-certs > ~/old-router-certs-secret.yaml
```

6. Create a new secret to hold the new certificate and key, and replace the contents of the existing secret:

```
$ oc create secret tls router-certs --cert=router.pem \ ❶
--key=router.key -o json --dry-run | \
oc replace -f -
```

❶ ***router.pem*** is the file that contains the concatenation of the certificates that you generated.

7. Remove the following annotations from the ***router*** service:

```
$ oc annotate service router \
service.alpha.openshift.io/serving-cert-secret-name- \
service.alpha.openshift.io/serving-cert-signed-by-
```

8. Re-add the annotations:

```
$ oc annotate service router \
service.alpha.openshift.io/serving-cert-secret-name=router-certs
```

9. Redeploy the router:

```
$ oc rollout latest dc/router
```

CHAPTER 11. CONFIGURING AUTHENTICATION AND USER AGENT

11.1. OVERVIEW

The OpenShift Container Platform [master](#) includes a built-in [OAuth server](#). Developers and administrators obtain [OAuth access tokens](#) to authenticate themselves to the API.

As an administrator, you can configure OAuth using the [master configuration file](#) to specify an [identity provider](#). It is a best practice to configure your identity provider during [cluster installation](#), but you can configure it after installation.



NOTE

OpenShift Container Platform user names containing `/`, `:`, and `%` are not supported.

The [Deny All](#) identity provider is used by default, which denies access for all user names and passwords. To allow access, you must choose a different identity provider and configure the master configuration file appropriately (located at `/etc/origin/master/master-config.yaml` by default).

When you run a master without a configuration file, the [Allow All](#) identity provider is used by default, which allows any non-empty user name and password to log in. This is useful for testing purposes. To use other identity providers, or to modify any [token](#), [grant](#), or [session options](#), you must run the master from a configuration file.



NOTE

[Roles](#) need to be assigned to administer the setup with an external user.

After making changes to an identity provider, you must restart the master services for the changes to take effect:

```
# master-restart api
# master-restart controllers
```

11.2. IDENTITY PROVIDER PARAMETERS

There are four parameters common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
challenge	<p>When true, unauthenticated token requests from non-web clients (like the CLI) are sent a WWW-Authenticate challenge header. Not supported by all identity providers.</p> <p>To prevent cross-site request forgery (CSRF) attacks against browser clients Basic authentication challenges are only sent if a X-CSRF-Token header is present on the request. Clients that expect to receive Basic WWW-Authenticate challenges should set this header to a non-empty value.</p>

Parameter	Description
login	<p>When true, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider. Not supported by all identity providers.</p> <p>If you want users to be sent to a branded page before being redirected to the identity provider's login, then set oauthConfig → alwaysShowProviderSelection: true in the master configuration file. This provider selection page can be customized.</p>
mappingMethod	<p>Defines how new identities are mapped to users when they log in. Enter one of the following values:</p> <p>claim</p> <p>The default value. Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity.</p> <p>lookup</p> <p>Looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. This allows cluster administrators to set up identities and users manually, or using an external process. Using this method requires you to manually provision users. See Manually Provisioning a User When Using the Lookup Mapping Method.</p> <p>generate</p> <p>Provisions a user with the identity's preferred user name. If a user with the preferred user name is already mapped to an existing identity, a unique user name is generated. For example, myuser2. This method should not be used in combination with external processes that require exact matches between OpenShift Container Platform user names and identity provider user names, such as LDAP group sync.</p> <p>add</p> <p>Provisions a user with the identity's preferred user name. If a user with that user name already exists, the identity is mapped to the existing user, adding to any existing identity mappings for the user. Required when multiple identity providers are configured that identify the same set of users and map to the same user names.</p>

**NOTE**

When adding or changing identity providers, you can map identities from the new provider to existing users by setting the **mappingMethod** parameter to **add**.

11.3. CONFIGURING IDENTITY PROVIDERS

OpenShift Container Platform supports configuring only a single identity provider. However, you can extend the basic authentication for more complex configurations such as [LDAP failover](#).

You can use these parameters to define the identity provider during installation or after installation.

11.3.1. Configuring identity providers with Ansible

For initial cluster installations, the [Deny All](#) identity provider is configured by default, though it can be overridden during installation by configuring **openshift_master_identity_providers** parameter in the inventory file. [Session options in the OAuth configuration](#) are also configurable in the inventory file.

Example identity provider configuration with Ansible

```
# httpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'user1': '<pre-hashed password>',
'user2': '<pre-hashed password>'}
# or
#openshift_master_htpasswd_file=<path to local pre-generated htpasswd
file>

# Allow all auth
#openshift_master_identity_providers=[{'name': 'allow_all', 'login':
'true', 'challenge': 'true', 'kind': 'AllowAllPasswordIdentityProvider'}]

# LDAP auth
#openshift_master_identity_providers=[{'name': 'my_ldap_provider',
'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email':
['mail'], 'name': ['cn'], 'preferredUsername': ['uid']}, 'bindDN': '',
'bindPassword': '', 'ca': '', 'insecure': 'false', 'url':
'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]
# Configuring the ldap ca certificate 1
#openshift_master_ldap_ca=<ca text>
# or
#openshift_master_ldap_ca_file=<path to local ca file to use>

# Available variables for configuring certificates for other identity
providers:
#openshift_master_openid_ca
#openshift_master_openid_ca_file
#openshift_master_request_header_ca
#openshift_master_request_header_ca_file
```

- 1** If you specify your CA certificate location in the **openshift_master_identity_providers** parameter, do not specify a certificate value in the **openshift_master_ldap_ca** parameter or path in the **openshift_master_ldap_ca_file** parameter.

11.3.2. Configuring identity providers in the master configuration file

You can configure the master host for authentication using your desired identity provider by modifying the [master configuration file](#).

Example 11.1. Example identity provider configuration in the master configuration file

```
...
oauthConfig:
  identityProviders:
    - name: htpasswd_auth
      challenge: true
      login: true
      mappingMethod: "claim"
  ...
```


When set to the default **claim** value, OAuth will fail if the identity is mapped to a previously-existing user name.

11.3.3. Configuring an identity provider or method

11.3.3.1. Manually provisioning a user when using the lookup mapping method

When using the **lookup** mapping method, user provisioning is done by an external system, via the API. Typically, identities are automatically mapped to users during login. The 'lookup' mapping method automatically disables this automatic mapping, which requires you to provision users manually.

For more information on identity objects, see the [Identity](#) user API object.

If you are using the **lookup** mapping method, use the following steps for each user after configuring the identity provider:

1. Create an OpenShift Container Platform User, if not created already:

```
$ oc create user <username>
```

For example, the following command creates a OpenShift Container Platform User **bob**:

```
$ oc create user bob
```

2. Create an OpenShift Container Platform Identity, if not created already. Use the name of the identity provider and the name that uniquely represents this identity in the scope of the identity provider:

```
$ oc create identity <identity-provider>:<user-id-from-identity-provider>
```

The **<identity-provider>** is the name of the identity provider in the master configuration, as shown in the appropriate identity provider section below.

For example, the following commands creates an Identity with identity provider **ldap_provider** and the identity provider user name **bob_s**.

```
$ oc create identity ldap_provider:bob_s
```

3. Create a user/identity mapping for the created user and identity:

```
$ oc create useridentitymapping <identity-provider>:<user-id-from-identity-provider> <username>
```

For example, the following command maps the identity to the user:

```
$ oc create useridentitymapping ldap_provider:bob_s bob
```

11.3.4. Allow all

Set **AllowAllPasswordIdentityProvider** in the **identityProviders** stanza to allow any non-empty user name and password to log in.

Example 11.2. Master Configuration Using AllowAllPasswordIdentityProvider

```

oauthConfig:
  ...
  identityProviders:
  - name: my_allow_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider

```

- ❶ This provider name is prefixed to provider user names to form an identity name.
- ❷ When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- ❸ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- ❹ Controls how mappings are established between this provider's identities and user objects, [as described above](#).

11.3.5. Deny all

Set **DenyAllPasswordIdentityProvider** in the **identityProviders** stanza to deny access for all user names and passwords.

Example 11.3. Master Configuration Using DenyAllPasswordIdentityProvider

```

oauthConfig:
  ...
  identityProviders:
  - name: my_deny_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: DenyAllPasswordIdentityProvider

```

- ❶ This provider name is prefixed to provider user names to form an identity name.
- ❷ When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- ❸ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- ❹

Controls how mappings are established between this provider's identities and user objects, [as described above](#).

11.3.6. HTPasswd

Set **HTPasswdPasswordIdentityProvider** in the **identityProviders** stanza to validate user names and passwords against a flat file generated using **htpasswd**.



NOTE

The **htpasswd** utility is in the **httpd-tools** package:

```
# yum install httpd-tools
```

OpenShift Container Platform supports the Bcrypt, SHA-1, and MD5 cryptographic hash functions, and MD5 is the default for **htpasswd**. Plaintext, encrypted text, and other hash functions are not currently supported.

The flat file is reread if its modification time changes, without requiring a server restart.

To use the **htpasswd** command:

- To create a flat file with a user name and hashed password, run:

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```

Then, enter and confirm a clear-text password for the user. The command generates a hashed version of the password.

For example:

```
htpasswd -c users.htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```



NOTE

You can include the **-b** option to supply the password on the command line:

```
$ htpasswd -c -b <user_name> <password>
```

For example:

```
$ htpasswd -c -b file user1 MyPassword!
Adding password for user user1
```

- To add or update a login to the file, run:

```
$ htpasswd </path/to/users.htpasswd> <user_name>
```

- To remove a login from the file, run:

```
$ htpasswd -D </path/to/users.htpasswd> <user_name>
```

Example 11.4. Master Configuration Using HTTPasswdPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: HTTPasswdPasswordIdentityProvider
      file: /path/to/users.htpasswd ❺
```

- ❶ This provider name is prefixed to provider user names to form an identity name.
- ❷ When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- ❸ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- ❹ Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- ❺ File generated using [htpasswd](#).

11.3.7. Keystone

[Keystone](#) is an OpenStack project that provides identity, token, catalog, and policy services. You can integrate your OpenShift Container Platform cluster with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database. Once configured, this configuration allows users to log in to OpenShift Container Platform with their Keystone credentials.

11.3.7.1. Configuring authentication on the master

1. If you have:
 - Already completed the installation of Openshift, then copy the */etc/origin/master/master-config.yaml* file into a new directory; for example:

```
$ cd /etc/origin/master
$ mkdir keystoneconfig; cp master-config.yaml keystoneconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

For example:

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=keystoneconfig
```



NOTE

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new **keystoneconfig/master-config.yaml** file's **identityProviders** stanza, and copy the example **KeystonePasswordIdentityProvider** configuration and paste it to replace the existing stanza:

```
oauthConfig:
  ...
  identityProviders:
  - name: my_keystone_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: KeystonePasswordIdentityProvider
      domainName: default 5
      url: http://keystone.example.com:5000 6
      ca: ca.pem 7
      certFile: keystone.pem 8
      keyFile: keystonekey.pem 9
```

- 1 This provider name is prefixed to provider user names to form an identity name.
- 2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- 3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 Keystone domain name. In Keystone, usernames are domain-specific. Only a single domain is supported.

- 6 The URL to use to connect to the Keystone server (required).
- 7 Optional: Certificate bundle to use to validate server certificates for the configured URL.
- 8 Optional: Client certificate to present when making requests to the configured URL.
- 9 Key for the client certificate. Required if **certFile** is specified.

3. Make the following modifications to the **identityProviders** stanza:

- a. Change the provider **name** ("my_keystone_provider") to match your Keystone server. This name is prefixed to provider user names to form an identity name.
- b. If required, **change mappingMethod** to control how mappings are established between the provider's identities and user objects.
- c. Change the **domainName** to the domain name of your OpenStack Keystone server. In Keystone, user names are domain-specific. Only a single domain is supported.
- d. Specify the **url** to use to connect to your OpenStack Keystone server.
- e. Optionally, change the **ca** to the certificate bundle to use in order to validate server certificates for the configured URL.
- f. Optionally, change the **certFile** to the client certificate to present when making requests to the configured URL.
- g. If **certFile** is specified, then you must change the **keyFile** to the key for the client certificate.

4. Save your changes and close the file.

5. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their Keystone credentials.

11.3.7.2. Creating Users with Keystone Authentication

You do not create users in OpenShift Container Platform when integrating with an external authentication provider, such as, in this case, Keystone. Keystone is the system of record, meaning that users are defined in a Keystone database, and any user with a valid Keystone user name for the configured authentication server can log in.

To add a user to OpenShift Container Platform, the user must exist in the Keystone database, and if required you must create a new Keystone account for the user.

11.3.7.3. Verifying Users

Once one or more users have logged in, you can run **oc get users** to view a list of users and verify that users were created successfully:

Example 11.5. Output of `oc get users` command

```
$ oc get users
NAME                                UID                                FULL NAME
IDENTITIES
bobsmith                            a0c1d95c-1cb5-11e6-a04a-002186a28631  Bob Smith
keystone:bobsmith 1
```

- 1 Identities in OpenShift Container Platform are comprised of the identity provider name prefixed to the Keystone user name.

From here, you might want to learn how to [manage user roles](#).

11.3.8. LDAP authentication

Set **LDAPPasswordIdentityProvider** in the **identityProviders** stanza to validate user names and passwords against an LDAPv3 server, using simple bind authentication.

**NOTE**

If you require failover for your LDAP server, instead of following these steps, extend the basic authentication method by [configuring SSSD for LDAP failover](#).

During authentication, the LDAP directory is searched for an entry that matches the provided user name. If a single unique match is found, a simple bind is attempted using the distinguished name (DN) of the entry plus the provided password.

These are the steps taken:

1. Generate a search filter by combining the attribute and filter in the configured **url** with the user-provided user name.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny access.
3. Attempt to bind to the LDAP server using the DN of the entry retrieved from the search, and the user-provided password.
4. If the bind is unsuccessful, deny access.
5. If the bind is successful, build an identity using the configured attributes as the identity, email address, display name, and preferred user name.

The configured **url** is an RFC 2255 URL, which specifies the LDAP host and search parameters to use. The syntax of the URL is:

```
ldap://host:port/basedn?attribute?scope?filter
```

For the above example:

URL Component	Description
ldap	For regular LDAP, use the string ldap . For secure LDAP (LDAPS), use ldaps instead.
host:port	The name and port of the LDAP server. Defaults to localhost:389 for ldap and localhost:636 for LDAPS.
basedn	The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but it could also specify a subtree in the directory.
attribute	The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use uid . It is recommended to choose an attribute that will be unique across all entries in the subtree you will be using.
scope	The scope of the search. Can be either one or sub . If the scope is not provided, the default is to use a scope of sub .
filter	A valid LDAP search filter. If not provided, defaults to (objectClass=*)

When doing searches, the attribute, filter, and provided user name are combined to create a search filter that looks like:

```
(<filter>(<attribute>=<username>))
```

For example, consider a URL of:

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

When a client attempts to connect using a user name of **bob**, the resulting search filter will be **(&(enabled=true)(cn=bob))**.

If the LDAP directory requires authentication to search, specify a **bindDN** and **bindPassword** to use to perform the entry search.

Master Configuration Using LDAPPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
```



```

id: 5
  - dn
email: 6
  - mail
name: 7
  - cn
preferredUsername: 8
  - uid
bindDN: "" 9
bindPassword: "" 10
ca: my-ldap-ca-bundle.crt 11
insecure: false 12
url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13

```

- 1 This provider name is prefixed to the returned user ID to form an identity name.
- 2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- 3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 List of attributes to use as the identity. First non-empty attribute is used. At least one attribute is required. If none of the listed attribute have a value, authentication fails.
- 6 List of attributes to use as the email address. First non-empty attribute is used.
- 7 List of attributes to use as the display name. First non-empty attribute is used.
- 8 List of attributes to use as the preferred user name when provisioning a user for this identity. First non-empty attribute is used.
- 9 Optional DN to use to bind during the search phase.
- 10 Optional password to use to bind during the search phase. This value may also be provided in an [environment variable](#), [external file](#), or [encrypted file](#).
- 11 Certificate bundle to use to validate server certificates for the configured URL. If empty, system trusted roots are used. Only applies if **insecure: false**.
- 12 When **true**, no TLS connection is made to the server. When **false**, **ldaps://** URLs connect using TLS, and **ldap://** URLs are upgraded to TLS.
- 13 An RFC 2255 URL which specifies the LDAP host and search parameters to use, [as described above](#).



NOTE

To whitelist users for an LDAP integration, use the **lookup** mapping method. Before a login from LDAP would be allowed, a cluster administrator must create an identity and user object for each LDAP user.

11.3.9. Basic authentication (remote)

Basic Authentication is a generic backend integration mechanism that allows users to log in to OpenShift Container Platform with credentials validated against a remote identity provider.

Because basic authentication is generic, you can use this identity provider for advanced authentication configurations. You can configure [LDAP failover](#) or use the [containerized basic authentication](#) repository as a starting point for another advanced remote basic authentication configuration.

CAUTION

Basic authentication must use an HTTPS connection to the remote server to prevent potential snooping of the user ID and password and man-in-the-middle attacks.

With **BasicAuthPasswordIdentityProvider** configured, users send their user name and password to OpenShift Container Platform, which then validates those credentials against a remote server by making a server-to-server request, passing the credentials as a Basic Auth header. This requires users to send their credentials to OpenShift Container Platform during login.



NOTE

This only works for user name/password login mechanisms, and OpenShift Container Platform must be able to make network requests to the remote authentication server.

Set **BasicAuthPasswordIdentityProvider** in the **identityProviders** stanza to validate user names and passwords against a remote server using a server-to-server Basic authentication request. User names and passwords are validated against a remote URL that is protected by Basic authentication and returns JSON.

A **401** response indicates failed authentication.

A non-**200** status, or the presence of a non-empty "error" key, indicates an error:

```
{"error": "Error message"}
```

A **200** status with a **sub** (subject) key indicates success:

```
{"sub": "userid"} 1
```

¹ The subject must be unique to the authenticated user and must not be able to be modified.

A successful response may optionally provide additional data, such as:

- A display name using the **name** key. For example:

```
{"sub": "userid", "name": "User Name", ...}
```

- An email address using the **email** key. For example:

```
{"sub": "userid", "email": "user@example.com", ...}
```

- A preferred user name using the **preferred_username** key. This is useful when the unique,

unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity. For example:

```
{ "sub": "014fbff9a07c", "preferred_username": "bob", ... }
```

11.3.9.1. Configuring authentication on the master

1. If you have:

- Already completed the installation of Openshift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ mkdir basicauthconfig; cp master-config.yaml basicauthconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

For example:

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=basicauthconfig
```



NOTE

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new `master-config.yaml` file's **identityProviders** stanza, and copy [the example BasicAuthPasswordIdentityProvider configuration](#) and paste it to replace the existing stanza:

```
oauthConfig:
  ...
  identityProviders:
  - name: my_remote_basic_auth_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://www.example.com/remote-idp ❺
```

```
ca: /path/to/ca.file 6
certFile: /path/to/client.crt 7
keyFile: /path/to/client.key 8
```

- 1** This provider name is prefixed to the returned user ID to form an identity name.
- 2** When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.
- 3** When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.
- 4** Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5** URL accepting credentials in Basic authentication headers.
- 6** Optional: Certificate bundle to use to validate server certificates for the configured URL.
- 7** Optional: Client certificate to present when making requests to the configured URL.
- 8** Key for the client certificate. Required if **certFile** is specified.

Make the following modifications to the **identityProviders** stanza:

- a. Set the provider **name** to something unique and relevant to your deployment. This name is prefixed to the returned user ID to form an identity name.
 - b. If required, [set mappingMethod](#) to control how mappings are established between the provider's identities and user objects.
 - c. Specify the HTTPS **url** to use to connect to a server that accepts credentials in Basic authentication headers.
 - d. Optionally, set the **ca** to the certificate bundle to use in order to validate server certificates for the configured URL, or leave it empty to use the system-trusted roots.
 - e. Optionally, remove or set the **certFile** to the client certificate to present when making requests to the configured URL.
 - f. If **certFile** is specified, then you must set the **keyFile** to the key for the client certificate.
3. Save your changes and close the file.
 4. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-
config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their Basic authentication credentials.

11.3.9.2. Troubleshooting

The most common issue relates to network connectivity to the backend server. For simple debugging, run **curl** commands on the master. To test for a successful login, replace the **<user>** and **<password>** in the following example command with valid credentials. To test an invalid login, replace them with false credentials.

```
curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key
/path/to/client.key -u <user>:<password> -v
https://www.example.com/remote-idp
```

Successful responses

A **200** status with a **sub** (subject) key indicates success:

```
{"sub": "userid"}
```

The subject must be unique to the authenticated user, and must not be able to be modified.

A successful response may optionally provide additional data, such as:

- A display name using the **name** key:

```
{"sub": "userid", "name": "User Name", ...}
```

- An email address using the **email** key:

```
{"sub": "userid", "email": "user@example.com", ...}
```

- A preferred user name using the **preferred_username** key:

```
{"sub": "014fbff9a07c", "preferred_username": "bob", ...}
```

The **preferred_username** key is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity.

Failed responses

- A **401** response indicates failed authentication.
- A non-**200** status or the presence of a non-empty "error" key indicates an error:
{"error": "Error message"}

11.3.10. Request header

Set **RequestHeaderIdentityProvider** in the **identityProviders** stanza to identify users from request header values, such as **X-Remote-User**. It is typically used in combination with an authenticating proxy, which sets the request header value. This is similar to how [the remote user plug-in in OpenShift Enterprise 2](#) allowed administrators to provide Kerberos, LDAP, and many other forms of enterprise authentication.

You can also use the request header identity provider for advanced configurations such as [SAML authentication](#).

For users to authenticate using this identity provider, they must access

https://<master>/oauth/authorize (and subpaths) via an authenticating proxy. To accomplish this, configure the OAuth server to redirect unauthenticated requests for OAuth tokens to the proxy endpoint that proxies to **https://<master>/oauth/authorize**.

To redirect unauthenticated requests from clients expecting browser-based login flows:

1. Set the **login** parameter to **true**.
2. Set the **provider.loginURL** parameter to the authenticating proxy URL that will authenticate interactive clients and then proxy the request to **https://<master>/oauth/authorize**.

To redirect unauthenticated requests from clients expecting **WWW-Authenticate** challenges:

1. Set the **challenge** parameter to **true**.
2. Set the **provider.challengeURL** parameter to the authenticating proxy URL that will authenticate clients expecting **WWW-Authenticate** challenges and then proxy the request to **https://<master>/oauth/authorize**.

The **provider.challengeURL** and **provider.loginURL** parameters can include the following tokens in the query portion of the URL:

- **\${url}** is replaced with the current URL, escaped to be safe in a query parameter.
For example: **https://www.example.com/sso-login?then=\${url}**
- **\${query}** is replaced with the current query string, unescaped.
For example: **https://www.example.com/auth-proxy/oauth/authorize?\${query}**



WARNING

If you expect unauthenticated requests to reach the OAuth server, a **clientCA** parameter **MUST** be set for this identity provider, so that incoming requests are checked for a valid client certificate before the request's headers are checked for a user name. Otherwise, any direct request to the OAuth server can impersonate any identity from this provider, merely by setting a request header.

Master Configuration Using RequestHeaderIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_request_header_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://www.example.com/challenging-
```

```

proxy/oauth/authorize?${query}" 5
  loginURL: "https://www.example.com/login-proxy/oauth/authorize?
${query}" 6
  clientCA: /path/to/client-ca.file 7
  clientCommonNames: 8
  - my-auth-proxy
  headers: 9
  - X-Remote-User
  - SS0-User
  emailHeaders: 10
  - X-Remote-User-Email
  nameHeaders: 11
  - X-Remote-User-Display-Name
  preferredUsernameHeaders: 12
  - X-Remote-User-Login

```

- 1 This provider name is prefixed to the user name in the request header to form an identity name.
- 2 **RequestHeaderIdentityProvider** can only respond to clients that request **WWW-Authenticate** challenges by redirecting to a configured **challengeURL**. The configured URL should respond with a **WWW-Authenticate** challenge.
- 3 **RequestHeaderIdentityProvider** can only respond to clients requesting a login flow by redirecting to a configured **loginURL**. The configured URL should respond with a login flow.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 Optional: URL to redirect unauthenticated **/oauth/authorize** requests to, that will authenticate browser-based clients and then proxy their request to **https://<master>/oauth/authorize**. The URL that proxies to **https://<master>/oauth/authorize** must end with **/authorize** (with no trailing slash), and also proxy subpaths, in order for OAuth approval flows to work properly. **\${url}** is replaced with the current URL, escaped to be safe in a query parameter. **\${query}** is replaced with the current query string.
- 6 Optional: URL to redirect unauthenticated **/oauth/authorize** requests to, that will authenticate clients which expect **WWW-Authenticate** challenges, and then proxy them to **https://<master>/oauth/authorize**. **\${url}** is replaced with the current URL, escaped to be safe in a query parameter. **\${query}** is replaced with the current query string.
- 7 Optional: PEM-encoded certificate bundle. If set, a valid client certificate must be presented and validated against the certificate authorities in the specified file before the request headers are checked for user names.
- 8 Optional: list of common names (**cn**). If set, a valid client certificate with a Common Name (**cn**) in the specified list must be presented before the request headers are checked for user names. If empty, any Common Name is allowed. Can only be used in combination with **clientCA**.
- 9 Header names to check, in order, for the user identity. The first header containing a value is used as the identity. Required, case-insensitive.
- 10 Header names to check, in order, for an email address. The first header containing a value is used as the email address. Optional, case-insensitive.

- 11 Header names to check, in order, for a display name. The first header containing a value is used as the display name. Optional, case-insensitive.
- 12 Header names to check, in order, for a preferred user name, if different than the immutable identity determined from the headers specified in **headers**. The first header containing a value is used as the preferred user name when provisioning. Optional, case-insensitive.

Apache authentication using Request header

This example configures an authentication proxy on the same host as the master. Having the proxy and master on the same host is merely a convenience and may not be suitable for your environment. For example, if you were already [running a router](#) on the master, port 443 would not be available.

It is also important to note that while this reference configuration uses Apache's **mod_auth_form**, it is by no means required and other proxies can easily be used if the following requirements are met:

1. Block the **X-Remote-User** header from client requests to prevent spoofing.
2. Enforce client certificate authentication in the **RequestHeaderIdentityProvider** configuration.
3. Require the **X-Csrf-Token** header be set for all authentication request using the challenge flow.
4. Only the **/oauth/authorize** endpoint and its subpaths should be proxied, and redirects should not be rewritten to allow the backend server to send the client to the correct location.
5. The URL that proxies to **https://<master>/oauth/authorize** must end with **/authorize** (with no trailing slash). For example:
 - **https://proxy.example.com/login-proxy/authorize?... → https://<master>/oauth/authorize?...**
6. Subpaths of the URL that proxies to **https://<master>/oauth/authorize** must proxy to subpaths of **https://<master>/oauth/authorize**. For example:
 - **https://proxy.example.com/login-proxy/authorize/approve?... → https://<master>/oauth/authorize/approve?...**

Installing the prerequisites

1. The **mod_auth_form** module is shipped as part of the **mod_session** package that is found in the [Optional channel](#). Install the following packages:

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl
```

2. Generate a CA for validating requests that submit the trusted header. This CA should be used as the file name for **clientCA** in the [master's identity provider configuration](#).

```
# oc adm ca create-signer-cert \
  --cert='/etc/origin/master/proxyca.crt' \
  --key='/etc/origin/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/origin/master/proxyca.serial.txt'
```




NOTE

The **oc adm ca create-signer-cert** command generates a certificate that is valid for five years. This can be altered with the **--expire-days** option, but for security reasons, it is recommended to not make it greater than this value.

Run **oc adm** commands only from the first master listed in the Ansible host inventory file, by default **/etc/ansible/hosts**.

3. Generate a client certificate for the proxy. This can be done using any x509 certificate tooling. For convenience, the **oc adm** CLI can be used:

```
# oc adm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy' ❶

# pushd /etc/origin/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt ❷
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

- ❶ The user name can be anything, however it is useful to give it a descriptive name as it will appear in logs.
- ❷ When running the authentication proxy on a different host name than the master, it is important to generate a certificate that matches the host name instead of using the default master certificate as shown above. The value for **masterPublicURL** in the **/etc/origin/master/master-config.yaml** file must be included in the **X509v3 Subject Alternative Name** in the certificate that is specified for **SSLCertificateFile**. If a new certificate needs to be created, the **oc adm ca create-server-cert** command can be used.



NOTE

The **oc adm create-api-client-config** command generates a certificate that is valid for two years. This can be altered with the **--expire-days** option, but for security reasons, it is recommended to not make it greater than this value. Run **oc adm** commands only from the first master listed in the Ansible host inventory file, by default **/etc/ansible/hosts**.

Configuring Apache

This proxy does not need to reside on the same host as the master. It uses a client certificate to connect to the master, which is configured to trust the **X-Remote-User** header.

1. Create the certificate for the Apache configuration. The certificate that you specify as the

SSLProxyMachineCertificateFile parameter value is the proxy's client cert that is used to authenticate the proxy to the server. It must use **TLS Web Client Authentication** as the extended key type.

2. Configure Apache per the following:

```
LoadModule auth_form_module modules/mod_auth_form.so
LoadModule session_module modules/mod_session.so
LoadModule request_module modules/mod_request.so

# Nothing needs to be served over HTTP. This virtual host simply
# redirects to
# HTTPS.
<VirtualHost *:80>
    DocumentRoot /var/www/html
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    # This needs to match the certificates you generated. See the CN and
    X509v3
    # Subject Alternative Name in the output of:
    # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
    ServerName www.example.com

    DocumentRoot /var/www/html
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
    SSLCACertificateFile /etc/pki/CA/certs/ca.crt

    SSLProxyEngine on
    SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
    # It's critical to enforce client certificates on the Master. Otherwise
    # requests could spoof the X-Remote-User header by accessing the
    Master's
    # /oauth/authorize endpoint directly.
    SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

    # Send all requests to the console
    RewriteEngine On
    RewriteRule ^/console(.*)$ https://%{HTTP_HOST}:8443/console$1
    [R,L]

    # In order to using the challenging-proxy an X-Csrftoken must be
    present.
    RewriteCond %{REQUEST_URI} ^/challenging-proxy
    RewriteCond %{HTTP:X-Csrftoken} ^$ [NC]
    RewriteRule ^.* - [F,L]

    <Location /challenging-proxy/oauth/authorize>
        # Insert your backend server name/ip here.
        ProxyPass https://[MASTER]:8443/oauth/authorize
        AuthType basic
    </Location>
```

```

<Location /login-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize

    # mod_auth_form providers are implemented by mod_authn_dbm,
mod_authn_file,
    # mod_authn_dbd, mod_authnz_ldap and mod_authn_socache.
    AuthFormProvider file
    AuthType form
    AuthName openshift
    ErrorDocument 401 /login.html
</Location>

<ProxyMatch /oauth/authorize>
    AuthUserFile /etc/origin/master/htpasswd
    AuthName openshift
    Require valid-user
    RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

    # For ldap:
    # AuthBasicProvider ldap
    # AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-
domain,dc=com?uid?sub?(objectClass=*)"

    # It's possible to remove the mod_auth_form usage and replace it with
    # something like mod_auth_kerb, mod_auth_gssapi or even
mod_auth_mellon.
    # The former would be able to support both the login and challenge
flows
    # from the Master. Mellon would likely only support the login flow.

    # For Kerberos
    # yum install mod_auth_gssapi
    # AuthType GSSAPI
    # GssapiCredStore keytab:/etc/httpd.keytab
</ProxyMatch>

</VirtualHost>

RequestHeader unset X-Remote-User

```

Additional mod_auth_form requirements

A sample login page is available from the [openshift_extras](#) repository. This file should be placed in the **DocumentRoot** location (**/var/www/html** by default).

Creating users

At this point, you can create the users in the system Apache is using to store accounts information. In this example, file-backed authentication is used:

```

# yum -y install httpd-tools
# touch /etc/origin/master/htpasswd
# htpasswd /etc/origin/master/htpasswd <user_name>

```

Configuring the master

The **identityProviders** stanza in the */etc/origin/master/master-config.yaml* file must be updated as well:

```
identityProviders:
- name: requestheader
  challenge: true
  login: true
  provider:
    apiVersion: v1
    kind: RequestHeaderIdentityProvider
    challengeURL: "https://[MASTER]/challenging-proxy/oauth/authorize?
${query}"
    loginURL: "https://[MASTER]/login-proxy/oauth/authorize?${query}"
    clientCA: /etc/origin/master/proxyca.crt
    headers:
    - X-Remote-User
```

Restarting services

Finally, restart the following services:

```
# systemctl restart httpd
# master-restart api
# master-restart controllers
```

Verifying the configuration

1. Test by bypassing the proxy. You should be able to request a token if you supply the correct client certificate and header:

```
# curl -L -k -H "X-Remote-User: joe" \
  --cert /etc/pki/tls/certs/authproxy.pem \
  https://[MASTER]:8443/oauth/token/request
```

2. If you do not supply the client certificate, the request should be denied:

```
# curl -L -k -H "X-Remote-User: joe" \
  https://[MASTER]:8443/oauth/token/request
```

3. This should show a redirect to the configured **challengeURL** (with additional query parameters):

```
# curl -k -v -H 'X-Csrf-Token: 1' \
  '<masterPublicURL>/oauth/authorize?client_id=openshift-
challenging-client&response_type=token'
```

4. This should show a 401 response with a **WWW-Authenticate** basic challenge:

```
# curl -k -v -H 'X-Csrf-Token: 1' \
  '<redirected_challengeURL_from_step_3 +query>'
```

5. This should show a redirect with an access token:

```
# curl -k -v -u <your_user>:<your_password> \
  -H 'X-Csrf-Token: 1' '<redirected_challengeURL_from_step_3
```

```
+query>'
```

11.3.11. GitHub

GitHub uses OAuth, and you can integrate your OpenShift Container Platform cluster to use that OAuth authentication. OAuth basically facilitates a token exchange flow.

Configuring GitHub authentication allows users to log in to OpenShift Container Platform with their GitHub credentials. To prevent anyone with any GitHub user ID from logging in to your OpenShift Container Platform cluster, you can restrict access to only those in specific GitHub organizations.

11.3.11.1. Registering the application on GitHub

1. On GitHub, click [Settings](#) → [OAuth applications](#) → [Developer applications](#) → [Register an application](#) to navigate to the page for a new OAuth application.
2. Type an application name. For example: **My OpenShift Install**
3. Type a homepage URL. For example: <https://myapiserver.com:8443>
4. Optionally, type an application description.
5. Type the authorization callback URL, where the end of the URL contains the identity provider **name** (defined in the **identityProviders** stanza of the [master configuration file](#), which you configure in the next section of this topic):

```
<apiserver>/oauth2callback/<identityProviderName>
```

For example:

```
https://myapiserver.com:8443/oauth2callback/github/
```

6. Click **Register application**. GitHub provides a Client ID and a Client Secret. Keep this window open so you can copy these values and paste them into the master configuration file.

11.3.11.2. Configuring authentication on the master

1. If you have:
 - Already completed the installation of Openshift, then copy the **/etc/origin/master/master-config.yaml** file into a new directory; for example:

```
$ cd /etc/origin/master
$ mkdir githubconfig; cp master-config.yaml githubconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

For example:

■

```
$ openshift start master --public-
master=https://myapiserver.com:8443 --write-config=githubconfig
```



NOTE

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.



NOTE

Using **openshift start master** on its own would auto-detect host names, but GitHub must be able to redirect to the exact host name that you specified when registering the application. For this reason, you cannot auto-detect the ID because it might redirect to the wrong address. Instead, you must specify the hostname that web browsers use to interact with your OpenShift Container Platform cluster.

2. Edit the new **master-config.yaml** file's **identityProviders** stanza, and copy the example **GitHubIdentityProvider** configuration and paste it to replace the existing stanza:

```
oauthConfig:
  ...
  identityProviders:
  - name: github ①
    challenge: false ②
    login: true ③
    mappingMethod: claim ④
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      clientId: ... ⑤
      clientSecret: ... ⑥
      organizations: ⑦
      - myorganization1
      - myorganization2
      teams: ⑧
      - myorganization1/team-a
      - myorganization2/team-b
```

- ① This provider name is prefixed to the GitHub numeric user ID to form an identity name. It is also used to build the callback URL.
- ② **GitHubIdentityProvider** cannot be used to send **WWW-Authenticate** challenges.
- ③ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to GitHub to log in.
- ④ Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- ⑤

The client ID of a [registered GitHub OAuth application](#). The application must be configured with a callback URL of `<master>/oauth2callback/<identityProviderName>`.

- 6 The client secret issued by GitHub. This value may also be provided in an [environment variable, external file, or encrypted file](#).
- 7 Optional list of organizations. If specified, only GitHub users that are members of at least one of the listed organizations will be allowed to log in. If the GitHub OAuth application configured in **clientId** is not owned by the organization, an organization owner must grant third-party access in order to use this option. This can be done during the first GitHub login by the organization's administrator, or from the GitHub organization settings. Cannot be used in combination with the **teams** field.
- 8 Optional list of teams. If specified, only GitHub users that are members of at least one of the listed teams will be allowed to log in. If the GitHub OAuth application configured in **clientId** is not owned by the team's organization, an organization owner must grant third-party access in order to use this option. This can be done during the first GitHub login by the organization's administrator, or from the GitHub organization settings. Cannot be used in combination with the **organizations** field.

3. Make the following modifications to the **identityProviders** stanza:

- a. Change the provider **name** to match the callback URL you configured on GitHub. For example, if you defined the callback URL as <https://myapiserver.com:8443/oauth2callback/github/> then the **name** must be **github**.
- b. Change **clientId** to the Client ID from GitHub that you [registered previously](#).
- c. Change **clientSecret** to the Client Secret from GitHub that you [registered previously](#).
- d. Change **organizations** or **teams** to include a list of one or more GitHub organizations or teams to which a user must have membership in order to authenticate. If specified, only GitHub users that are members of at least one of the listed organizations or teams will be allowed to log in. If this is not specified, then any person with a valid GitHub account can log in.

4. Save your changes and close the file.

5. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their GitHub credentials. On their first login, the user must click **authorize application** to permit GitHub to use their user name, password, and organization membership with OpenShift Container Platform. The user is then redirected back to the web console.

11.3.11.3. Creating users with GitHub authentication

You do not create users in OpenShift Container Platform when integrating with an external authentication provider, such as, in this case, GitHub. GitHub is the system of record, meaning that users are defined by GitHub, and any user belonging to a specified organization can log in.

To add a user to OpenShift Container Platform, you must add that user to an approved organization on GitHub, and if required create a new GitHub account for the user.

11.3.11.4. Verifying users

Once one or more users have logged in, you can run `oc get users` to view a list of users and verify that users were created successfully:

Example 11.6. Output of `oc get users` command

```
$ oc get users
NAME                UID                                FULL NAME
IDENTITIES
bobsmith            433b5641-066f-11e6-a6d8-acfc32c1ca87  Bob Smith
github:873654 1
```

- ¹ Identities in OpenShift Container Platform are comprised of the identity provider name and GitHub's internal numeric user ID. This way, if a user changes their GitHub user name or e-mail they can still log in to OpenShift Container Platform instead of relying on the credentials attached to the GitHub account. This creates a stable login.

From here, you might want to learn how to [control user roles](#).

11.3.12. GitLab

Set `GitLabIdentityProvider` in the `identityProviders` stanza to use [GitLab.com](#) or any other GitLab instance as an identity provider, using the [OAuth integration](#). The OAuth provider feature requires GitLab version 7.7.0 or higher.



NOTE

Using GitLab as an identity provider requires users to get a token using `<master>/oauth/token/request` to use with command-line tools.

Example 11.7. Master Configuration Using `GitLabIdentityProvider`

```
oauthConfig:
...
identityProviders:
- name: gitlab 1
  challenge: true 2
  login: true 3
  mappingMethod: claim 4
  provider:
    apiVersion: v1
    kind: GitLabIdentityProvider
    url: ... 5
    clientID: ... 6
    clientSecret: ... 7
    ca: ... 8
```


- 1 This provider name is prefixed to the GitLab numeric user ID to form an identity name. It is also used to build the callback URL.
- 2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider. This uses the [Resource Owner Password Credentials](#) grant flow to obtain an access token from GitLab.
- 3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to GitLab to log in.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 The host URL of a GitLab OAuth provider. This could either be **https://gitlab.com/** or any other self hosted instance of GitLab.
- 6 The client ID of a [registered GitLab OAuth application](#). The application must be configured with a callback URL of **<master>/oauth2callback/<identityProviderName>**.
- 7 The client secret issued by GitLab. This value may also be provided in an [environment variable, external file, or encrypted file](#).
- 8 CA is an optional trusted certificate authority bundle to use when making requests to the GitLab instance. If empty, the default system roots are used.

11.3.13. Google

Set **GoogleIdentityProvider** in the **identityProviders** stanza to use Google as an identity provider, using [Google's OpenID Connect integration](#).



NOTE

Using Google as an identity provider requires users to get a token using **<master>/oauth/token/request** to use with command-line tools.



WARNING

Using Google as an identity provider allows any Google user to authenticate to your server. You can limit authentication to members of a specific hosted domain with the **hostedDomain** configuration attribute, as shown below.

Example 11.8. Master Configuration Using GoogleIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
    - name: google 1
```

```

challenge: false 2
login: true 3
mappingMethod: claim 4
provider:
  apiVersion: v1
  kind: GoogleIdentityProvider
  clientId: ... 5
  clientSecret: ... 6
  hostedDomain: "" 7

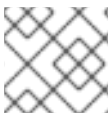
```

- 1 This provider name is prefixed to the Google numeric user ID to form an identity name. It is also used to build the redirect URL.
- 2 **GoogleIdentityProvider** cannot be used to send **WWW-Authenticate** challenges.
- 3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to Google to log in.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 The client ID of a [registered Google project](#). The project must be configured with a redirect URI of `<master>/oauth2callback/<identityProviderName>`.
- 6 The client secret issued by Google. This value may also be provided in an [environment variable](#), [external file](#), or [encrypted file](#).
- 7 Optional [hosted domain](#) to restrict sign-in accounts to. If empty, any Google account is allowed to authenticate.

11.3.14. OpenID connect

Set **OpenIDIdentityProvider** in the **identityProviders** stanza to integrate with an OpenID Connect identity provider using an [Authorization Code Flow](#).

You can [configure Red Hat Single Sign-On](#) as an OpenID Connect identity provider for OpenShift Container Platform.



NOTE

ID Token and **UserInfo** decryptions are not supported.

By default, the **openid** scope is requested. If required, extra scopes can be specified in the **extraScopes** field.

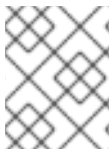
Claims are read from the JWT **id_token** returned from the OpenID identity provider and, if specified, from the JSON returned by the **UserInfo** URL.

At least one claim must be configured to use as the user's identity. The standard identity claim is **sub**.

You can also indicate which claims to use as the user's preferred user name, display name, and email address. If multiple claims are specified, the first one with a non-empty value is used. The standard claims are:

sub	Short for "subject identifier." The remote identity for the user at the issuer.
preferred_username	The preferred user name when provisioning a user. A shorthand name that the user wants to be referred to as, such as janedoe . Typically a value that corresponding to the user's login or username in the authentication system, such as username or email.
email	Email address.
name	Display name.

See the [OpenID claims documentation](#) for more information.



NOTE

Using an OpenID Connect identity provider requires users to get a token using **<master>/oauth/token/request** to use with command-line tools.

Standard Master Configuration Using OpenIDIdentityProvider

```

oauthConfig:
  ...
  identityProviders:
    - name: my_openid_connect ❶
      challenge: true ❷
      login: true ❸
      mappingMethod: claim ❹
      provider:
        apiVersion: v1
        kind: OpenIDIdentityProvider
        clientID: ... ❺
        clientSecret: ... ❻
        claims:
          id: ❼
          - sub
          preferredUsername:
            - preferred_username
          name:
            - name
          email:
            - email
        urls:
          authorize: https://myidp.example.com/oauth2/authorize ❽
          token: https://myidp.example.com/oauth2/token ❾

```

- ❶ This provider name is prefixed to the value of the identity claim to form an identity name. It is also used to build the redirect URL.

- 2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider. This requires the OpenID provider to support the [Resource Owner Password Credentials](#) grant flow.
- 3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to the authorize URL to log in.
- 4 Controls how mappings are established between this provider's identities and user objects, [as described above](#).
- 5 The client ID of a client registered with the OpenID provider. The client must be allowed to redirect to `<master>/oauth2callback/<identityProviderName>`.
- 6 The client secret. This value may also be provided in an [environment variable, external file, or encrypted file](#).
- 7 List of claims to use as the identity. First non-empty claim is used. At least one claim is required. If none of the listed claims have a value, authentication fails. For example, this uses the value of the **sub** claim in the returned **id_token** as the user's identity.
- 8 [Authorization Endpoint](#) described in the OpenID spec. Must use **https**.
- 9 [Token Endpoint](#) described in the OpenID spec. Must use **https**.

A custom certificate bundle, extra scopes, extra authorization request parameters, and **userInfo** URL can also be specified:

Example 11.9. Full Master Configuration Using OpenIDIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ...
      clientSecret: ...
      ca: my-openid-ca-bundle.crt 1
      extraScopes: 2
      - email
      - profile
      extraAuthorizeParameters: 3
      include_granted_scopes: "true"
      claims:
        id: 4
        - custom_id_claim
        - sub
      preferredUsername: 5
      - preferred_username
      - email
```

```

    name: ❸
    - nickname
    - given_name
    - name
    email: ❷
    - custom_email_claim
    - email
  urls:
    authorize: https://myidp.example.com/oauth2/authorize
    token: https://myidp.example.com/oauth2/token
    userInfo: https://myidp.example.com/oauth2/userinfo ❹

```

- ❶ Certificate bundle to use to validate server certificates for the configured URLs. If empty, system trusted roots are used.
- ❷ Optional list of scopes to request, in addition to the **openid** scope, during the authorization token request.
- ❸ Optional map of extra parameters to add to the authorization token request.
- ❹ List of claims to use as the identity. First non-empty claim is used. At least one claim is required. If none of the listed claims have a value, authentication fails.
- ❺ List of claims to use as the preferred user name when provisioning a user for this identity. First non-empty claim is used.
- ❻ List of claims to use as the display name. First non-empty claim is used.
- ❼ List of claims to use as the email address. First non-empty claim is used.
- ❽ [UserInfo Endpoint](#) described in the OpenID spec. Must use **https**.

11.4. TOKEN OPTIONS

The OAuth server generates two kinds of tokens:

Access tokens	Longer-lived tokens that grant access to the API.
Authorize codes	Short-lived tokens whose only use is to be exchanged for an access token.

Use the **tokenConfig** stanza to set token options:

Example 11.10. Master Configuration Token Options

```

oauthConfig:
  ...
tokenConfig:
  accessTokenMaxAgeSeconds: 86400 ❶
  authorizeTokenMaxAgeSeconds: 300 ❷

```

- 1 Set **accessTokenMaxAgeSeconds** to control the lifetime of access tokens. The default lifetime is 24 hours.
- 2 Set **authorizeTokenMaxAgeSeconds** to control the lifetime of authorize codes. The default lifetime is five minutes.

**NOTE**

You can override the **accessTokenMaxAgeSeconds** value [through an OAuthClient object definition](#).

11.5. GRANT OPTIONS

When the OAuth server receives token requests for a client to which the user has not previously granted permission, the action that the OAuth server takes is dependent on the OAuth client's grant strategy.

When the OAuth client requesting token does not provide its own grant strategy, the server-wide default strategy is used. To configure the default strategy, set the **method** value in the **grantConfig** stanza. Valid values for **method** are:

auto	Auto-approve the grant and retry the request.
prompt	Prompt the user to approve or deny the grant.
deny	Auto-deny the grant and return a failure error to the client.

Example 11.11. Master Configuration Grant Options

```
oauthConfig:
  ...
  grantConfig:
    method: auto
```

11.6. SESSION OPTIONS

The OAuth server uses a signed and encrypted cookie-based session during login and redirect flows.

Use the **sessionConfig** stanza to set session options:

Example 11.12. Master Configuration Session Options

```
oauthConfig:
  ...
  sessionConfig:
```

```

sessionMaxAgeSeconds: 300 ❶
sessionName: ssn ❷
sessionSecretsFile: "... " ❸

```

- ❶ Controls the maximum age of a session; sessions auto-expire once a token request is complete. If [auto-grant](#) is not enabled, sessions must last as long as the user is expected to take to approve or reject a client authorization request.
- ❷ Name of the cookie used to store the session.
- ❸ File name containing serialized **SessionSecrets** object. If empty, a random signing and encryption secret is generated at each server start.

If no **sessionSecretsFile** is specified, a random signing and encryption secret is generated at each start of the master server. This means that any logins in progress will have their sessions invalidated if the master is restarted. It also means they will not be able to decode sessions generated by one of the other masters.

To specify the signing and encryption secret to use, specify a **sessionSecretsFile**. This allows you separate secret values from the configuration file and keep the configuration file distributable, for example for debugging purposes.

Multiple secrets can be specified in the **sessionSecretsFile** to enable rotation. New sessions are signed and encrypted using the first secret in the list. Existing sessions are decrypted and authenticated by each secret until one succeeds.

Example 11.13. Session Secret Configuration:

```

apiVersion: v1
kind: SessionSecrets
secrets: ❶
- authentication: "... " ❷
  encryption: "... " ❸
- authentication: "... "
  encryption: "... "
...

```

- ❶ List of secrets used to authenticate and encrypt cookie sessions. At least one secret must be specified. Each secret must set an authentication and encryption secret.
- ❷ Signing secret, used to authenticate sessions using HMAC. Recommended to use a secret with 32 or 64 bytes.
- ❸ Encrypting secret, used to encrypt sessions. Must be 16, 24, or 32 characters long, to select AES-128, AES-192, or AES-256.

11.7. PREVENTING CLI VERSION MISMATCH WITH USER AGENT

OpenShift Container Platform implements a user agent that can be used to prevent an application developer's CLI accessing the OpenShift Container Platform API.

User agents for the OpenShift Container Platform CLI are constructed from a set of values within OpenShift Container Platform:

```
<command>/<version> (<platform>/<architecture>) <client>/<git_commit>
```

So, for example, when:

- `<command>` = **oc**
- `<version>` = The client version. For example, **v3.3.0**. Requests made against the Kubernetes API at **/api** receive the Kubernetes version, while requests made against the OpenShift Container Platform API at **/oapi** receive the OpenShift Container Platform version (as specified by **oc version**)
- `<platform>` = **linux**
- `<architecture>` = **amd64**
- `<client>` = **openshift**, or **kubernetes** depending on if the request is made against the Kubernetes API at **/api**, or the OpenShift Container Platform API at **/oapi**
- `<git_commit>` = The Git commit of the client version (for example, **f034127**)

the user agent will be:

```
oc/v3.3.0 (linux/amd64) openshift/f034127
```

As an OpenShift Container Platform administrator, you can prevent clients from accessing the API with the **userAgentMatching** configuration setting of a master configuration. So, if a client is using a particular library or binary, they will be prevented from accessing the API.

The following user agent example denies the Kubernetes 1.2 client binary, OpenShift Origin 1.1.3 binary, and the POST and PUT **httpVerbs**:

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old. Go to
https://example.org to update it."
    deniedClients:
      - regex: '\w+/v(?:1\.1\.1|1\.0\.1) \(.+/.+\) openshift/\w{7}'
      - regex: '\w+/v(?:1\.1\.3) \(.+/.+\) openshift/\w{7}'
      httpVerbs:
        - POST
        - PUT
      - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
        - POST
        - PUT
    requiredClients: null
```

Administrators can also deny clients that do not exactly match the expected clients:

```
policyConfig:
  userAgentMatchingConfig:
```



```
defaultRejectionMessage: "Your client is too old. Go to
https://example.org to update it."
deniedClients: []
requiredClients:
- regex: '\w+/v1\..1\..3 \(.+/.+\) openshift/\w{7}'
- regex: '\w+/v1\..2\..0 \(.+/.+\) kubernetes/\w{7}'
httpVerbs:
- POST
- PUT
```



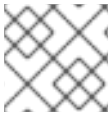
NOTE

When the client's user agent mismatches the configuration, errors occur. To ensure that mutating requests match, enforce a whitelist. Rules are mapped to specific verbs, so you can ban mutating requests while allowing non-mutating requests.

CHAPTER 12. SYNCING GROUPS WITH LDAP

12.1. OVERVIEW

As an OpenShift Container Platform administrator, you can use groups to manage users, change their permissions, and enhance collaboration. Your organization may have already created user groups and stored them in an LDAP server. OpenShift Container Platform can sync those LDAP records with internal OpenShift Container Platform records, enabling you to manage your groups in one place. OpenShift Container Platform currently supports group sync with LDAP servers using three common schemas for defining group membership: RFC 2307, Active Directory, and augmented Active Directory.



NOTE

You must have [cluster-admin privileges](#) to sync groups.

12.2. CONFIGURING LDAP SYNC

Before you can [run LDAP sync](#), you need a sync configuration file. This file contains LDAP client configuration details:

- Configuration for connecting to your LDAP server.
- Sync configuration options that are dependent on the schema used in your LDAP server.

A sync configuration file can also contain an administrator-defined list of name mappings that maps OpenShift Container Platform group names to groups in your LDAP server.

12.2.1. LDAP client configuration

LDAP client configuration

```
url: ldap://10.0.0.0:389 ❶
bindDN: cn=admin,dc=example,dc=com ❷
bindPassword: password ❸
insecure: false ❹
ca: my-ldap-ca-bundle.crt ❺
```

- ❶ The connection protocol, IP address of the LDAP server hosting your database, and the port to connect to, formatted as **scheme://host:port**.
- ❷ Optional distinguished name (DN) to use as the Bind DN. OpenShift Container Platform uses this if elevated privilege is required to retrieve entries for the sync operation.
- ❸ Optional password to use to bind. OpenShift Container Platform uses this if elevated privilege is necessary to retrieve entries for the sync operation. This value may also be provided in an [environment variable](#), [external file](#), or [encrypted file](#).
- ❹ When **true**, no TLS connection is made to the server. When **false**, secure LDAP (**ldaps://**) URLs connect using TLS, and insecure LDAP (**ldap://**) URLs are upgraded to TLS.

❺

The certificate bundle to use for validating server certificates for the configured URL. If empty, OpenShift Container Platform uses system-trusted roots. This only applies if **insecure** is set to

12.2.2. LDAP query definition

Sync configurations consist of LDAP query definitions for the entries that are required for synchronization. The specific definition of an LDAP query depends on the schema used to store membership information in the LDAP server.

LDAP query definition

```
baseDN: ou=users,dc=example,dc=com ❶
scope: sub ❷
derefAliases: never ❸
timeout: 0 ❹
filter: (objectClass=inetOrgPerson) ❺
pageSize: 0 ❻
```

- ❶ The distinguished name (DN) of the branch of the directory where all searches will start from. It is required that you specify the top of your directory tree, but you can also specify a subtree in the directory.
- ❷ The scope of the search. Valid values are **base**, **one**, or **sub**. If this is left undefined, then a scope of **sub** is assumed. Descriptions of the scope options can be found in the [table below](#).
- ❸ The behavior of the search with respect to aliases in the LDAP tree. Valid values are **never**, **search**, **base**, or **always**. If this is left undefined, then the default is to **always** dereference aliases. Descriptions of the dereferencing behaviors can be found in the [table below](#).
- ❹ The time limit allowed for the search by the client, in seconds. A value of 0 imposes no client-side limit.
- ❺ A valid LDAP search filter. If this is left undefined, then the default is **(objectClass=*)**.
- ❻ The optional maximum size of response pages from the server, measured in LDAP entries. If set to 0, no size restrictions will be made on pages of responses. Setting paging sizes is necessary when queries return more entries than the client or server allow by default.

Table 12.1. LDAP search scope options

LDAP Search Scope	Description
base	Only consider the object specified by the base DN given for the query.
one	Consider all of the objects on the same level in the tree as the base DN for the query.
sub	Consider the entire subtree rooted at the base DN given for the query.

Table 12.2. LDAP dereferencing behaviors

Dereferencing Behavior	Description
never	Never dereference any aliases found in the LDAP tree.
search	Only dereference aliases found while searching.
base	Only dereference aliases while finding the base object.
always	Always dereference all aliases found in the LDAP tree.

12.2.3. User-defined name mapping

A user-defined name mapping explicitly maps the names of OpenShift Container Platform groups to unique identifiers that find groups on your LDAP server. The mapping uses normal YAML syntax. A user-defined mapping can contain an entry for every group in your LDAP server or only a subset of those groups. If there are groups on the LDAP server that do not have a user-defined name mapping, the default behavior during sync is to use the attribute specified as the OpenShift Container Platform group's name.

User-defined name mapping

```
groupUIDNameMapping:
  "cn=group1,ou=groups,dc=example,dc=com": firstgroup
  "cn=group2,ou=groups,dc=example,dc=com": secondgroup
  "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

12.3. RUNNING LDAP SYNC

Once you have created a [sync configuration file](#), then sync can begin. OpenShift Container Platform allows administrators to perform a number of different sync types with the same server.



NOTE

By default, all group synchronization or pruning operations are dry-run, so you must set the **--confirm** flag on the **sync-groups** command in order to make changes to OpenShift Container Platform Group records.

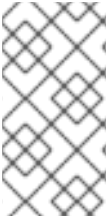
To sync all groups from the LDAP server with OpenShift Container Platform:

```
$ oc adm groups sync --sync-config=config.yaml --confirm
```

To sync all groups already in OpenShift Container Platform that correspond to groups in the LDAP server specified in the configuration file:

```
$ oc adm groups sync --type=openshift --sync-config=config.yaml --confirm
```

To sync a subset of LDAP groups with OpenShift Container Platform, you can use whitelist files, blacklist files, or both:

**NOTE**

You can use any combination of blacklist files, whitelist files, or whitelist literals. Whitelist and blacklist files must contain one unique group identifier per line, and you can include whitelist literals directly in the command itself. These guidelines apply to groups found on LDAP servers as well as groups already present in OpenShift Container Platform.

```
$ oc adm groups sync --whitelist=<whitelist_file> \
    --sync-config=config.yaml \
    --confirm
$ oc adm groups sync --blacklist=<blacklist_file> \
    --sync-config=config.yaml \
    --confirm
$ oc adm groups sync <group_unique_identifier> \
    --sync-config=config.yaml \
    --confirm
$ oc adm groups sync <group_unique_identifier> \
    --whitelist=<whitelist_file> \
    --blacklist=<blacklist_file> \
    --sync-config=config.yaml \
    --confirm
$ oc adm groups sync --type=openshift \
    --whitelist=<whitelist_file> \
    --sync-config=config.yaml \
    --confirm
```

12.4. RUNNING A GROUP PRUNING JOB

An administrator can also choose to remove groups from OpenShift Container Platform records if the records on the LDAP server that created them are no longer present. The prune job will accept the same sync configuration file and white- or black-lists as used for the sync job. More information is available in [Pruning groups](#) section.

12.5. SYNC EXAMPLES

This section contains examples for the [RFC 2307](#), [Active Directory](#), and [augmented Active Directory](#) schemas. All of the following examples synchronize a group named **admins** that has two members: **Jane** and **Jim**. Each example explains:

- How the group and users are added to the LDAP server.
- What the LDAP sync configuration file looks like.
- What the resulting group record in OpenShift Container Platform will be after synchronization.

**NOTE**

These examples assume that all users are direct members of their respective groups. Specifically, no groups have other groups as members. See [Nested Membership Sync Example](#) for information on how to sync nested groups.

12.5.1. Syncing groups by using RFC 2307 schema

In the RFC 2307 schema, both users (Jane and Jim) and groups exist on the LDAP server as first-class entries, and group membership is stored in attributes on the group. The following snippet of **ldif** defines the users and group for this schema:

LDAP entries that use RFC 2307 schema: *rfc2307.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com ❶
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com ❷
member: cn=Jim,ou=users,dc=example,dc=com
```

- ❶ The group is a first-class entry in the LDAP server.
- ❷ Members of a group are listed with an identifying reference as attributes on the group.

To sync this group, you must first create the configuration file. The RFC 2307 schema requires you to provide an LDAP query definition for both user and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships:

**NOTE**

If using user-defined name mappings, your [configuration file](#) will differ.

LDAP sync configuration that uses RFC 2307 schema: *rfc2307_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389 ❶
insecure: false ❷
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ❸
  groupNameAttributes: [ cn ] ❹
  groupMembershipAttributes: [ member ] ❺
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  userUIDAttribute: dn ❻
  userNameAttributes: [ mail ] ❼
  tolerateMemberNotFoundErrors: false
  tolerateMemberOutOfScopeErrors: false
```

- ❶ The IP address and host of the LDAP server where this group's record is stored.
- ❷ When **true**, no TLS connection is made to the server. When **false**, secure LDAP (**ldaps://**) URLs connect using TLS, and insecure LDAP (**ldap://**) URLs are upgraded to TLS.
- ❸ The attribute that uniquely identifies a group on the LDAP server. You cannot specify **groupsQuery** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the [whitelist / blacklist method](#).
- ❹ The attribute to use as the name of the group.
- ❺ The attribute on the group that stores the membership information.
- ❻ The attribute that uniquely identifies a user on the LDAP server. You cannot specify **usersQuery** filters when using DN for userUIDAttribute. For fine-grained filtering, use the [whitelist / blacklist method](#).
- ❼ The attribute to use as the name of the user in the OpenShift Container Platform group record.

To run sync with the *rfc2307_config.yaml* file:

```
$ oc adm groups sync --sync-config=rfc2307_config.yaml --confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift Container Platform group created by using the *rfc2307_config.yaml* file

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ❶
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com ❷
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ❸
  creationTimestamp:
  name: admins ❹
users: ❺
- jane.smith@example.com
- jim.adams@example.com

```

- ❶ The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
- ❷ The unique identifier for the group on the LDAP server.
- ❸ The IP address and host of the LDAP server where this group's record is stored.
- ❹ The name of the group as specified by the sync file.
- ❺ The users that are members of the group, named as specified by the sync file.

12.5.1.1. RFC2307 with user-defined name mappings

When syncing groups with user-defined name mappings, the configuration file changes to contain these mappings as shown below.

LDAP sync configuration that uses RFC 2307 schema with user-defined name mappings:
rfc2307_config_user_defined.yaml

```

kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators ❶
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ❷
  groupNameAttributes: [ cn ] ❸
  groupMembershipAttributes: [ member ]
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  userUIDAttribute: dn ❹

```



```

userNameAttributes: [ mail ]
tolerateMemberNotFoundErrors: false
tolerateMemberOutOfScopeErrors: false

```

- 1 The user-defined name mapping.
- 2 The unique identifier attribute that is used for the keys in the user-defined name mapping. You cannot specify **groupsQuery** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the [whitelist / blacklist method](#).
- 3 The attribute to name OpenShift Container Platform groups with if their unique identifier is not in the user-defined name mapping.
- 4 The attribute that uniquely identifies a user on the LDAP server. You cannot specify **usersQuery** filters when using DN for userUIDAttribute. For fine-grained filtering, use the [whitelist / blacklist method](#).

To run sync with the *rfc2307_config_user_defined.yaml* file:

```

$ oc adm groups sync --sync-config=rfc2307_config_user_defined.yaml --confirm

```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift Container Platform group created by using the *rfc2307_config_user_defined.yaml* file

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
    name: Administrators 1
users:
- jane.smith@example.com
- jim.adams@example.com

```

- 1 The name of the group as specified by the user-defined name mapping.

12.5.2. Syncing groups by using RFC 2307 with user-defined error tolerances

By default, if the groups being synced contain members whose entries are outside of the scope defined in the member query, the group sync fails with an error:

```

Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with dn="<user-dn>" would search outside of the base dn specified (dn="
<base-dn>")".

```

This often indicates a mis-configured **baseDN** in the **usersQuery** field. However, in cases where the **baseDN** intentionally does not contain some of the members of the group, setting **tolerateMemberOutOfScopeErrors: true** allows the group sync to continue. Out of scope members will be ignored.

Similarly, when the group sync process fails to locate a member for a group, it fails outright with errors:

```
Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with base dn="<user-dn>" refers to a non-existent entry".
```

```
Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with base dn="<user-dn>" and filter "<filter>" did not return any results".
```

This often indicates a mis-configured **usersQuery** field. However, in cases where the group contains member entries that are known to be missing, setting **tolerateMemberNotFoundErrors: true** allows the group sync to continue. Problematic members will be ignored.



WARNING

Enabling error tolerances for the LDAP group sync causes the sync process to ignore problematic member entries. If the LDAP group sync is not configured correctly, this could result in synced OpenShift Container Platform groups missing members.

LDAP entries that use RFC 2307 schema with problematic group membership: *rfc2307_problematic_users.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
```

```

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
member: cn=INVALID,ou=users,dc=example,dc=com ❶
member: cn=Jim,ou=OUTOFSCOPE,dc=example,dc=com ❷

```

- ❶ A member that does not exist on the LDAP server.
- ❷ A member that may exist, but is not under the **baseDN** in the user query for the sync job.

In order to tolerate the errors in the above example, the following additions to your sync configuration file must be made:

LDAP sync configuration that uses RFC 2307 schema tolerating errors: *rfc2307_config_tolerating.yaml*

```

kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
  groupUIDAttribute: dn
  groupNameAttributes: [ cn ]
  groupMembershipAttributes: [ member ]
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
  userUIDAttribute: dn ❶
  userNameAttributes: [ mail ]
  tolerateMemberNotFoundErrors: true ❷
  tolerateMemberOutOfScopeErrors: true ❸

```

- ❷ When **true**, the sync job tolerates groups for which some members were not found, and members whose LDAP entries are not found are ignored. The default behavior for the sync job is to fail if a member of a group is not found.
- ❸ When **true**, the sync job tolerates groups for which some members are outside the user scope given in the **usersQuery** base DN, and members outside the member query scope are ignored. The default behavior for the sync job is to fail if a member of a group is out of scope.

❶

The attribute that uniquely identifies a user on the LDAP server. You cannot specify **usersQuery** filters when using DN for `userUIDAttribute`. For fine-grained filtering, use the [whitelist](#) / [blacklist](#)

To run sync with the *rfc2307_config_tolerating.yaml* file:

```
$ oc adm groups sync --sync-config=rfc2307_config_tolerating.yaml --confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift Container Platform group created by using the *rfc2307_config.yaml* file

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: admins
users: ①
- jane.smith@example.com
- jim.adams@example.com
```

- ① The users that are members of the group, as specified by the sync file. Members for which lookup encountered tolerated errors are absent.

12.5.3. Syncing groups by using Active Directory

In the Active Directory schema, both users (Jane and Jim) exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of **ldif** defines the users and group for this schema:

LDAP entries that use Active Directory schema: *active_directory.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: admins ①

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
```

```
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: admins
```

- 1 The user's group memberships are listed as attributes on the user, and the group does not exist as an entry on the server. The **memberOf** attribute does not have to be a literal attribute on the user; in some LDAP servers, it is created during search and returned to the client, but not committed to the database.

To sync this group, you must first create the configuration file. The Active Directory schema requires you to provide an LDAP query definition for user entries, as well as the attributes to represent them with in the internal OpenShift Container Platform group records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, but define the name of the group by the name of the group on the LDAP server. The following configuration file creates these relationships:

LDAP sync configuration that uses Active Directory schema: *active_directory_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
activeDirectory:
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
    pageSize: 0
  userNameAttributes: [ mail ] 1
  groupMembershipAttributes: [ memberOf ] 2
```

- 1 The attribute to use as the name of the user in the OpenShift Container Platform group record.
- 2 The attribute on the user that stores the membership information.

To run sync with the *active_directory_config.yaml* file:

```
$ oc adm groups sync --sync-config=active_directory_config.yaml --confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift Container Platform group created by using the *active_directory_config.yaml* file

```
apiVersion: user.openshift.io/v1
```

```

kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ❶
    openshift.io/ldap.uid: admins ❷
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ❸
  creationTimestamp:
    name: admins ❹
users: ❺
- jane.smith@example.com
- jim.adams@example.com

```

- ❶ The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
- ❷ The unique identifier for the group on the LDAP server.
- ❸ The IP address and host of the LDAP server where this group's record is stored.
- ❹ The name of the group as listed in the LDAP server.
- ❺ The users that are members of the group, named as specified by the sync file.

12.5.4. Syncing groups by using augmented Active Directory

In the augmented Active Directory schema, both users (Jane and Jim) and groups exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user. The following snippet of `ldif` defines the users and group for this schema:

LDAP entries that use augmented Active Directory schema: *augmented_active_directory.ldif*

```

dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com ❶

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams

```

```

displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com ❷
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com

```

- ❶ The user's group memberships are listed as attributes on the user.
- ❷ The group is a first-class entry on the LDAP server.

To sync this group, you must first create the configuration file. The augmented Active Directory schema requires you to provide an LDAP query definition for both user entries and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform group records.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships.

LDAP sync configuration that uses augmented Active Directory schema: *augmented_active_directory_config.yaml*

```

kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ❶
  groupNameAttributes: [ cn ] ❷
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
    pageSize: 0
  userNameAttributes: [ mail ] ❸
  groupMembershipAttributes: [ memberOf ] ❹

```

- 1 The attribute that uniquely identifies a group on the LDAP server. You cannot specify **groupsQuery** filters when using DN for groupUIDAttribute. For fine-grained filtering, use the [whitelist / blacklist method](#).
- 2 The attribute to use as the name of the group.
- 3 The attribute to use as the name of the user in the OpenShift Container Platform group record.
- 4 The attribute on the user that stores the membership information.

To run sync with the ***augmented_active_directory_config.yaml*** file:

```
$ oc adm groups sync --sync-config=augmented_active_directory_config.yaml
--confirm
```

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift group created by using the ***augmented_active_directory_config.yaml*** file

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
    name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

- 1 The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
- 2 The unique identifier for the group on the LDAP server.
- 3 The IP address and host of the LDAP server where this group's record is stored.
- 4 The name of the group as specified by the sync file.
- 5 The users that are members of the group, named as specified by the sync file.

12.6. NESTED MEMBERSHIP SYNC EXAMPLE

Groups in OpenShift Container Platform do not nest. The LDAP server must flatten group membership before the data can be consumed. Microsoft's Active Directory Server supports this feature via the [LDAP_MATCHING_RULE_IN_CHAIN](#) rule, which has the OID **1.2.840.113556.1.4.1941**. Furthermore, only explicitly [whitelisted](#) groups can be synced when using this matching rule.

This section has an example for the augmented Active Directory schema, which synchronizes a group named **admins** that has one user **Jane** and one group **otheradmins** as members. The **otheradmins** group has one user member: **Jim**. This example explains:

- How the group and users are added to the LDAP server.
- What the LDAP sync configuration file looks like.
- What the resulting group record in OpenShift Container Platform will be after synchronization.

In the augmented Active Directory schema, both users (**Jane** and **Jim**) and groups exist in the LDAP server as first-class entries, and group membership is stored in attributes on the user or the group. The following snippet of **ldif** defines the users and groups for this schema:

LDAP entries that use augmented Active Directory schema with nested members:
augmented_active_directory_nested.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com 1

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com 2

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com 3
objectClass: group
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=otheradmins,ou=groups,dc=example,dc=com

dn: cn=otheradmins,ou=groups,dc=example,dc=com 4
objectClass: group
cn: otheradmins
owner: cn=admin,dc=example,dc=com
```

```
description: Other System Administrators
memberOf: cn=admins,ou=groups,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
```

5 6

1 2 5 The user's and group's memberships are listed as attributes on the object.

3 4 The groups are first-class entries on the LDAP server.

6 The **otheradmins** group is a member of the **admins** group.

To sync nested groups with Active Directory, you must provide an LDAP query definition for both user entries and group entries, as well as the attributes with which to represent them in the internal OpenShift Container Platform group records. Furthermore, certain changes are required in this configuration:

- The **oc adm groups sync** command must explicitly [whitelist](#) groups.
- The user's **groupMembershipAttributes** must include **"memberOf:1.2.840.113556.1.4.1941:"** to comply with the [LDAP_MATCHING_RULE_IN_CHAIN](#) rule.
- The **groupUIDAttribute** must be set to **dn**.
- The **groupsQuery**:
 - Must not set **filter**.
 - Must set a valid **derefAliases**.
 - Should not set **baseDN** as that value is ignored.
 - Should not set **scope** as that value is ignored.

For clarity, the group you create in OpenShift Container Platform should use attributes other than the distinguished name whenever possible for user- or administrator-facing fields. For example, identify the users of an OpenShift Container Platform group by their e-mail, and use the name of the group as the common name. The following configuration file creates these relationships:

LDAP sync configuration that uses augmented Active Directory schema with nested members: *augmented_active_directory_config_nested.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery: 1
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn 2
  groupNameAttributes: [ cn ] 3
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
```

```

    pageSize: 0
    userNameAttributes: [ mail ] ④
    groupMembershipAttributes: [ "memberOf:1.2.840.113556.1.4.1941:" ] ⑤

```

- ① **groupsQuery** filters cannot be specified. The **groupsQuery** base DN and scope values are ignored. **groupsQuery** must set a valid **derefAliases**.
- ② The attribute that uniquely identifies a group on the LDAP server. It must be set to **dn**.
- ③ The attribute to use as the name of the group.
- ④ The attribute to use as the name of the user in the OpenShift Container Platform group record. **mail** or **sAMAccountName** are preferred choices in most installations.
- ⑤ The attribute on the user that stores the membership information. Note the use of [LDAP_MATCHING_RULE_IN_CHAIN](#).

To run sync with the ***augmented_active_directory_config_nested.yaml*** file:

```

$ oc adm groups sync \
  'cn=admins,ou=groups,dc=example,dc=com' \
  --sync-config=augmented_active_directory_config_nested.yaml \
  --confirm

```



NOTE

You must explicitly [whitelist](#) the **cn=admins,ou=groups,dc=example,dc=com** group.

OpenShift Container Platform creates the following group record as a result of the above sync operation:

OpenShift group created by using the ***augmented_active_directory_config_nested.yaml*** file

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ①
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com ②
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ③
  creationTimestamp:
  name: admins ④
users: ⑤
- jane.smith@example.com
- jim.adams@example.com

```

- ① The last time this OpenShift Container Platform group was synchronized with the LDAP server, in ISO 6801 format.
- ② The unique identifier for the group on the LDAP server.
- ③ The IP address and host of the LDAP server where this group's record is stored.

- 4 The name of the group as specified by the sync file.
- 5 The users that are members of the group, named as specified by the sync file. Note that members of nested groups are included since the group membership was flattened by the Microsoft Active Directory Server.

12.7. LDAP SYNC CONFIGURATION SPECIFICATION

The object specification for the configuration file is below. Note that the different schema objects have different fields. For example, [v1.ActiveDirectoryConfig](#) has no **groupsQuery** field whereas [v1.RFC2307Config](#) and [v1.AugmentedActiveDirectoryConfig](#) both do.



IMPORTANT

There is no support for binary attributes. All attribute data coming from the LDAP server must be in the format of a UTF-8 encoded string. For example, never use a binary attribute, such as **objectGUID**, as an ID attribute. You must use string attributes, such as **sAMAccountName** or **userPrincipalName**, instead.

12.7.1. v1.LDAPSyncConfig

LDAPSyncConfig holds the necessary configuration options to define an LDAP group sync.

Name	Description	Schema
kind	String value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#types-kinds	string
apiVersion	Defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#resources	string
url	Host is the scheme, host and port of the LDAP server to connect to: scheme://host:port	string

Name	Description	Schema
bindDN	Optional DN to bind to the LDAP server with.	string
bindPassword	Optional password to bind with during the search phase.	v1.StringSource
insecure	If true , indicates the connection should not use TLS. Cannot be set to true with a URL scheme of ldaps:// . If false , ldaps:// URLs connect using TLS, and ldap:// URLs are upgraded to a TLS connection using StartTLS as specified in https://tools.ietf.org/html/rfc2830 .	boolean
ca	Optional trusted certificate authority bundle to use when making requests to the server. If empty, the default system roots are used.	string
groupUIDNameMapping	Optional direct mapping of LDAP group UIDs to OpenShift Container Platform group names.	object
rfc2307	Holds the configuration for extracting data from an LDAP server set up in a fashion similar to RFC2307: first-class group and user entries, with group membership determined by a multi-valued attribute on the group entry listing its members.	v1.RFC2307Config
activeDirectory	Holds the configuration for extracting data from an LDAP server set up in a fashion similar to that used in Active Directory: first-class user entries, with group membership determined by a multi-valued attribute on members listing groups they are a member of.	v1.ActiveDirectoryConfig

Name	Description	Schema
augmentedActiveDirectory	Holds the configuration for extracting data from an LDAP server set up in a fashion similar to that used in Active Directory as described above, with one addition: first-class group entries exist and are used to hold metadata but not group membership.	v1.AugmentedActiveDirectoryCon fig

12.7.2. v1.StringSource

StringSource allows specifying a string inline, or externally via environment variable or file. When it contains only a string value, it marshals to a simple JSON string.

Name	Description	Schema
value	Specifies the cleartext value, or an encrypted value if keyFile is specified.	string
env	Specifies an environment variable containing the cleartext value, or an encrypted value if the keyFile is specified.	string
file	References a file containing the cleartext value, or an encrypted value if a keyFile is specified.	string
keyFile	References a file containing the key to use to decrypt the value.	string

12.7.3. v1.LDAPQuery

LDAPQuery holds the options necessary to build an LDAP query.

Name	Description	Schema
baseDN	DN of the branch of the directory where all searches should start from.	string

Name	Description	Schema
scope	The (optional) scope of the search. Can be base (only the base object), one (all objects on the base level), sub (the entire subtree). Defaults to sub if not set.	string
derefAliases	The (optional) behavior of the search with regards to aliases. Can be never (never dereference aliases), search (only dereference in searching), base (only dereference in finding the base object), always (always dereference). Defaults to always if not set.	string
timeout	Holds the limit of time in seconds that any request to the server can remain outstanding before the wait for a response is given up. If this is 0 , no client-side limit is imposed.	integer
filter	A valid LDAP search filter that retrieves all relevant entries from the LDAP server with the base DN.	string
pageSize	Maximum preferred page size, measured in LDAP entries. A page size of 0 means no paging will be done.	integer

12.7.4. v1.RFC2307Config

RFC2307Config holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the RFC2307 schema.

Name	Description	Schema
groupsQuery	Holds the template for an LDAP query that returns group entries.	v1.LDAPQuery

Name	Description	Schema
groupUIDAttribute	Defines which attribute on an LDAP group entry will be interpreted as its unique identifier. (ldapGroupUID)	string
groupNameAttributes	Defines which attributes on an LDAP group entry will be interpreted as its name to use for an OpenShift Container Platform group.	string array
groupMembershipAttributes	Defines which attributes on an LDAP group entry will be interpreted as its members. The values contained in those attributes must be queryable by your UserUIDAttribute .	string array
usersQuery	Holds the template for an LDAP query that returns user entries.	v1.LDAPQuery
userUIDAttribute	Defines which attribute on an LDAP user entry will be interpreted as its unique identifier. It must correspond to values that will be found from the GroupMembershipAttributes .	string
userNameAttributes	Defines which attributes on an LDAP user entry will be used, in order, as its OpenShift Container Platform user name. The first attribute with a non-empty value is used. This should match your PreferredUsername setting for your LDAPPasswordIdentityProvider . The attribute to use as the name of the user in the OpenShift Container Platform group record. mail or sAMAccountName are preferred choices in most installations.	string array

Name	Description	Schema
tolerateMemberNotFoundErrors	Determines the behavior of the LDAP sync job when missing user entries are encountered. If true , an LDAP query for users that does not find any will be tolerated and an only and error will be logged. If false , the LDAP sync job will fail if a query for users doesn't find any. The default value is 'false'. Misconfigured LDAP sync jobs with this flag set to 'true' can cause group membership to be removed, so it is recommended to use this flag with caution.	boolean
tolerateMemberOutOfScopeErrors	Determines the behavior of the LDAP sync job when out-of-scope user entries are encountered. If true , an LDAP query for a user that falls outside of the base DN given for the all user query will be tolerated and only an error will be logged. If false , the LDAP sync job will fail if a user query would search outside of the base DN specified by the all user query. Misconfigured LDAP sync jobs with this flag set to true can result in groups missing users, so it is recommended to use this flag with caution.	boolean

12.7.5. v1.ActiveDirectoryConfig

ActiveDirectoryConfig holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the Active Directory schema.

Name	Description	Schema
usersQuery	Holds the template for an LDAP query that returns user entries.	v1.LDAPQuery

Name	Description	Schema
userNameAttributes	Defines which attributes on an LDAP user entry will be interpreted as its OpenShift Container Platform user name. The attribute to use as the name of the user in the OpenShift Container Platform group record. mail or sAMAccountName are preferred choices in most installations.	string array
groupMembershipAttributes	Defines which attributes on an LDAP user entry will be interpreted as the groups it is a member of.	string array

12.7.6. v1.AugmentedActiveDirectoryConfig

AugmentedActiveDirectoryConfig holds the necessary configuration options to define how an LDAP group sync interacts with an LDAP server using the augmented Active Directory schema.

Name	Description	Schema
usersQuery	Holds the template for an LDAP query that returns user entries.	v1.LDAPQuery
userNameAttributes	Defines which attributes on an LDAP user entry will be interpreted as its OpenShift Container Platform user name. The attribute to use as the name of the user in the OpenShift Container Platform group record. mail or sAMAccountName are preferred choices in most installations.	string array
groupMembershipAttributes	Defines which attributes on an LDAP user entry will be interpreted as the groups it is a member of.	string array
groupsQuery	Holds the template for an LDAP query that returns group entries.	v1.LDAPQuery

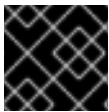
Name	Description	Schema
groupUIDAttribute	Defines which attribute on an LDAP group entry will be interpreted as its unique identifier. (ldapGroupUID)	string
groupNameAttributes	Defines which attributes on an LDAP group entry will be interpreted as its name to use for an OpenShift Container Platform group.	string array

CHAPTER 13. CONFIGURING LDAP FAILOVER

OpenShift Container Platform provides an [authentication provider](#) for use with Lightweight Directory Access Protocol (LDAP) setups, but it can connect to only a single LDAP server. During OpenShift Container Platform installation, you can configure the System Security Services Daemon (SSSD) for LDAP failover to ensure access to your cluster if one LDAP server fails.

The setup for this configuration is advanced and requires a separate authentication server, also called an **remote basic authentication server**, for OpenShift Container Platform to communicate with. You configure this server to pass extra attributes, such as email addresses, to OpenShift Container Platform so it can display them in the web console.

This topic describes how to complete this set up on a dedicated physical or virtual machine (VM), but you can also configure SSSD in containers.



IMPORTANT

You must complete all sections of this topic.

13.1. PREREQUISITES FOR CONFIGURING BASIC REMOTE AUTHENTICATION

- Before starting setup, you need to know the following information about your LDAP server:
 - Whether the directory server is powered by [FreeIPA](#), Active Directory, or another LDAP solution.
 - The Uniform Resource Identifier (URI) for the LDAP server, for example, **ldap.example.com**.
 - The location of the CA certificate for the LDAP server.
 - Whether the LDAP server corresponds to RFC 2307 or RFC2307bis for user groups.
- Prepare the servers:
 - **remote-basic.example.com**: A VM to use as the remote basic authentication server.
 - Select an operating system that includes SSSD version 1.12.0 for this server such as Red Hat Enterprise Linux 7.0 or later.
 - **openshift.example.com**: A new installation of OpenShift Container Platform.
 - You must not have an authentication method configured for this cluster.
 - Do not start OpenShift Container Platform on this cluster.

13.2. GENERATING AND SHARING CERTIFICATES WITH THE REMOTE BASIC AUTHENTICATION SERVER

Complete the following steps on the first master host listed in the Ansible host inventory file, by default **/etc/ansible/hosts**.

1. To ensure that communication between the remote basic authentication server and OpenShift Container Platform is trustworthy, create a set of Transport Layer Security (TLS) certificates to use during the other phases of this set up. Run the following command:

```
# openshift start \
  --public-master=https://openshift.example.com:8443 \
  --write-config=/etc/origin/
```

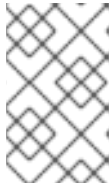
The output includes the **/etc/origin/master/ca.crt** and **/etc/origin/master/ca.key** signing certificates.

2. Use the signing certificate to generate keys to use on the remote basic authentication server:

```
# mkdir -p /etc/origin/remote-basic/
# oc adm ca create-server-cert \
  --cert='/etc/origin/remote-basic/remote-basic.example.com.crt' \
  --key='/etc/origin/remote-basic/remote-basic.example.com.key' \
  --hostnames=remote-basic.example.com \ 1
  --signer-cert='/etc/origin/master/ca.crt' \
  --signer-key='/etc/origin/master/ca.key' \
  --signer-serial='/etc/origin/master/ca.serial.txt'
```

1

A comma-separated list of all the host names and interface IP addresses that need to access the remote basic authentication server.



NOTE

The certificate files that you generate are valid for two years. You can alter this period by changing the **--expire-days** and **--signer-expire-days** values, but for security reasons, do not make them greater than 730.



IMPORTANT

If you do not list all host names and interface IP addresses that need to access the remote basic authentication server, the HTTPS connection will fail.

3. Copy the necessary certificates and key to the remote basic authentication server:

```
# scp /etc/origin/master/ca.crt \
  root@remote-basic.example.com:/etc/pki/CA/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.crt \
  root@remote-basic.example.com:/etc/pki/tls/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.key \
  root@remote-basic.example.com:/etc/pki/tls/private/
```

13.3. CONFIGURING SSSD FOR LDAP FAILOVER

Complete these steps on the remote basic authentication server.

You can configure the SSSD to retrieve attributes, such as email addresses and display names, and pass them to OpenShift Container Platform to display in the web interface. In the following steps, you configure the SSSD to provide email addresses to OpenShift Container Platform:

1. Install the required SSSD and the web server components:

```
# yum install -y sssd \
                    sssd-dbus \
                    realmd \
                    httpd \
                    mod_session \
                    mod_ssl \
                    mod_lookup_identity \
                    mod_authnz_pam \
                    php \
                    mod_php
```

2. Set up SSSD to authenticate this VM against the LDAP server. If the LDAP server is a FreeIPA or Active Directory environment, then use **realmd** to join this machine to the domain.

```
# realm join ldap.example.com
```

For more advanced cases, see the [System-Level Authentication Guide](#)

3. To use SSSD to manage failover situations for LDAP, add more entries to the **/etc/sss/sss.conf** file on the **ldap_uri** line. Systems that are enrolled with FreeIPA can automatically handle failover by using DNS SRV records.
4. Modify the **[domain/DOMAINNAME]** section of the **/etc/sss/sss.conf** file and add this attribute:

```
[domain/example.com]
...
ldap_user_extra_attrs = mail ❶
```

- ❶ Specify the correct attribute to retrieve email addresses for your LDAP solution. For IPA, specify **mail**. Other LDAP solutions might use another attribute, such as **email**.

5. Confirm that the **domain** parameter in the **/etc/sss/sss.conf** file contains only the domain name listed in the **[domain/DOMAINNAME]** section.

```
domains = example.com
```

6. Grant Apache permission to retrieve the email attribute. Add the following lines to the **[ifp]** section of the **/etc/sss/sss.conf** file:

```
[ifp]
user_attributes = +mail
allowed_uids = apache, root
```

7. To ensure that all of the changes are applied properly, restart SSSD:

```
$ systemctl restart sssd.service
```

- Test that the user information can be retrieved properly:

```
$ getent passwd <username>
username:*:12345:12345:Example User:/home/username:/usr/bin/bash
```

- Confirm that the mail attribute you specified returns an email address from your domain:

```
# dbus-send --print-reply --system --
dest=org.freedesktop.sssd.infopipe \
  /org/freedesktop/sss/infopipe
org.freedesktop.sssd.infopipe.GetUserAttr \
  string:username \ ❶
  array:string:mail ❷

method return time=1528091855.672691 sender=:1.2787 ->
destination=:1.2795 serial=13 reply_serial=2
array [
  dict entry(
    string "mail"
    variant
      array [
        string "username@example.com"
      ]
  )
]
```

❶ Provide a user name in your LDAP solution.

❷ Specify the attribute that you configured.

- Attempt to log into the VM as an LDAP user and confirm that you can log in using LDAP credentials. You can use either the local console or a remote service like SSH to log in.



IMPORTANT

By default, all users can log into the remote basic authentication server by using their LDAP credentials. You can change this behavior:

- If you use IPA joined systems, [configure host-based access control](#).
- If you use Active Directory joined systems, use a [group policy object](#).
- For other cases, see the [SSSD configuration](#) documentation.

13.4. CONFIGURING APACHE TO USE SSSD

- Create a `/etc/pam.d/openshift` file that contains the following contents:

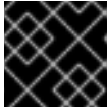
```
auth required pam_sss.so
account required pam_sss.so
```

This configuration enables PAM, the pluggable authentication module, to use **pam_sss.so** to determine authentication and access control when an authentication request is issued for the **openshift** stack.

2. Edit the `/etc/httpd/conf.modules.d/55-authnz_pam.conf` file and uncomment the following line:

```
LoadModule authnz_pam_module modules/mod_authnz_pam.so
```

3. To configure the Apache `httpd.conf` file for remote basic authentication, create the ***openshift-remote-basic-auth.conf*** file in the `/etc/httpd/conf.d` directory. Use the following template to provide your required settings and values:



IMPORTANT

Carefully review the template and customize its contents to fit your environment.

```
LoadModule request_module modules/mod_request.so
LoadModule php7_module modules/libphp7.so

# Nothing needs to be served over HTTP. This virtual host simply
# redirects to
# HTTPS.
<VirtualHost *:80>
    DocumentRoot /var/www/html
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    # This needs to match the certificates you generated. See the CN
    # and X509v3
    # Subject Alternative Name in the output of:
    # openssl x509 -text -in /etc/pki/tls/certs/remote-
    # basic.example.com.crt
    ServerName remote-basic.example.com

    DocumentRoot /var/www/html

    # Secure all connections with TLS
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/remote-basic.example.com.crt
    SSLCertificateKeyFile /etc/pki/tls/private/remote-
    basic.example.com.key
    SSLCACertificateFile /etc/pki/CA/certs/ca.crt

    # Require that TLS clients provide a valid certificate
    SSLVerifyClient require
    SSLVerifyDepth 10

    # Other SSL options that may be useful
    # SSLCertificateChainFile ...
    # SSLCARevocationFile ...

    # Send logs to a specific location to make them easier to find
    ErrorLog logs/remote_basic_error_log
    TransferLog logs/remote_basic_access_log
    LogLevel warn

    # PHP script that turns the Apache REMOTE_USER env var
```



```

# into a JSON formatted response that OpenShift understands
<Location /check_user.php>
# all requests not using SSL are denied
SSLRequireSSL
# denies access when SSLRequireSSL is applied
SSLOptions +StrictRequire
# Require both a valid basic auth user (so REMOTE_USER is always
set)
# and that the CN of the TLS client matches that of the
OpenShift master
<RequireAll>
    Require valid-user
    Require expr %{SSL_CLIENT_S_DN_CN} == 'system:openshift-
master'
</RequireAll>
# Use basic auth since OpenShift will call this endpoint with a
basic challenge
AuthType Basic
AuthName openshift
AuthBasicProvider PAM
AuthPAMService openshift

# Store attributes in environment variables. Specify the email
attribute that
# you confirmed.
LookupOutput Env
LookupUserAttr mail REMOTE_USER_MAIL
LookupUserGECOS REMOTE_USER_DISPLAY_NAME

# Other options that might be useful

# While REMOTE_USER is used as the sub field and serves as the
immutable ID,
# REMOTE_USER_PREFERRED_USERNAME could be used to have a
different username
# LookupUserAttr <attr_name> REMOTE_USER_PREFERRED_USERNAME

# Group support may be added in a future release
# LookupUserGroupsIter REMOTE_USER_GROUP
</Location>

# Deny everything else
<Location ~ "^(?!\/check_user\.php)\.*)">
    Deny from all
</Location>
</VirtualHost>

```

4. Create the **check_user.php** script in the **/var/www/html** directory. Include the following code:

```

<?php
// Get the user based on the Apache var, this should always be
// set because we 'Require valid-user' in the configuration
$user = apache_getenv('REMOTE_USER');

// However, we assume it may not be set and
// build an error response by default

```

```

$data = array(
    'error' => 'remote PAM authentication failed'
);

// Build a success response if we have a user
if (!empty($user)) {
    $data = array(
        'sub' => $user
    );
    // Map of optional environment variables to optional JSON fields
    $env_map = array(
        'REMOTE_USER_MAIL' => 'email',
        'REMOTE_USER_DISPLAY_NAME' => 'name',
        'REMOTE_USER_PREFERRED_USERNAME' => 'preferred_username'
    );

    // Add all non-empty environment variables to JSON data
    foreach ($env_map as $env_name => $json_name) {
        $env_data = apache_getenv($env_name);
        if (!empty($env_data)) {
            $data[$json_name] = $env_data;
        }
    }
}

// We always output JSON from this script
header('Content-Type: application/json', true);

// Write the response as JSON
echo json_encode($data);
?>

```

5. Enable Apache to load the module. Modify the `/etc/httpd/conf.modules.d/55-lookup_identity.conf` file and uncomment the following line:

```
LoadModule lookup_identity_module modules/mod_lookup_identity.so
```

6. Set an SELinux boolean so that SELinux allows Apache to connect to SSSD over D-BUS:

```
# setsebool -P httpd_dbus_sssd on
```

7. Set a boolean to tell SELinux that it is acceptable for Apache to contact the PAM subsystem:

```
# setsebool -P allow_httpd_mod_auth_pam on
```

8. Start Apache:

```
# systemctl start httpd.service
```

13.5. CONFIGURING OPENSIFT CONTAINER PLATFORM TO USE SSSD AS THE BASIC REMOTE AUTHENTICATION SERVER

Modify the default configuration of your cluster to use the new identity provider that you created. Complete the following steps on the first master host listed in the Ansible host inventory file.

1. Open the `/etc/origin/master/master-config.yaml` file.
2. Locate the **identityProviders** section and replace it with the following code:

```
identityProviders:
- name: sssd
  challenge: true
  login: true
  mappingMethod: claim
  provider:
    apiVersion: v1
    kind: BasicAuthPasswordIdentityProvider
    url: https://remote-basic.example.com/check_user.php
    ca: /etc/origin/master/ca.crt
    certFile: /etc/origin/master/openshift-master.crt
    keyFile: /etc/origin/master/openshift-master.key
```

3. Start OpenShift Container Platform with the updated configuration:

```
# openshift start \
--public-master=https://openshift.example.com:8443 \
--master-config=/etc/origin/master/master-config.yaml \
--node-config=/etc/origin/node-node1.example.com/node-
config.yaml
```

4. Test a login by using the **oc** CLI:

```
oc login https://openshift.example.com:8443
```

You can log in only with valid LDAP credentials.

5. List the identities and confirm that an email address is displayed for each user name. Run the following command:

```
$ oc get identity -o yaml
```

CHAPTER 14. CONFIGURING THE SDN

14.1. OVERVIEW

The [OpenShift SDN](#) enables communication between pods across the OpenShift Container Platform cluster, establishing a *pod network*. Three [SDN plug-ins](#) are currently available (**ovs-subnet**, **ovs-multitenant**, and **ovs-networkpolicy**), which provide different methods for configuring the pod network.

14.2. AVAILABLE SDN PROVIDERS

The upstream Kubernetes project does not come with a default network solution. Instead, Kubernetes has developed a Container Network Interface (CNI) to allow network providers for integration with their own SDN solutions.

There are several OpenShift SDN plug-ins available out of the box from Red Hat, as well as third-party plug-ins.

Red Hat has worked with a number of SDN providers to certify their SDN network solution on OpenShift Container Platform via the Kubernetes CNI interface, including a support process for their SDN plug-in through their product's entitlement process. Should you open a support case with OpenShift, Red Hat can facilitate an exchange process so that both companies are involved in meeting your needs.

The following SDN solutions are validated and supported on OpenShift Container Platform directly by the third-party vendor:

- Cisco Contiv (™)
- Juniper Contrail (™)
- Nokia Nuage (™)
- Tigera Calico (™)
- VMware NSX-T (™)

Installing VMware NSX-T (™) on OpenShift Container Platform

VMware NSX-T (™) provides an SDN and security infrastructure to build cloud-native application environments. In addition to vSphere hypervisors (ESX), these environments include KVM and native public clouds.

The current integration requires a *new* install of both NSX-T and OpenShift Container Platform. Currently, NSX-T version 2.1 is supported, and only supports the use of ESX and KVM hypervisors at this time.

See the [NSX-T Container Plug-in for OpenShift - Installation and Administration Guide](#) for more information.

14.3. CONFIGURING THE POD NETWORK WITH ANSIBLE

For initial cluster installations, the **ovs-subnet** plug-in is installed and configured by default, though it can be overridden during installation using the **os_sdn_network_plugin_name** parameter, which is configurable in the Ansible inventory file.

For example, to override the standard **ovs-subnet** plug-in and use the **ovs-multitenant** plug-in instead:

■

```
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-
subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
```

See [Configuring Cluster Variables](#) for descriptions of networking-related Ansible variables that can be set in your inventory file.

14.4. CONFIGURING THE POD NETWORK ON MASTERS

The cluster administrators can control pod network settings on master hosts by modifying parameters in the **networkConfig** section of the [master configuration file](#) (located at `/etc/origin/master/master-config.yaml` by default):

Configuring a pod network for a single CIDR

```
networkConfig:
  clusterNetworks:
    - cidr: 10.128.0.0/14 ①
      hostSubnetLength: 9 ②
  networkPluginName: "redhat/openshift-ovs-subnet" ③
  serviceNetworkCIDR: 172.30.0.0/16 ④
```

- ① Cluster network for node IP allocation
- ② Number of bits for pod IP allocation within a node
- ③ Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in, **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in, or **redhat/openshift-ovs-networkpolicy** for the **ovs-networkpolicy** plug-in
- ④ Service IP allocation for the cluster

Alternatively, you can create a pod network with multiple CIDR ranges by adding separate ranges into the **clusterNetworks** field with the range and the **hostSubnetLength**.

Multiple ranges can be used at once, and the range can be expanded or contracted. Nodes can be moved from one range to another by evacuating a node, then deleting and re-creating the node. See the [Managing Nodes](#) section for more information. Node allocations occur in the order listed, then when the range is full, move to the next on the list.

Configuring a pod network for multiple CIDRs

```
networkConfig:
  clusterNetworks:
    - cidr: 10.128.0.0/14 ①
      hostSubnetLength: 9 ②
    - cidr: 10.132.0.0/14
      hostSubnetLength: 9
  externalIPNetworkCIDRs: null
  hostSubnetLength: 9
  ingressIPNetworkCIDR: 172.29.0.0/16
  networkPluginName: redhat/openshift-ovs-multitenant ③
  serviceNetworkCIDR: 172.30.0.0/16
```

-
- 1 Cluster network for node IP allocation.
- 2 Number of bits for pod IP allocation within a node.
- 3 Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in, **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in, or **redhat/openshift-ovs-networkpolicy** for the **ovs-networkpolicy** plug-in.

You can add elements to the **clusterNetworks** value, or remove them if no node is using that CIDR range, but be sure to restart the API and master services for any changes to take effect.

```
master-restart api
master-restart controllers
```

IMPORTANT

The **hostSubnetLength** value cannot be changed after the cluster is first created, A **cidr** field can only be changed to be a larger network that still contains the original network if nodes are allocated within it's range , and **serviceNetworkCIDR** can only be expanded. For example, given the default value of **10.128.0.0/14**, you could change **cidr** to **10.128.0.0/9** (i.e., the entire upper half of net 10) but not to **10.64.0.0/16**, because that does not overlap the original value.

You can change **serviceNetworkCIDR** from **172.30.0.0/16** to **172.30.0.0/15**, but not to **172.28.0.0/14**, because even though the original range is entirely inside the new range, the original range must be at the start of the CIDR. See [Expanding the service network](#) for more information.

14.5. CONFIGURING THE POD NETWORK ON NODES

The cluster administrators can control pod network settings on nodes by modifying parameters in the **networkConfig** section of the appropriate [node configuration map](#):

```
networkConfig:
  mtu: 1450 1
  networkPluginName: "redhat/openshift-ovs-subnet" 2
```

- 1 Maximum transmission unit (MTU) for the pod overlay network
- 2 Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in, **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in, or **redhat/openshift-ovs-networkpolicy** for the **ovs-networkpolicy** plug-in

NOTE

You must change the MTU size on all masters and nodes that are part of the OpenShift Container Platform SDN. Also, the MTU size of the tun0 interface must be the same across all nodes that are part of the cluster.

14.6. EXPANDING THE SERVICE NETWORK

If you are running low on addresses in your service network, you can expand the range as long as you ensure that the current range is at the beginning of the new range:

1. Change the **serviceNetworkCIDR** parameter in the configuration files for all masters (**/etc/origin/master/master-config.yaml** by default). Change only the number following the / to a smaller number.

2. Delete the **clusterNetwork** object:

```
$ oc delete clusternetwork default
```

3. Restart the master host services:

```
# master-restart controllers
```

4. Update the value of the **openshift_portal_net** variable in the Ansible inventory file to the new CIDR:

```
# Configure SDN cluster network and kubernetes service CIDR blocks.
These
# network blocks should be private and should not conflict with
network blocks
# in your infrastructure that pods may require access to. Can not be
changed
# after deployment.
#osm_cluster_network_cidr=10.1.0.0/16
#openshift_portal_net=172.30.0.0/<new_CIDR_range>
```

5. Run the **config.yml** Ansible playbook to redeploy the cluster:

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-master/config.yml
```

6. From a master instance, or as a cluster administrator, allow the evacuation of any pod from the node and disable scheduling of other pods on that node:

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
```

NAME	STATUS	AGE
VERSION		
ose-app-node01.example.com	Ready,SchedulingDisabled	20m
v1.6.1+5115d708d7		

```
$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```

**NOTE**

If there are containers running with local volumes that will not migrate, run the following command: **oc adm drain \${NODE} --ignore-daemonsets --delete-local-data**.

7. List the pods on the node to verify that they have been removed:

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME          READY    STATUS    RESTARTS   AGE
```

8. Repeat the previous two steps for each node.

For more information on evacuating and draining pods or nodes, see [Node maintenance](#).

14.7. MIGRATING BETWEEN SDN PLUG-INS

If you are already using one SDN plug-in and want to switch to another:

1. Change the **networkPluginName** parameter on all [masters](#) and [nodes](#) in their configuration files.
2. Restart the API and master services on all masters.

```
master-restart api
master-restart controllers
```

3. Stop the node service on all masters and nodes by removing the pod definition:

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

4. If you are switching between OpenShift SDN plug-ins, restart OpenShift SDN on all masters and nodes.

```
oc delete pod --all -n openshift-sdn
```

5. Restart the node service on all masters and nodes.

```
# systemctl restart atomic-openshift-node.service
```

6. If you are switching from an OpenShift SDN plug-in to a third-party plug-in, then clean up OpenShift SDN-specific artifacts:

```
$ oc delete clusternetwork --all
$ oc delete hostsubnets --all
$ oc delete netnamespaces --all
```


When switching from the **ovs-subnet** to the **ovs-multitenant** OpenShift SDN plug-in, all the existing projects in the cluster will be fully isolated (assigned unique VNIDs). The cluster administrators can choose to [modify the project networks](#) using the administrator CLI.

Check VNIDs by running:

```
$ oc get netnamespace
```

14.7.1. Migrating from ovs-multitenant to ovs-networkpolicy

In addition to the generic plug-in migration steps above in the [Migrating between SDN plug-ins](#) section, there is one additional step when migrating from the **ovs-multitenant** plug-in to the **ovs-networkpolicy** plug-in; you must ensure that every namespace has a unique **NetID**. This means that if you have previously [joined projects together](#) or [made projects global](#), you will need to undo that before switching to the **ovs-networkpolicy** plug-in, or the NetworkPolicy objects may not function correctly.

A helper script is available that fixes **NetID**'s, creates NetworkPolicy objects to isolate previously-isolated namespaces, and enables connections between previously-joined namespaces.

Use the following steps to migrate to the **ovs-networkpolicy** plug-in, by using this helper script, while still running the **ovs-multitenant** plug-in:

1. Download the script and add the execution file permission:

```
$ curl -O
https://raw.githubusercontent.com/openshift/origin/master/contrib/mi
gration/migrate-network-policy.sh
$ chmod a+x migrate-network-policy.sh
```

2. Run the script (requires the cluster administrator role).

```
$ ./migrate-network-policy.sh
```

After running this script, every namespace is fully isolated from every other namespace, therefore connection attempts between pods in different namespaces will fail until you complete the migration to the **ovs-networkpolicy** plug-in.

If you want newly-created namespaces to also have the same policies by default, you can set [default NetworkPolicy objects](#) to be created matching the **default-deny** and **allow-from-global-namespaces** policies created by the migration script.



NOTE

In case of script failures or other errors, or if you later decide you want to revert back to the **ovs-multitenant** plug-in, you can use the [un-migration script](#). This script undoes the changes made by the migration script and re-joins previously-joined namespaces.

14.8. EXTERNAL ACCESS TO THE CLUSTER NETWORK

If a host that is external to OpenShift Container Platform requires access to the cluster network, you have two options:

1. Configure the host as an OpenShift Container Platform node but mark it [unschedulable](#) so that the master does not schedule containers on it.

2. Create a tunnel between your host and a host that is on the cluster network.

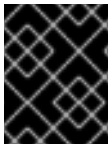
Both options are presented as part of a practical use-case in the documentation for configuring [routing from an edge load-balancer to containers within OpenShift SDN](#).

14.9. USING FLANNEL

As an alternate to the default SDN, OpenShift Container Platform also provides Ansible playbooks for installing **flannel**-based networking. This is useful if running OpenShift Container Platform within a cloud provider platform that also relies on SDN, such as Red Hat OpenStack Platform, and you want to avoid encapsulating packets twice through both platforms.

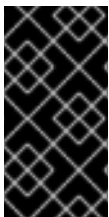
Flannel uses a single IP network space for all of the containers allocating a contiguous subset of the space to each instance. Consequently, nothing prevents a container from attempting to contact any IP address in the same network space. This hinders multi-tenancy because the network cannot be used to isolate containers in one application from another.

Depending on whether you prefer multi-tenancy isolation or performance, you should determine the appropriate choice when deciding between OpenShift SDN (multi-tenancy) and flannel (performance) for internal networks.



IMPORTANT

Flannel is only supported for OpenShift Container Platform on Red Hat OpenStack Platform.



IMPORTANT

The current version of Neutron enforces port security on ports by default. This prevents the port from sending or receiving packets with a MAC address different from that on the port itself. Flannel creates virtual MACs and IP addresses and must send and receive packets on the port, so port security must be disabled on the ports that carry flannel traffic.

To enable flannel within your OpenShift Container Platform cluster:

1. Neutron port security controls must be configured to be compatible with Flannel. The default configuration of Red Hat OpenStack Platform disables user control of **port_security**. Configure Neutron to allow users to control the **port_security** setting on individual ports.
 - a. On the Neutron servers, add the following to the `/etc/neutron/plugins/ml2/ml2_conf.ini` file:

```
[ml2]
...
extension_drivers = port_security
```

- b. Then, restart the Neutron services:

```
service neutron-dhcp-agent restart
service neutron-ovs-cleanup restart
service neutron-metadata-agent restart
service neutron-l3-agent restart
service neutron-plugin-openvswitch-agent restart
service neutron-vpn-agent restart
service neutron-server restart
```

2. When creating the OpenShift Container Platform instances on Red Hat OpenStack Platform, disable both port security and security groups in the ports where the container network flannel interface will be:

```
neutron port-update $port --no-security-groups --port-security-enabled=False
```



NOTE

Flannel gather information from etcd to configure and assign the subnets in the nodes. Therefore, the security group attached to the etcd hosts should allow access from nodes to port 2379/tcp, and nodes security group should allow egress communication to that port on the etcd hosts.

- a. Set the following variables in your Ansible inventory file before running the installation:

```
openshift_use_openshift_sdn=false ❶  
openshift_use_flannel=true ❷  
flannel_interface=eth0
```

- ❶ Set **openshift_use_openshift_sdn** to **false** to disable the default SDN.
- ❷ Set **openshift_use_flannel** to **true** to enable **flannel** in place.

- b. Optionally, you can specify the interface to use for inter-host communication using the **flannel_interface** variable. Without this variable, the OpenShift Container Platform installation uses the default interface.



NOTE

Custom networking CIDR for pods and services using flannel will be supported in a future release. [BZ#1473858](#)

3. After the OpenShift Container Platform installation, add a set of iptables rules on every OpenShift Container Platform node:

```
iptables -A DOCKER -p all -j ACCEPT  
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

To persist those changes in the **/etc/sysconfig/iptables** use the following command on every node:

```
cp /etc/sysconfig/iptables{,.orig}  
sh -c "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/'  
s/:DOCKER -/:DOCKER ACCEPT/' | awk '!\"p && /POSTROUTING/{print \"-  
A POSTROUTING -o eth1 -j MASQUERADE\"; p=1} 1' | tac >  
/etc/sysconfig/iptables"
```

**NOTE**

The **iptables-save** command saves all the current *in memory* iptables rules. However, because Docker, Kubernetes and OpenShift Container Platform create a high number of iptables rules (services, etc.) not designed to be persisted, saving these rules can become problematic.

To isolate container traffic from the rest of the OpenShift Container Platform traffic, Red Hat recommends creating an isolated tenant network and attaching all the nodes to it. If you are using a different network interface (eth1), remember to configure the interface to start at boot time through the ***/etc/sysconfig/network-scripts/ifcfg-eth1*** file:

```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFROUTE=no
PEERDNS=no
```

CHAPTER 15. CONFIGURING NUAGE SDN

15.1. NUAGE SDN AND OPENSIFT CONTAINER PLATFORM

Nuage Networks Virtualized Services Platform (VSP) provides virtual networking and software-defined networking (SDN) infrastructure to Docker container environments that simplifies IT operations and expands OpenShift Container Platform's native networking capabilities.

Nuage Networks VSP supports Docker-based applications running on OpenShift Container Platform to accelerate the provisioning of virtual networks between pods and traditional workloads, and to enable security policies across the entire cloud infrastructure. VSP allows for the automation of security appliances to include granular security and microsegmentation policies for container applications.

Integrating VSP with the OpenShift Container Platform application workflow allows business applications to be quickly turned up and updated by removing the network lag faced by DevOps teams. VSP supports different workflows with OpenShift Container Platform in order to accommodate scenarios where users can choose ease-of-use or complete control using policy-based automation.

See [Networking](#) for more information on how VSP is integrated with OpenShift Container Platform.

15.2. DEVELOPER WORKFLOW

This workflow is used in developer environments and requires little input from the developer in setting up the networking. In this workflow, **nuage-openshift-monitor** is responsible for creating the VSP constructs (Zone, Subnets, etc.) needed to provide appropriate policies and networking for pods created in an OpenShift Container Platform project. When a project is created, a default zone and default subnet for that project are created by **nuage-openshift-monitor**. When the default subnet created for a given project gets depleted, **nuage-openshift-monitor** dynamically creates additional subnets.



NOTE

A separate VSP Zone is created for each OpenShift Container Platform project ensuring isolation amongst the projects.

15.3. OPERATIONS WORKFLOW

This workflow is used by operations teams rolling out applications. In this workflow, the network and security policies are first configured on the VSD in accordance with the rules set by the organization to deploy applications. Administrative users can potentially create multiple zones and subnets and map them to the same project using labels. While spinning up the pods, the user can use the Nuage Labels to specify what network a pod needs to attach to and what network policies need to be applied to it. This allows for deployments where inter- and intra-project traffic can be controlled in a fine-grained manner. For example, inter-project communication is enabled on a project by project basis. This may be used to connect projects to common services that are deployed in a shared project.

15.4. INSTALLATION

The VSP integration with OpenShift Container Platform works for both virtual machines (VMs) and bare metal OpenShift Container Platform installations.

An environment with High Availability (HA) can be configured with multiple masters and multiple nodes.

Nuage VSP integration in multi-master mode only supports the native HA configuration method

described in this section. This can be combined with any load balancing solution, the default being HAProxy. The inventory file contains three master hosts, the nodes, an etcd server, and a host that functions as the HAProxy to balance the master API on all master hosts. The HAProxy host is defined in the [lb] section of the inventory file enabling Ansible to automatically install and configure HAProxy as the load balancing solution.

In the Ansible nodes file, the following parameters need to be specified in order to setup Nuage VSP as the network plug-in:

```
# Create and OSEv3 group that contains masters, nodes, load-balancers,
and etcd hosts
masters
nodes
etcd
lb

# Nuage specific parameters
openshift_use_openshift_sdn=False
openshift_use_nuage=True
os_sdn_network_plugin_name='nuage/vsp-openshift'
openshift_node_proxy_mode='userspace'

# VSP related parameters
vsd_api_url=https://192.168.103.200:8443
vsp_version=v4_0
enterprise=nuage
domain=openshift
vsc_active_ip=192.168.103.201
vsc_standby_ip=192.168.103.202
uplink_interface=eth0

# rpm locations
nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/n
uage-openshift-monitor-4.0.X.1830.el7.centos.x86_64.rpm
vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-
openvswitch-4.0.X.225.el7.x86_64.rpm
plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-
openshift-4.0.X1830.el7.centos.x86_64.rpm

# Required for Nuage Monitor REST server and HA
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.nuageopenshift.com
openshift_master_cluster_public_hostname=lb.nuageopenshift.com
nuage_openshift_monitor_rest_server_port=9443

# Optional parameters
nuage_interface_mtu=1460
nuage_master_adminusername='admin's user-name'
nuage_master_adminuserpasswd='admin's password'
nuage_master_cspadminpasswd='csp admin password'
nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor

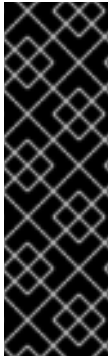
# Required for brownfield install (where a {product-title} cluster exists
without Nuage as the networking plugin)
nuage_dockker_bridge=lbr0
```

```
# Specify master hosts
[masters]
fqdn_of_master_1
fqdn_of_master_2
fqdn_of_master_3

# Specify load balancer host
[lb]
fqdn_of_load_balancer
```

CHAPTER 16. CONFIGURING KURYR SDN

16.1. KURYR SDN AND OPENSIFT CONTAINER PLATFORM



IMPORTANT

The ability to configure Kuryr SDN is a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

Kuryr (or more specifically Kuryr-Kubernetes) is an SDN solution built using **CNI** and **OpenStack Neutron**. Its advantages include being able to use a wide range of Neutron SDN backends and providing inter-connectivity between Kubernetes pods and OpenStack virtual machines (VMs).

Kuryr-Kubernetes and OpenShift Container Platform integration is primarily designed for OpenShift Container Platform clusters running on OpenStack VMs. Kuryr-Kubernetes components are installed as pods on OpenShift Container Platform in the **openshift-infra** namespace:

- kuryr-controller - a single service instance, installed on any node. Modeled in OpenShift Container Platform as a **Deployment**.
- kuryr-cni - container installing and configuring Kuryr as CNI driver on each OpenShift Container Platform node. Modeled in OpenShift Container Platform as a **DaemonSet**.

16.2. INSTALLATION

The system running openshift-ansible must be subscribed to the OSP as well as OCP repositories. The OpenStack integration requires a few extra packages. To install the dependencies, run:

```
$ sudo yum install -y ansible openshift-ansible python2-shade python-dns \
    python2-heatclient python2-octaviaclient python-openstackclient bind-
utils
```

In the Ansible nodes file, specify the following parameters in order to set up Kuryr-Kubernetes as the network plug-in:

```
# Enable Kuryr.
openshift_use_openshift_sdn=False
openshift_use_kuryr=True
os_sdn_network_plugin_name=cni

# Set userspace so that there are no iptables remains.
openshift_node_proxy_mode='userspace'

# Keystone URL.
kuryr_openstack_auth_url=http://127.0.0.1/identity

# OpenStack domain name of user owning Kuryr resources.
```



```

kuryr_openstack_user_domain_name=default

# OpenStack project name of user owning Kuryr resources.
kuryr_openstack_user_project_name=admin

# OpenStack project id for Kuryr resources.
kuryr_openstack_project_id=ec0b31802fd043c08bc15b74d2f9a3d3

# OpenStack username that will own kuryr resources.
kuryr_openstack_username=admin

# Password for that user.
kuryr_openstack_password=password

# Default Neutron security groups' IDs for Kubernetes pods
kuryr_openstack_pod_sg_id=f74c83a8-a520-421a-930e-21b6cd098c6a,01f85594-
9950-4ded-a92c-5ad546a41188

# Default Neutron subnet ID for Kubernetes pods.
kuryr_openstack_pod_subnet_id=c85cdee6-0ed1-4d8f-ae61-7afa4674b311

# Default OpenStack project ID for Kubernetes resources.
kuryr_openstack_pod_project_id=ec0b31802fd043c08bc15b74d2f9a3d3

# Neutron subnet ID for Kubernetes worker node VMs.
kuryr_openstack_worker_nodes_subnet_id=477cfa49-e641-4d31-a7b5-
5bc834743f61

# Default Neutron subnet ID for Kubernetes services.
kuryr_openstack_service_subnet_id=3b31a106-4084-4db9-bc0c-00b97afe186e

```

You must also specify an OpenStack cloud provider as described in the [OpenStack configuration documentation](#).

Prior to the installation, you must also provide a DNS server the OpenShift Container Platform nodes will be using for internal name resolution. OpenStack does not provide a node name resolution out of the box. In the following example, **10.20.30.40** is the IP address of the DNS server:

```

openshift_openstack_dns_nameservers=[10.20.30.40]

```

If the DNS server supports remote updates via **nsupdate** (RFC 2136), the playbooks can populate it automatically, if you add the following configuration:

```

openshift_openstack_external_nsupdate_keys={private: {"key_secret": "
<nsupdate key>", "key_algorithm": "<nsupdate key algorithm>", "key_name": "
<nsupdate key name>", "server": 10.20.30.40}}

```

Finally, install OpenShift Container Platform by running the **provision_install.yml** playbook. You must specify the dynamic inventory file, **inventory.py**, and the the path to the Ansible nodes file that you created:

```

$ ansible-playbook --user openshift -i /usr/share/ansible/openshift-
ansible/playbooks/openstack/inventory.py -i ansible-nodes.txt
/usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-
cluster/provision_install.yml

```

If you want to do any custom setup on the created nodes before the OpenShift Container Platform installation, you can run the ***provision.yml*** and ***install.yml*** playbooks separately. ***provision.yml*** will create the OpenStack resources (nodes, networks, and so on) and ***install.yml*** will install OpenShift Container Platform.

16.3. VERIFICATION

Once the installation of OpenShift Container Platform is finished, you can check if Kuryr pods are deployed successfully:

```
$ oc -n openshift-infra get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE
bootstrap-autoapprover-0	1/1	Running	0	3d
10.11.0.7 master-0.openshift.example.com				
kuryr-cni-ds-66kt2	2/2	Running	0	3d
192.168.99.14 infra-node-0.openshift.example.com				
kuryr-cni-ds-ggcpz	2/2	Running	0	3d
192.168.99.16 master-0.openshift.example.com				
kuryr-cni-ds-mhzjt	2/2	Running	0	3d
192.168.99.6 app-node-1.openshift.example.com				
kuryr-cni-ds-njctb	2/2	Running	0	3d
192.168.99.12 app-node-0.openshift.example.com				
kuryr-cni-ds-v8hp8	2/2	Running	0	3d
192.168.99.5 infra-node-1.openshift.example.com				
kuryr-controller-59fc7f478b-qwk4k	1/1	Running	0	3d
192.168.99.5 infra-node-1.openshift.example.com				

kuryr-cni pods should run on every OpenShift Container Platform node. Single kuryr-controller instances should run on any of the nodes.

CHAPTER 17. CONFIGURING FOR AMAZON WEB SERVICES (AWS)

17.1. OVERVIEW

OpenShift Container Platform can be configured to access an [AWS EC2 infrastructure](#), including [using AWS volumes as persistent storage](#) for application data. After you configure AWS, some additional configurations must be completed on the OpenShift Container Platform hosts.

17.2. PERMISSIONS

Configuring AWS for OpenShift Container Platform requires the following permissions:

Table 17.1. Master Permissions

Elastic Compute Cloud(EC2)	<code>ec2:DescribeVolume</code> , <code>ec2:CreateVolume</code> , <code>ec2:CreateTags</code> , <code>ec2:DescribeInstance</code> , <code>ec2:AttachVolume</code> , <code>ec2:DetachVolume</code> , <code>ec2>DeleteVolume</code> , <code>ec2:DescribeSubnets</code> , <code>ec2:CreateSecurityGroup</code> , <code>ec2:DescribeSecurityGroups</code> , <code>ec2:DescribeRouteTables</code> , <code>ec2:AuthorizeSecurityGroupIngress</code> , <code>ec2:RevokeSecurityGroupIngress</code> , <code>ec2>DeleteSecurityGroup</code>
Elastic Load Balancing	<code>elasticloadbalancing:DescribeTags</code> , <code>elasticloadbalancing:CreateLoadBalancerListeners</code> , <code>elasticloadbalancing:ConfigureHealthCheck</code> , <code>elasticloadbalancing>DeleteLoadBalancerListeners</code> , <code>elasticloadbalancing:RegisterInstancesWithLoadBalancer</code> , <code>elasticloadbalancing:DescribeLoadBalancers</code> , <code>elasticloadbalancing:CreateLoadBalancer</code> , <code>elasticloadbalancing>DeleteLoadBalancer</code> , <code>elasticloadbalancing:ModifyLoadBalancerAttributes</code> , <code>elasticloadbalancing:DescribeLoadBalancerAttributes</code>

Table 17.2. Node Permissions

Elastic Compute Cloud(EC2)	<code>ec2:DescribeInstance*</code>
----------------------------	------------------------------------



IMPORTANT

- Every master host, node host, and subnet must have the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag.
- One security group, preferably the one linked to the nodes, must have the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag.
 - Do not tag all security groups with the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag or the Elastic Load Balancing (ELB) will not be able to create a load balancer.

17.3. CONFIGURING A SECURITY GROUP

When installing OpenShift Container Platform on AWS, ensure that you set up the appropriate security groups.

These are some ports that you must have in your security groups, without which the installation fails. You may need more depending on the cluster configuration you want to install. For more information and to adjust your security groups accordingly, see [Required Ports](#) for more information.

All OpenShift Container Platform Hosts	<ul style="list-style-type: none"> • tcp/22 from host running the installer/Ansible
etcd Security Group	<ul style="list-style-type: none"> • tcp/2379 from masters • tcp/2380 from etcd hosts
Master Security Group	<ul style="list-style-type: none"> • tcp/8443 from 0.0.0.0/0 • tcp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 • udp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 • tcp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2 • udp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2
Node Security Group	<ul style="list-style-type: none"> • tcp/10250 from masters • udp/4789 from nodes
Infrastructure Nodes (ones that can host the OpenShift Container Platform router)	<ul style="list-style-type: none"> • tcp/443 from 0.0.0.0/0 • tcp/80 from 0.0.0.0/0

CRI-O	If using CRI-O, you must open tcp/10010 to allow oc exec and oc rsh operations.
--------------	---

If configuring external load-balancers (ELBs) for load balancing the masters and/or routers, you also need to configure Ingress and Egress security groups for the ELBs appropriately.

17.3.1. Overriding Detected IP Addresses and Host Names

In AWS, situations that require overriding the variables include:

Variable	Usage
hostname	The user is installing in a VPC that is not configured for both DNS hostnames and DNS resolution .
ip	You have multiple network interfaces configured and want to use one other than the default.
public_hostname	<ul style="list-style-type: none"> A master instance where the VPC subnet is not configured for Auto-assign Public IP. For external access to this master, you need to have an ELB or other load balancer configured that would provide the external access needed, or you need to connect over a VPN connection to the internal name of the host. A master instance where metadata is disabled. This value is not actually used by the nodes.
public_ip	<ul style="list-style-type: none"> A master instance where the VPC subnet is not configured for Auto-assign Public IP. A master instance where metadata is disabled. This value is not actually used by the nodes.

For EC2 hosts in particular, they must be deployed in a VPC that has both **DNS host names** and **DNS resolution** enabled.

17.4. CONFIGURING AWS VARIABLES

To set the required AWS variables, create a `/etc/origin/cloudprovider/aws.conf` file with the following contents on all of your OpenShift Container Platform hosts, both masters and nodes:

```
[Global]
Zone = us-east-1c 1
```

- 1** This is the Availability Zone of your AWS Instance and where your EBS Volume resides; this information is obtained from the AWS Management Console.

17.5. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR AWS

You can set the AWS configuration on OpenShift Container Platform in two ways:

- using [Ansible](#) or
- manually, by modifying the [master-config.yaml](#), [node-config.yaml](#), and related [/etc/sysconfig/](#) files.

17.5.1. Configuring OpenShift Container Platform for AWS with Ansible

During cluster installations, AWS can be configured using the `openshift_cloudprovider_aws_access_key`, `openshift_cloudprovider_aws_secret_key`, `openshift_cloudprovider_kind`, `openshift_clusterid` parameters, which are configurable in the [inventory file](#).

Example AWS Configuration with Ansible

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_aws_access_key="{{
lookup('env', 'AWS_ACCESS_KEY_ID') }}"
#openshift_cloudprovider_aws_secret_key="{{
lookup('env', 'AWS_SECRET_ACCESS_KEY') }}"
#
#openshift_clusterid=unique_identifier_per_availability_zone
#
# AWS (Using API Credentials)
#openshift_cloudprovider_kind=aws
#openshift_cloudprovider_aws_access_key=aws_access_key_id
#openshift_cloudprovider_aws_secret_key=aws_secret_access_key
#
# AWS (Using IAM Profiles)
#openshift_cloudprovider_kind=aws
# Note: IAM roles must exist before launching the instances.
```

NOTE

When Ansible configures AWS, it automatically makes the necessary changes to the following files:

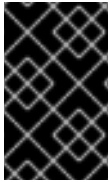
- `/etc/origin/cloudprovider/aws.conf`
- `/etc/origin/master/master-config.yaml`
- `/etc/origin/node/node-config.yaml`

17.5.2. Manually Configuring OpenShift Container Platform Masters for AWS

Edit or [create](#) the master configuration file on all masters (`/etc/origin/master/master-config.yaml` by default) and update the contents of the `apiServerArguments` and `controllerArguments` sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
  controllerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
```

Currently, the **nodeName** must match the instance name in AWS in order for the cloud provider integration to work properly. The name must also be RFC1123 compliant.



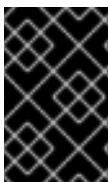
IMPORTANT

When triggering a containerized installation, only the directories of `/etc/origin` and `/var/lib/origin` are mounted to the master and node container. Therefore, **aws.conf** should be in `/etc/origin/` instead of `/etc/`.

17.5.3. Manually Configuring OpenShift Container Platform Nodes for AWS

Edit the appropriate [node configuration map](#) and update the contents of the `kubeletArguments` section:

```
kubeletArguments:
  cloud-provider:
    - "aws"
  cloud-config:
    - "/etc/origin/cloudprovider/aws.conf"
```



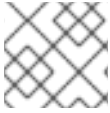
IMPORTANT

When triggering a containerized installation, only the directories of `/etc/origin` and `/var/lib/origin` are mounted to the master and node container. Therefore, **aws.conf** should be in `/etc/origin/` instead of `/etc/`.

17.5.4. Manually Setting Key-Value Access Pairs

Make sure the following environment variables are set in the `/etc/origin/master/master.env` file on masters and the `/etc/sysconfig/atomic-openshift-node` file on nodes:

```
AWS_ACCESS_KEY_ID=<key_ID>
AWS_SECRET_ACCESS_KEY=<secret_key>
```

**NOTE**

Access keys are obtained when setting up your AWS IAM user.

17.6. APPLYING CONFIGURATION CHANGES

Start or restart OpenShift Container Platform services on all master and node hosts to apply your configuration changes, see [Restarting OpenShift Container Platform services](#):

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

Switching from not using a cloud provider to using a cloud provider produces an error message. Adding the cloud provider tries to delete the node because the node switches from using the **hostname** as the **externalID** (which would have been the case when no cloud provider was being used) to using the cloud provider's **instance-id** (which is what the cloud provider specifies). To resolve this issue:

1. Log in to the CLI as a cluster administrator.
2. Check and back up existing node labels:

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. Delete the nodes:

```
$ oc delete node <node_name>
```

4. On each node host, restart the OpenShift Container Platform service.

```
# systemctl restart atomic-openshift-node
```

5. Add back any [labels on each node](#) that you previously had.

17.7. LABELING CLUSTERS FOR AWS

Starting with OpenShift Container Platform version 3.7 of the **atomic-openshift-installer**, if you configured AWS provider credentials, you must also ensure that all hosts are labeled.

To correctly identify which resources are associated with a cluster, tag resources with the key **kubernetes.io/cluster/<clusterid>**, where:

- **<clusterid>** is a unique name for the cluster.

Set the corresponding value to **owned** if the node belongs exclusively to the cluster or to **shared** if it is a resource shared with other systems.

Tagging all resources with the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag avoids potential issues with multiple zones or multiple clusters.

**NOTE**

In versions prior to OpenShift Container Platform version 3.6, this was **Key=KubernetesCluster, Value=clusterid**.

See [Pods and Services](#) to learn more about labeling and tagging in OpenShift Container Platform.

17.7.1. Resources That Need Tags

There are four types of resources that need to be tagged:

- Instances
- Security Groups
- Load Balancers
- EBS Volumes

17.7.2. Tagging an Existing Cluster

A cluster uses the value of the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag to determine which resources belong to the AWS cluster. This means that all relevant resources must be labeled with the **kubernetes.io/cluster/<clusterid>, Value=(owned|shared)** tag using the same values for that key. These resources include:

- All hosts.
- All relevant load balancers to be used in the AWS instances.
- All EBS volumes. The EBS Volumes that need to be tagged can found with:

```
$ oc get pv -o json|jq '.items[].spec.awsElasticBlockStore.volumeID'
```

- All relevant security groups to be used with the AWS instances.

**NOTE**

Do not tag all existing security groups with the **kubernetes.io/cluster/<name>, Value=<clusterid>** tag, or the Elastic Load Balancing (ELB) will not be able to create a load balancer.

After tagging any resources, restart the master services on the master and the node service on all nodes. See the [Applying Configuration Section](#).

CHAPTER 18. CONFIGURING FOR RED HAT VIRTUALIZATION

You can configure OpenShift Container Platform to use Red Hat Virtualization.

18.1. CONFIGURING RED HAT VIRTUALIZATION OBJECTS

To integrate OpenShift Container Platform with Red Hat Virtualization, take the following actions as part of your [host preparation](#).

1. To provide high availability in case of the loss of one hypervisor host, add each class of instance to a negative affinity group. See [VM Affinity](#).
2. To ensure that the OpenShift Container Platform environment meets the [minimum hardware requirements](#), create templates for virtual machines that use the following resources:

Master nodes	<ul style="list-style-type: none"> • Minimum 2 CPU Cores • 16 GB Memory • Minimum 10 GB root disk • Minimum 15 GB container image registry disk • 30 GB local volume disk • Minimum 25 GB etcd disk
Infrastructure nodes	<ul style="list-style-type: none"> • Minimum 2 CPU Cores • 16 GB Memory • Minimum 10 GB root disk • Minimum 15 GB container image registry disk • 30 GB local volume disk • Minimum 25 GB Gluster registry disk
Application nodes	<ul style="list-style-type: none"> • 2 CPU Cores • 8 GB Memory • Minimum 10 GB root disk • Minimum 15 GB container image registry disk • 30 GB local volume disk

Load balancer node	<ul style="list-style-type: none"> • 1 CPU Core • 4 GB Memory • 10 GB root disk
--------------------	--

3. Create master, infrastructure, and application nodes as well as a load balancer node. Use the templates that you created.
4. Create DNS entries for the routers. Provide entries for all infrastructure instances and configure a round-robin strategy so that the router can pass traffic to applications.
5. Create a DNS entry for the OpenShift Container Platform web console. Specify the IP address of the load balancer node.
6. To use Red Hat Virtualization, you must provide external storage, for example GlusterFS, for persistent storage of registry images and for application storage.

18.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR RED HAT VIRTUALIZATION

You configure OpenShift Container Platform for Red Hat Virtualization by modifying the Ansible inventory file before you install the cluster.

1. Modify the Ansible inventory file, located at `/etc/ansible/hosts` by default, to use the following YAML sections:

```
[OSEv3:children]
nodes
masters
etcd
glusterfs_registry
lb

[OSEv3:vars]
# General variables
ansible_ssh_user=root
openshift_deployment_type=openshift-enterprise
openshift_release='3.10'
openshift_master_cluster_method=native
debug_level=2
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level |
default(debug_level, true) }}"
openshift_enable_service_catalog=False

app_dns_prefix=apps
public_hosted_zone=example.com
load_balancer_hostname=lb.{{public_hosted_zone}}
openshift_master_cluster_hostname="{{ load_balancer_hostname }}"
openshift_master_cluster_public_hostname="{{ load_balancer_hostname
}}"
openshift_master_default_subdomain="{{ app_dns_prefix }}.{{
```

```

public_hosted_zone }}"

# Pod Networking
os_sdn_network_plugin_name=redhat/openshift-ovs-networkpolicy

# Registry
openshift_hosted_registry_storage_kind=glusterfs

# Authentication (example here creates one user, myuser with
password changeme)
openshift_master_identity_providers="[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'True', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]"
openshift_master_htpasswd_users={'myuser':
'$apr1$zAhyA9Ko$rBxB0wAwwtRuuaw80tCwH0'}

# Docker and extra file system setup
container_runtime_docker_storage_setup_device=/dev/vdb
container_runtime_docker_storage_type=overlay2
openshift_docker_use_system_container=False
openshift_node_local_quota_per_fsgroup=512Mi ❶
openshift_use_system_containers=False

[masters]
master0.example.com
master1.example.com
master2.example.com

[etcd]
master0.example.com
master1.example.com
master2.example.com

[infras]
infra0.example.com
infra1.example.com
infra2.example.com

[glusterfs_registry]
infra0.example.com glusterfs_devices="['/dev/vdd']"
infra1.example.com glusterfs_devices="['/dev/vdd']"
infra2.example.com glusterfs_devices="['/dev/vdd']"

[lb]
lb.example.com

[nodes]
master0.example.com openshift_node_group_name=node-config-master
master1.example.com openshift_node_group_name=node-config-master
master2.example.com openshift_node_group_name=node-config-master
infra0.example.com openshift_node_group_name=node-config-infra
infra1.example.com openshift_node_group_name=node-config-infra
infra2.example.com openshift_node_group_name=node-config-infra

```

```
app0.example.com openshift_node_group_name=node-config-compute
app1.example.com openshift_node_group_name=node-config-compute
app2.example.com openshift_node_group_name=node-config-compute
```

- 1 If you use the **openshift_node_local_quota_per_fsgroup** parameter, you must specify the partition or LVM to use for the directory of **/var/lib/origin/openshift.local.volumes**. The partition must be mounted with the option of **gquota** in **fstab**.

This inventory file uses the following nodes and disks:

- One load balancer instance
 - Three master instances
 - Extra disks attached: 15 GB for the container image registry, 30 GB for local volume storage, and 25 GB for etcd
 - Three infrastructure instances
 - Extra disks attached: 15 GB for Docker, 30 GB for local volume storage, and, because this cluster uses GlusterFS for persistent storage, 25 GB for a GlusterFS registry
 - One or more application instance
 - Extra disks attached: 15 GB for Docker, 30 GB for local volume storage
2. Continue to install the cluster following the [Installing OpenShift Container Platform](#) steps. During that process, make any changes to your inventory file that your cluster needs.

CHAPTER 19. CONFIGURING FOR OPENSTACK

19.1. OVERVIEW

When deployed on [OpenStack](#), OpenShift Container Platform can be configured to access the OpenStack infrastructure, including [using OpenStack Cinder volumes as persistent storage](#) for application data.

19.2. PERMISSIONS

Configuring OpenStack for OpenShift Container Platform requires the following role:

Member

For creating assets such as instances, networking ports, floating IPs, volumes, and so on. You need the member role for the tenant.

19.3. CONFIGURING A SECURITY GROUP

When installing OpenShift Container Platform on OpenStack, ensure that you set up the appropriate security groups.

These are some ports that you must have in your security groups, without which the installation fails. You may need more depending on the cluster configuration you want to install. For more information and to adjust your security groups accordingly, see [Required Ports](#) for more information.

All OpenShift Container Platform Hosts	<ul style="list-style-type: none"> • tcp/22 from host running the installer/Ansible
etcd Security Group	<ul style="list-style-type: none"> • tcp/2379 from masters • tcp/2380 from etcd hosts
Master Security Group	<ul style="list-style-type: none"> • tcp/8443 from 0.0.0.0/0 • tcp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 • udp/53 from all OpenShift Container Platform hosts for environments installed prior to or upgraded to 3.2 • tcp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2 • udp/8053 from all OpenShift Container Platform hosts for new environments installed with 3.2
Node Security Group	<ul style="list-style-type: none"> • tcp/10250 from masters • udp/4789 from nodes

Infrastructure Nodes (ones that can host the OpenShift Container Platform router)	<ul style="list-style-type: none"> • tcp/443 from 0.0.0.0/0 • tcp/80 from 0.0.0.0/0
CRI-O	If using CRI-O, you must open tcp/10010 to allow oc exec and oc rsh operations.

If configuring external load-balancers (ELBs) for load balancing the masters and/or routers, you also need to configure Ingress and Egress security groups for the ELBs appropriately.

19.4. CONFIGURING OPENSTACK VARIABLES

To set the required OpenStack variables, create a `/etc/cloud.conf` file with the following contents on all of your OpenShift Container Platform hosts, both masters and nodes:

```
[Global]
auth-url = <OS_AUTH_URL>
username = <OS_USERNAME>
password = <password>
domain-id = <OS_USER_DOMAIN_ID>
tenant-id = <OS_TENANT_ID>
region = <OS_REGION_NAME>

[LoadBalancer]
subnet-id = <UUID of the load balancer subnet>
```

Consult your OpenStack administrators for values of the **OS_** variables, which are commonly used in OpenStack configuration.

19.5. CONFIGURING OPENSIFT CONTAINER PLATFORM MASTERS FOR OPENSTACK

You can set an OpenStack configuration on your OpenShift Container Platform master and node hosts in two different ways:

- [Using Ansible](#)
- Manually, by [modifying the `master-config.yaml` file](#) and the appropriate [node configuration maps](#).

19.5.1. Configuring OpenShift Container Platform for OpenStack with Ansible

During cluster installations, OpenStack can be configured using the following parameters, which are configurable in the [inventory file](#):

- `openshift_cloudprovider_kind`
- `openshift_cloudprovider_openstack_auth_url`
- `openshift_cloudprovider_openstack_username`

- `openshift_cloudprovider_openstack_password`
- `openshift_cloudprovider_openstack_domain_id`
- `openshift_cloudprovider_openstack_domain_name`
- `openshift_cloudprovider_openstack_tenant_id`
- `openshift_cloudprovider_openstack_tenant_name`
- `openshift_cloudprovider_openstack_region`
- `openshift_cloudprovider_openstack_lb_subnet_id`

Example OpenStack Configuration with Ansible

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_openstack_username="{{ lookup('env', 'USERNAME')
}}"
#openshift_cloudprovider_openstack_password="{{ lookup('env', 'PASSWORD')
}}"
#
# Openstack
#openshift_cloudprovider_kind=openstack
#openshift_cloudprovider_openstack_auth_url=http://openstack.example.com:3
5357/v2.0/
#openshift_cloudprovider_openstack_username=username
#openshift_cloudprovider_openstack_password=password
#openshift_cloudprovider_openstack_domain_id=domain_id
#openshift_cloudprovider_openstack_domain_name=domain_name
#openshift_cloudprovider_openstack_tenant_id=tenant_id
#openshift_cloudprovider_openstack_tenant_name=tenant_name
#openshift_cloudprovider_openstack_region=region
#openshift_cloudprovider_openstack_lb_subnet_id=subnet_id
```

19.5.2. Manually Configuring OpenShift Container Platform Masters for OpenStack

Edit or [create](#) the master configuration file on all masters (`/etc/origin/master/master-config.yaml` by default) and update the contents of the `apiServerArguments` and `controllerArguments` sections:

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "openstack"
    cloud-config:
      - "/etc/cloud.conf"
  controllerArguments:
    cloud-provider:
```



```
- "openstack"
cloud-config:
- "/etc/cloud.conf"
```



IMPORTANT

When triggering a containerized installation, only the directories of */etc/origin* and */var/lib/origin* are mounted to the master and node container. Therefore, **cloud.conf** should be in */etc/origin/* instead of */etc/*.

19.5.3. Manually Configuring OpenShift Container Platform Nodes for OpenStack

Edit the appropriate [node configuration map](#) and update the contents of the **kubeletArguments** and **nodeName** sections:

```
nodeName:
  <instance_name> 1

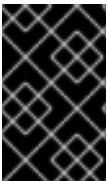
kubeletArguments:
  cloud-provider:
    - "openstack"
  cloud-config:
    - "/etc/cloud.conf"
```

1 The RFC1123-compliant OpenStack instance name of the node host.



NOTE

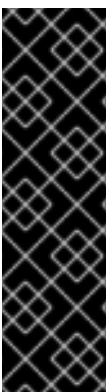
If the **nodeName** does not match the OpenStack instance name, the cloud provider integration will not work.



IMPORTANT

When triggering a containerized installation, only the directories of */etc/origin* and */var/lib/origin* are mounted to the master and node container. Therefore, **cloud.conf** should be in */etc/origin/* instead of */etc/*.

19.5.4. Installing OpenShift Container Platform by Using an Ansible Playbook



IMPORTANT

The OpenStack installation playbook is a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

To install OpenShift Container Platform on an existing OpenStack installation, use the OpenStack playbook. For more information about the playbook, including detailed prerequisites, see [the OpenStack Provisioning readme file](#).

To run the playbook, run the following command:

```
$ ansible-playbook --user openshift \
  -i openshift-ansible/playbooks/openstack/inventory.py \
  -i inventory \
  openshift-ansible/playbooks/openstack/openshift-
  cluster/provision_install.yml
```

19.6. APPLYING CONFIGURATION CHANGES

Start or restart OpenShift Container Platform services on all master and node hosts to apply your configuration changes, see [Restarting OpenShift Container Platform services](#):

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

Switching from not using a cloud provider to using a cloud provider produces an error message. Adding the cloud provider tries to delete the node because the node switches from using the **hostname** as the **externalID** (which would have been the case when no cloud provider was being used) to using the cloud provider's **instance-id** (which is what the cloud provider specifies). To resolve this issue:

1. Log in to the CLI as a cluster administrator.
2. Check and back up existing node labels:

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. Delete the nodes:

```
$ oc delete node <node_name>
```

4. On each node host, restart the OpenShift Container Platform service.

```
# systemctl restart atomic-openshift-node
```

5. Add back any [labels on each node](#) that you previously had.

19.6.1. Configuring Zone Labels for Dynamically Created OpenStack PVs

Administrators can configure zone labels for dynamically created OpenStack PVs. This option is useful if the OpenStack Cinder zone name does not match the compute zone names, for example, if there is only one Cinder zone and many compute zones. Administrators can create Cinder volumes dynamically and then check the labels.

To view the zone labels for the PVs:

```
# oc get pv --show-labels
```

NAME		CAPACITY	ACCESS MODES
RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS
AGE	LABELS		
pvc-1faa6f93-64ac-11e8-930c-fa163e3c373c	1Gi	RW0	
Delete	Bound	openshift-node/pvc1	standard
12s	failure-domain.beta.kubernetes.io/zone=nova		

The default setting is enabled. Using the **oc get pv --show-labels** command returns the **failure-domain.beta.kubernetes.io/zone=nova** label.

To disable the zone label, update the cloud.conf file by adding:

```
[BlockStorage]
ignore-volume-az = yes
```

The PVs created after restarting the master services will not have the zone label.

CHAPTER 20. CONFIGURING FOR GOOGLE COMPUTE ENGINE

You can configure OpenShift Container Platform to access an existing [Google Compute Engine \(GCE\) infrastructure](#), including [using GCE volumes as persistent storage](#) for application data.

20.1. BEFORE YOU BEGIN

20.1.1. Configuring authorization for Google Cloud Platform

Roles

Configuring GCP for OpenShift Container Platform requires the following GCP role:

roles/owner	Needed for creating service accounts, cloud storage, instances, images, templates, Cloud DNS entries, and to deploy load balancers and health checks.
--------------------	---

delete permissions might also be required if the user is expected to redeploy the environment during testing phases.

You can also create a service account to avoid using personal users when deploying GCP objects.

See [the Understanding roles section of the GCP documentation](#) for more information, including steps for how to configure roles.

Scopes and service accounts

GCP uses scopes to determine if an authenticated identity is authorized to perform operations within a resource. For example, if application A with a read-only scope access token can only read, while application B with a read-write scope access token can read and modify data.

The scopes are defined at the GCP API level as <https://www.googleapis.com/auth/compute.readonly>.

You can specify scopes using the `--scopes=[SCOPE,...]` option when creating instances, or you can use the `--no-scopes` option to create the instance without scopes if you don't want the instance accessing the GCP API.

See [the Scopes section of the GCP documentation](#) for more information.

All GCP projects include a default `[PROJECT_NUMBER]-compute@developer.gserviceaccount.com` service account with project editor permissions.

By default, a newly created instance is automatically enabled to run as the default service account with the following access scopes:

- https://www.googleapis.com/auth/devstorage.read_only
- <https://www.googleapis.com/auth/logging.write>
- <https://www.googleapis.com/auth/monitoring.write>

- <https://www.googleapis.com/auth/pubsub>
- <https://www.googleapis.com/auth/service.management.readonly>
- <https://www.googleapis.com/auth/servicecontrol>
- <https://www.googleapis.com/auth/trace.append>
- <https://www.googleapis.com/auth/bigquery>
- <https://www.googleapis.com/auth/cloud-platform>
- <https://www.googleapis.com/auth/compute.readonly>
- <https://www.googleapis.com/auth/compute>
- <https://www.googleapis.com/auth/datastore>
- <https://www.googleapis.com/auth/logging.write>
- <https://www.googleapis.com/auth/monitoring>
- <https://www.googleapis.com/auth/monitoring.write>
- <https://www.googleapis.com/auth/servicecontrol>
- <https://www.googleapis.com/auth/service.management.readonly>
- <https://www.googleapis.com/auth/sqlservice.admin>
- https://www.googleapis.com/auth/devstorage.full_control
- https://www.googleapis.com/auth/devstorage.read_only
- https://www.googleapis.com/auth/devstorage.read_write
- <https://www.googleapis.com/auth/taskqueue>
- <https://www.googleapis.com/auth/userinfo.email>

You can specify another service account with the **--service-account=SERVICE_ACCOUNT** option when creating the instance, or explicitly disabling service accounts for the instance using the **--no-service-account** option using the **gcloud** CLI.

See the [Creating a new service account section of the GCP documentation](#) for more information.

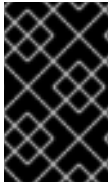
20.1.2. Google Compute Engine objects

Integrating OpenShift Container Platform with Google Compute Engine (GCE) requires the following components or services.

A GCP project

A GCP project is the base level organizing entity that forms the basis for creating, enabling, and using all GCP services. This includes managing APIs, enabling billing, adding and removing collaborators, and managing permissions.

See [the project resource section in the GCP documentation](#) for more information.



IMPORTANT

Project IDs are unique identifiers, and project IDs must be unique across all of Google Cloud Engine. This means you cannot use **myproject** as a project ID if someone else has created a project with that ID before.

Billing

You cannot create new resources unless billing is attached to an account. The new project can be linked to an existing project or new information can be entered.

See [Create, Modify, or Close Your Billing Account in the GCP documentation](#) for more information.

Cloud identity and access management

Deploying OpenShift Container Platform requires the proper permissions. A user must be able to create service accounts, cloud storage, instances, images, templates, Cloud DNS entries, and deploy load balancers and health checks. Delete permissions are also helpful in order to be able to redeploy the environment while testing.

You can create service accounts with specific permissions, then use them to deploy infrastructure components instead of regular users. You can also create roles to limit access to different users or service accounts.

GCP instances use service accounts to allow applications to call GCP APIs. For example, OpenShift Container Platform node hosts can call the GCP disk API to provide a persistent volume to an application.

Access control to the various infrastructure, service resources, and fine-grained roles are available using the IAM service. For more information, see [the Access cloud overview section of the GCP documentation](#).

SSH keys

GCP injects SSH public keys as authorized keys so you can log in using SSH in the created instances. You can configure the SSH keys per instance or per project.

You can use existing SSH keys. GCP metadata can help with storing the SSH keys that are injected at boot time in the instances to allow SSH access.

See [the Metadata section of the GCP documentation](#) for more information.

GCP regions and zones

GCP has a global infrastructure that covers regions and availability zones. While deploying OpenShift Container Platform in GCP on different zones can help avoid single-point-of-failures, there are some caveats regarding storage.

GCP disks are created within a zone. Therefore, if a OpenShift Container Platform node host goes down in zone "A" and the pods move to zone "B", the persistent storage cannot be attached to those pods because the disks are in a different zone.

Deploying a single zone of multizone OpenShift Container Platform environment is an important decision to make before installing OpenShift Container Platform. If deploying a multizone environment, the recommended setup is to use three different zones in a single region.

See [the GCP documentation on regions and zones](#) and [the Kubernetes documentation on multiple zones](#) for more information.

External IP address

So that GCP instances can communicate with the Internet, you must attach an external IP address to the instance. Also, an external IP address is required to communicate with instances deployed in GCP from outside the Virtual Private Cloud (VPC) Network.



WARNING

Requiring an **External IP address** for internet access is a limitation of the provider. You can configure firewall rules to block incoming external traffic in instances if not needed.

See [the GCP documentation on external IP address](#) for more information.

Cloud DNS

GCP cloud DNS is a DNS service used to publish domain names to the global DNS using GCP DNS servers.

The public cloud DNS zone requires a domain name that you purchased either through Google's "Domains" service or through a third-party provider. When you create the zone, you must [add the name servers provided by Google to the registrar](#).

See [the GCP documentation on Cloud DNS](#) for more information.



NOTE

GCP VPC networks have an internal DNS service that automatically resolves internal host names.

The internal fully qualified domain name (FQDN) for an instance follows the **[HOST_NAME].c.[PROJECT_ID].internal** format.

See [the GCP documentation on Internal DNS](#) for more information.

Load balancing

The GCP load balancing service enables the distribution of traffic across multiple instances in the GCP cloud.

There are five types of Load Balancing:

- [Internal](#)
- [Network load balancing](#)
- [HTTP\(S\) load balancing](#)
- [SSL Proxy load balancing](#)
- [TCP Proxy load balancing](#)

**NOTE**

HTTPS and TCP proxy load balancing are the only options for using HTTPS health checks for master nodes, which checks the status of */healthz*.

Because HTTPS load balancing requires a custom certificate, this implementation uses TCP Proxy load balancing to simplify the process.

See [the GCP documentation on Load balancing](#) for more information.

Instances sizes

A successful OpenShift Container Platform environment requires some minimum hardware requirements:

Table 20.1. Instances sizes

Role	Size
Master	n1-standard-8
Node	n1-standard-4

GCP allows you to create custom instance sizes to fit different requirements. See [Creating an Instance with a Custom Machine Type](#) for more information, or see [Machine types](#) and [OpenShift Container Platform Minimum Hardware Requirements](#) for more information about instance sizes.

Storage Options

By default, each GCP instance has a small root persistent disk that contains the operating system. When applications running on the instance require more storage space, you can add additional storage options to the instance:

- Standard persistent disks
- SSD persistent disks
- Local SSDs
- Cloud storage buckets

For more information, see [the GCP documentation on Storage options](#).

20.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR GCE

You can configure OpenShift Container Platform for GCE in two ways:

- [Using Ansible](#)
- [Manually by modifying the *master-config.yaml* file](#)

20.2.1. Option 1: Configuring OpenShift Container Platform for GCP using Ansible

You can configure OpenShift Container Platform for Google Compute Platform (GCP) by modifying the [Ansible inventory file](#) at installation time or after installation.

Procedure

1. At minimum, you must define the **openshift_cloudprovider_kind**, **openshift_gcp_project** and **openshift_gcp_prefix** parameters, as well as the optional **openshift_gcp_multizone** for multizone deployments and **openshift_gcp_network_name** if you are not using the default network name. Add the following section to the Ansible inventory file at installation to configure your OpenShift Container Platform environment for GCP:

```
[OSEv3:vars]
openshift_cloudprovider_kind=gce
openshift_gcp_project=<projectid> ❶
openshift_gcp_prefix=<uid> ❷
openshift_gcp_multizone=False ❸
openshift_gcp_network_name=<network name> ❹
```

- ❶ Provide the GCP project ID where the existing instances are running. This ID is generated when you create the project in the Google Cloud Platform Console.
- ❷ Provide a unique string to identify each OpenShift Container Platform cluster. This must be unique across GCP.
- ❸ Optionally, set to **True** to trigger multizone deployments on GCP. Set to **False** by default.
- ❹ Optionally, provide the network name if not using **default** network.

Installing with Ansible also creates and configures the following files to fit your GCP environment:

- ***/etc/origin/cloudprovider/gce.conf***
 - ***/etc/origin/master/master-config.yaml***
 - ***/etc/origin/node/node-config.yaml***
2. If you are [running load balancer services using GCP](#), the Compute Engine VM node instances require the **ocp** suffix. For example, if the value of the **openshift_gcp_prefix** parameter is set to **mycluster**, you must tag the nodes with **myclusterocp**. See [Adding and Removing Network Tags](#) for more information on how to add network tags to Compute Engine VM instances.
 3. Optionally, you can configure multizone support. The cluster installation process configures single-zone support by default, but you can configure for multiple zones to avoid single-point-of-failures.

Because GCP disks are created within a zone, deploying OpenShift Container Platform in GCP on different zones can cause problems with storage. If an OpenShift Container Platform node host goes down in zone "A" and the pods move to zone "B", the persistent storage cannot be attached to those pods because the disks are now in a different zone. See [Multiple zone limitations](#) in the Kubernetes documentation for more information.

To enable multizone support using the Ansible inventory file, add the following parameter:

```
[OSEv3:vars]
openshift_gcp_multizone=true
```

To return to single-zone support, set the **openshift_gcp_multizone** value to **false** and rerun the Ansible inventory file.

20.2.2. Option 2: Manually configuring OpenShift Container Platform for GCE

20.2.2.1. Manually configuring master hosts for GCE

Perform the following procedure on all master hosts.

Procedure

1. Add the GCE parameters to the **apiServerArguments** and **controllerArguments** sections of the master configuration file at **/etc/origin/master/master-config.yaml** by default:

```
apiServerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
controllerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
```

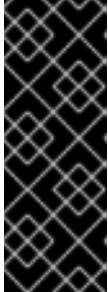
2. When you configure OpenShift Container Platform for GCP using Ansible, the **/etc/origin/cloudprovider/gce.conf** file is created automatically. Because you are manually configuring OpenShift Container Platform for GCP, you must create the file and enter the following:

```
[Global]
project-id = <project-id> ❶
network-name = <network-name> ❷
node-tags = <node-tags> ❸
node-instance-prefix = <instance-prefix> ❹
multizone = true ❺
```

- ❶ Provide the GCP project ID where the existing instances are running.
- ❷ Provide the network name if not using the default.
- ❸ Provide the tag for the GCP nodes. Must contain **ocp** as a suffix. For example, if the value of the **node-instance-prefix** parameter is set to **mycluster**, the nodes must be tagged with **myclusterocp**.
- ❹ Provide a unique string to identify your OpenShift Container Platform cluster.
- ❺ Set to **true** to trigger multizone deployments on GCP. Set to **False** by default.

The cluster installation process configures single-zone support by default.

Deploying OpenShift Container Platform in GCP on different zones can be helpful to avoid single-point-of-failures, but can cause problems with storage. This is because GCP disks are created within a zone. If an OpenShift Container Platform node host goes down in zone "A" and the pods should be moved to zone "B", the persistent storage cannot be attached to those pods, because the disks are now in a different zone. See [Multiple zone limitations](#) in the Kubernetes documentation for more information.



IMPORTANT

For [running load balancer services using GCP](#), the Compute Engine VM node instances require the **ocp** suffix: **<openshift_gcp_prefix>ocp**. For example, if the value of the **openshift_gcp_prefix** parameter is set to **mycluster**, you must tag the nodes with **myclusterocp**. See [Adding and Removing Network Tags](#) for more information on how to add network tags to Compute Engine VM instances.

- Restart the OpenShift Container Platform host services:

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

To return to single-zone support, set the **multizone** value to **false** and restart the master and node host services.

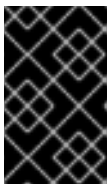
20.2.2.2. Manually configuring node hosts for GCE

Perform the following on all node hosts.

Procedure

- Edit the appropriate [node configuration map](#) and update the contents of the **kubeletArguments** section:

```
kubeletArguments:
  *cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"*
```



IMPORTANT

The **nodeName** must match the instance name in GCP in order for the cloud provider integration to work properly. The name must also be RFC1123 compliant.

- Restart the OpenShift Container Platform services on all nodes.

```
# systemctl restart atomic-openshift-node
```

20.2.3. Configuring the OpenShift Container Platform registry for GCP

Google Cloud Platform (GCP) provides object cloud storage that OpenShift Container Platform can use to store container images using the OpenShift Container Platform container registry.

For more information, see [Cloud Storage in the GCP documentation](#).

Prerequisites

You must create the bucket to host the registry images before the installation. The following commands create a regional bucket using the configured service account:

```
gsutil mb -c regional -l <region> gs://ocp-registry-bucket
cat <<EOF > labels.json
{
  "ocp-cluster": "mycluster"
}
EOF
gsutil label set labels.json gs://ocp-registry-bucket
rm -f labels.json
```



NOTE

A bucket's data is automatically encrypted using a Google-managed key by default. To specify a different key to encrypt the data, see the [Data Encryption Options](#) available in GCP.

See the [Creating storage buckets documentation](#) for more information.

Procedure

To configure the [Ansible inventory file](#) for the registry to use a Google Cloud Storage (GCS) bucket:

```
[OSEv3:vars]
# GCP Provider Configuration
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_replicas=1 ❶
openshift_hosted_registry_storage_gcs_bucket=<bucket_name> ❷
openshift_hosted_registry_storage_gcs_keyfile=<bucket_keyfile> ❸
openshift_hosted_registry_storage_gcs_rootdirectory=<registry_directory> ❹
```

- ❶ The number of replicas to configure.
- ❷ The bucket name to for registry storage.
- ❸ The path on the installer host where the bucket's keyfile is located if you use a custom key file to encrypt the data.
- ❹ Directory used to store the data. **/registry** by default

For more information, see [Cloud Storage in the GCP documentation](#).

20.2.3.1. Manually configuring OpenShift Container Platform registry for GCP

To use GCP object storage, edit the registry's configuration file and mount to the registry pod.

See the [Google Cloud Storage Driver documentation](#) for more information about storage driver configuration files.

Procedure

1. Export the current `/etc/registry/config.yml` file:

```
$ oc get secret registry-config \
  -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. Create a new configuration file from the old `/etc/registry/config.yml` file:

```
$ cp config.yml.old config.yml
```

3. Edit the file to include the GCP parameters. Specify the bucket and keyfile in the **storage** section of a registry's configuration file:

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry 1
    keyfile: mykeyfile 2
```

- 1** Replace with the GCP bucket name.

- 2** A private service account key file in JSON format. If using the Google Application Default Credentials, do not specify the **keyfile** parameter.

4. Delete the **registry-config** secret:

```
$ oc delete secret registry-config -n default
```

5. Recreate the secret to reference the updated configuration file:

```
$ oc create secret generic registry-config \
  --from-file=config.yml -n default
```

6. Redeploy the registry to read the updated configuration:

```
$ oc rollout latest docker-registry -n default
```

20.2.3.1.1. Verify the registry is using GCP object storage

To verify if the registry is using GCP bucket storage:

Procedure

1. After a successful registry deployment using GCP storage, the registry **deploymentconfig** does not show any information if the registry is using an **emptydir** instead of GCP bucket storage:

```
$ oc describe dc docker-registry -n default
...
Mounts:
...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:          EmptyDir 1
...
```

- 1** The temporary directory that shares a pod's lifetime.

2. Check if the **/registry** mountpoint is empty. This is the volume GCP storage will use:

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l
/registry
total 0
```

3. If it is empty, it is because the GCP bucket configuration is performed in the **registry-config** secret:

```
$ oc describe secret registry-config
Name:          registry-config
Namespace:     default
Labels:        <none>
Annotations:   <none>

Type: Opaque

Data
====
config.yml: 398 bytes
```

4. The installer creates a **config.yml** file with the desired configuration using the extended registry capabilities as seen in [Storage in the installation documentation](#). To view the configuration file, including the **storage** section where the storage bucket configuration is stored:

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
```

```

    addr: :5000
  storage:
    delete:
      enabled: true
    cache:
      blobdescriptor: inmemory
    gcs:
      bucket: ocp-registry
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
      - name: openshift
    repository:
      - name: openshift
      options:
        pullthrough: True
        acceptschema2: True
        enforcequota: False
  storage:
    - name: openshift

```

Or you can view the secret:

```

$ oc get secret registry-config -o jsonpath='{.data.config\.yaml}' |
base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
      - name: openshift
    repository:
      - name: openshift
      options:
        pullthrough: True
        acceptschema2: True
        enforcequota: False
  storage:
    - name: openshift

```

You can verify that any image push was successful by viewing **Storage** in the GCP console, then clicking **Browser** and selecting the bucket, or by running the **gsutil** command:

```
$ gsutil ls gs://ocp-registry/
gs://ocp-registry/docker/

$ gsutil du gs://ocp-registry/
7660385      gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d
00a4575720a928db111274e0fbf31b410a093c10/data
7660385      gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d
00a4575720a928db111274e0fbf31b410a093c10/
7660385      gs://ocp-registry/docker/registry/v2/blobs/sha256/03/
...
```

If using an **emptyDir** volume, the **/registry** mountpoint looks similar to the following:

```
$ oc exec \
$(oc get pod -l deploymentconfig=docker-registry \
-o=jsonpath='{.items[0].metadata.name}') -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc         30G  226M   30G   1% /registry

$ oc exec \
$(oc get pod -l deploymentconfig=docker-registry \
-o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

20.2.4. Configuring OpenShift Container Platform to use GCP storage

OpenShift Container Platform can use GCP storage using persistent volumes mechanisms. OpenShift Container Platform creates the disk in GCP and attaches the disk to the correct instance.

GCP disks are **ReadWriteOnce** access mode, which means the volume can be mounted as read-write by a single node. See [the Access modes section of the Architecture guide](#) for more information.

Procedure

1. OpenShift Container Platform creates the following **storageclass** when you use the **gce-pd** provisioner and if you use the **openshift_cloudprovider_kind=gce** and **openshift_gcp_*** variables in the Ansible inventory. Otherwise, if you configured OpenShift Container Platform without using Ansible and the **storageclass** has not been created at installation time, you can create it manually:

```
$ oc get --export storageclass standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
  creationTimestamp: null
  name: standard
  selfLink: /apis/storage.k8s.io/v1/storageclasses/standard
parameters:
```



```
type: pd-standard
provisioner: kubernetes.io/gce-pd
reclaimPolicy: Delete
```

After you request a PV and using the storageclass shown in the previous step, OpenShift Container Platform creates disks in the GCP infrastructure. To verify that the disks were created:

```
$ gcloud compute disks list | grep kubernetes
kubernetes-dynamic-pvc-10ded514-7625-11e8-8c52-42010af00003  us-
west1-b 10 pd-standard READY
```

20.3. USING THE GCP EXTERNAL LOAD BALANCER AS A SERVICE

You can configure OpenShift Container Platform to use the GCP load balancer by exposing services externally using a **LoadBalancer** service. OpenShift Container Platform creates the load balancer in GCP and creates the necessary firewall rules.

Procedure

1. Create a new application:

```
$ oc new-app openshift/hello-openshift
```

2. Expose the load balancer service:

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --
type='LoadBalancer'
```

This command creates a **LoadBalancer** service similar to the following example:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-openshift
    name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer
```

3. To verify that the service has been created:

```
$ oc get svc
NAME                                TYPE                                CLUSTER-IP      <none>
EXTERNAL-IP    PORT(S)                            AGE
hello-openshift    8080/TCP,8888/TCP                ClusterIP        172.30.62.10    20m
hello-openshift-external  35.230.97.224    8080:31521/TCP,8888:30843/TCP  LoadBalancer  172.30.147.214 19m
```

The **LoadBalancer** type and External IP values indicate that the service is using GCP load balancers to expose the application.

OpenShift Container Platform creates the required objects in the GCP infrastructure such as:

- Firewall rules:

```
$ gcloud compute firewall-rules list | grep k8s
k8s-4612931a3a47c204-node-http-hc      my-net    INGRESS    1000
tcp:10256
k8s-fw-a1a8afaa7762811e88c5242010af0000  my-net    INGRESS    1000
tcp:8080,tcp:8888
```



NOTE

These firewall rules are applied to instances tagged with **<openshift_gcp_prefix>ocp**. For example, if the value of the **openshift_gcp_prefix** parameter is set to **mycluster**, you must tag the nodes with **myclusterocp**. See [Adding and Removing Network Tags](#) for more information on how to add network tags to Compute Engine VM instances.

- Health checks:

```
$ gcloud compute http-health-checks list | grep k8s
k8s-4612931a3a47c204-node      10256  /healthz
```

- A load balancer:

```
$ gcloud compute target-pools list | grep k8s
a1a8afaa7762811e88c5242010af0000  us-west1  NONE
k8s-4612931a3a47c204-node
$ gcloud compute forwarding-rules list | grep
a1a8afaa7762811e88c5242010af0000
a1a8afaa7762811e88c5242010af0000  us-west1  35.230.97.224  TCP
us-west1/targetPools/a1a8afaa7762811e88c5242010af0000
```

To verify that the load balancer is properly configured, run the following command from an external host:

```
$ curl 35.230.97.224:8080
Hello OpenShift!
```

CHAPTER 21. CONFIGURING FOR AZURE

You can configure OpenShift Container Platform to use [Microsoft Azure](#) load balancers and [disks for persistent application data](#).

21.1. BEFORE YOU BEGIN

21.1.1. Configuring authorization for Microsoft Azure

Azure roles

Configuring Microsoft Azure for OpenShift Container Platform requires the following Microsoft Azure role:

Contributor	To create and manage all types of Microsoft Azure resources.
-------------	--

See the [Classic subscription administrator roles vs. Azure RBAC roles vs. Azure AD administrator roles documentation](#) for more information.

Permissions

Configuring Microsoft Azure for OpenShift Container Platform requires a service principal, which allows the creation and management of Kubernetes service load balancers and disks for persistent storage. The service principal values are defined at installation time and deployed to the Azure configuration file, located at `/etc/origin/cloudprovider/azure.conf` on OpenShift Container Platform master and node hosts.

Procedure

1. Using the Azure CLI, obtain the account subscription ID:

```
# az account list
[
  {
    "cloudName": "AzureCloud",
    "id": "<subscription>", 1
    "isDefault": false,
    "name": "Pay-As-You-Go",
    "state": "Enabled",
    "tenantId": "<tenant-id>",
    "user": {
      "name": "admin@example.com",
      "type": "user"
    }
  }
]
```

- 1** The subscription ID to use to create the new permissions.

2. Create the service principal with the Microsoft Azure role of contributor and with the scope of the Microsoft Azure subscription and the resource group. Record the output of these values to be used when defining the inventory. Use the `<subscription>` value from the previous step in place of the value below:

```
# az ad sp create-for-rbac --name openshiftcloudprovider \
  --password <secret> --role contributor \
  --scopes
/subscriptions/<subscription>/resourceGroups/<resource-group>

Retrying role assignment creation: 1/36
Retrying role assignment creation: 2/36
{
  "appId": "<app-id>",
  "displayName": "ocpcloudprovider",
  "name": "http://ocpcloudprovider",
  "password": "<secret>",
  "tenant": "<tenant-id>"
}
```

21.1.2. Configuring Microsoft Azure objects

Integrating OpenShift Container Platform with Microsoft Azure requires the following components or services to create a highly-available and full-featured environment.



IMPORTANT

To ensure that the appropriate amount of instances can be launched, request an increase in CPU quota from Microsoft before creating instances.

A resource group

Resource groups contain all Microsoft Azure components for a deployment, including networking, load balancers, virtual machines, and DNS. Quotas and permissions can be applied to resources groups to control and manage resources deployed on Microsoft Azure. Resource groups are created and defined per geographic region. All resources created for an OpenShift Container Platform environment should be within the same geographic region and within the same resource group.

See [Azure Resource Manager overview](#) for more information.

Azure Virtual Networks

Azure Virtual Networks are used to isolate Azure cloud networks from one another. Instances and load balancers use the virtual network to allow communication with each other and to and from the Internet. The virtual network allows for the creation of one or many subnets to be used by components within a resource group. You can also connect virtual networks to various VPN services, allowing communication with on-premise services.

See [What is Azure Virtual Network?](#) for more information.

Azure DNS

Azure offers a managed DNS service that provides internal and Internet-accessible host name and load balancer resolution. The reference environment uses a DNS zone to host three DNS A records to allow for mapping of public IPs to OpenShift Container Platform resources and a bastion.

See [What is Azure DNS?](#) for more information.

Load balancing

Azure load balancers allow network connectivity for scaling and high availability of services running on virtual machines within the Azure environment.

See [What is Azure Load Balancer?](#)

Storage Account

Storage Accounts allow for resources, such as virtual machines, to access the different type of storage components offered by Microsoft Azure. During installation, the storage account defines the location of the object-based **blob** storage used for the OpenShift Container Platform registry.

See [Introduction to Azure Storage](#) for more information, or the [Configuring the OpenShift Container Platform registry for Microsoft Azure section](#) for steps to create the storage account for the registry.

Service Principal

Azure offers the ability to create service accounts, which access, manage, or create components within Azure. The service account grants API access to specific services. For example, a service principal allows Kubernetes or OpenShift Container Platform instances to request persistent storage and load balancers. Service principals allow for granular access to be given to instances or users for specific functions.

See [Application and service principal objects in Azure Active Directory](#) for more information.

Availability Sets

Availability sets ensure that the deployed VMs are distributed across multiple isolated hardware nodes in a cluster. The distribution helps to ensure that when maintenance on the cloud provider hardware occurs, instances will not all run on one specific node.

You should segment instances to different availability sets based on their role. For example, one availability set containing three master hosts, one availability set containing infrastructure hosts, and one availability set containing application hosts. This allows for segmentation and the ability to use external load balancers within OpenShift Container Platform.

See [Manage the availability of Linux virtual machines](#) for more information.

Network Security Groups

Network Security Groups (NSGs) provide a list of rules to either allow or deny traffic to resources deployed within an Azure Virtual Network. NSGs use numeric priority values and rules to define what items are allowed to communicate with each other. You can place restrictions on where communication is allowed to occur, such as within only the virtual network, from load balancers, or from everywhere.

Priority values allow for administrators to grant granular values on the order in which port communication is allowed or not allowed to occur.

See [Plan virtual networks](#) for more information.

Instances sizes

A successful OpenShift Container Platform environment requires some minimum hardware requirements.

See the [Minimum Hardware Requirements section in the OpenShift Container Platform documentation](#) or [Sizes for Cloud Services](#) for more information.

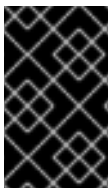
21.2. THE AZURE CONFIGURATION FILE

Configuring OpenShift Container Platform for Azure requires the `/etc/azure/azure.conf` file, on each node host.

If the file does not exist, you can create it.

```
tenantId: <> 1
subscriptionId: <> 2
aadClientId: <> 3
aadClientSecret: <> 4
aadTenantId: <> 5
resourceGroup: <> 6
cloud: <> 7
location: <> 8
vnetName: <> 9
securityGroupName: <> 10
primaryAvailabilitySetName: <> 11
```

- 1 The AAD tenant ID for the subscription that the cluster is deployed in.
- 2 The Azure subscription ID that the cluster is deployed in.
- 3 The client ID for an AAD application with RBAC access to talk to Azure RM APIs.
- 4 The client secret for an AAD application with RBAC access to talk to Azure RM APIs.
- 5 Ensure this is the same as tenant ID (optional).
- 6 The Azure Resource Group name that the Azure VM belongs to.
- 7 The specific cloud region. For example, **AzurePublicCloud**.
- 8 The compact style Azure region. For example, **southeastasia** (optional).
- 9 Virtual network containing instances and used when creating load balancers.
- 10 Security group name associated with instances and load balancers.
- 11 Availability set to use when creating resources such as load balancers (optional).



IMPORTANT

The NIC used for accessing the instance must have an **internal-dns-name** set or the node will not be able to rejoin the cluster, display build logs to the console, and will cause **oc rsh** to not work correctly.

21.3. EXAMPLE INVENTORY FOR OPENSIFT CONTAINER PLATFORM ON MICROSOFT AZURE

The example inventory below assumes that the following items have been created:

- A resource group
- An Azure virtual network
- One or more network security groups that contain the required OpenShift Container Platform ports

- A storage account
- A service principal
- Two load balancers
- Two or more DNS entries for the routers and for the OpenShift Container Platform web console
- Three Availability Sets
- Three master instances
- Three infrastructure instances
- One or more application instances

The inventory below uses the default **storageclass** to create persistent volumes to be used by the metrics, logging, and service catalog components managed by a service principal. The registry uses Microsoft Azure blob storage.

IMPORTANT

If the Microsoft Azure instances use managed disks, provide the following variable in the inventory:

```
openshift_storageclass_parameters={'kind': 'managed',
'storageaccounttype': 'Premium_LRS'}
```

or

```
openshift_storageclass_parameters={'kind': 'managed',
'storageaccounttype': 'Standard_LRS'}
```

This ensures the **storageclass** creates the correct disk type for **PVs** as it relates to the instances deployed. If unmanaged disks are used, the **storageclass** will use the **shared** parameter allowing for unmanaged disks to be created for **PVs**.

```
[OSEv3:children]
masters
etcd
nodes

[OSEv3:vars]
ansible_ssh_user=cloud-user
ansible_become=true
openshift_cloudprovider_kind=azure

#cloudprovider
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=v9c97ead-1v7E-4175-93e3-
623211bed834
openshift_cloudprovider_azure_client_secret=s3r3tR3gistryN0special
openshift_cloudprovider_azure_tenant_id=422r3f91-21fe-4esb-vad5-
d96dfeooee5d
openshift_cloudprovider_azure_subscription_id=6003c1c9-d10d-4366-86cc-
```

```

e3ddddcooe2d
openshift_cloudprovider_azure_resource_group=openshift
openshift_cloudprovider_azure_location=eastus
#endcloudprovider

openshift_master_api_port=443
openshift_master_console_port=443
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-master.example.com
openshift_master_cluster_public_hostname=openshift-master.example.com
openshift_master_default_subdomain=apps.openshift.example.com
openshift_deployment_type=openshift-enterprise
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']], 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?
(memberOf=cn=ose-user,cn=groups,cn=accounts,dc=example,dc=com)'}]}
networkPluginName=redhat/ovs-networkpolicy
openshift_examples_modify_imagestreams=true

# Storage Class change to use managed storage
openshift_storageclass_parameters={'kind': 'managed',
'storageaccounttype': 'Standard_LRS'}

# service catalog
openshift_enable_service_catalog=true
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=SC_STORAGE
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra":
"true"}

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_storage_volume_size=50Gi
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra":

```



```

"true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"}

# Setup azure blob registry storage
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=uZdkVlba6xzwBqK8V
Dz15/loLUoc8I6cPfP31ZS+Q0SxL6ylWT6CLrcadSqvtNTMgztXH4CGjYfVnRNUhvMiA==
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=registry
openshift_hosted_registry_storage_azure_blob_container=registry
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net

[masters]
ocp-master-1
ocp-master-2
ocp-master-3

[etcd]
ocp-master-1
ocp-master-2
ocp-master-3

[nodes]
ocp-master-1 openshift_node_group_name="node-config-master"
ocp-master-2 openshift_node_group_name="node-config-master"
ocp-master-3 openshift_node_group_name="node-config-master"
ocp-infra-1 openshift_node_group_name="node-config-infra"
ocp-infra-2 openshift_node_group_name="node-config-infra"
ocp-infra-3 openshift_node_group_name="node-config-infra"
ocp-app-1 openshift_node_group_name="node-config-compute"

```

21.4. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR MICROSOFT AZURE

You can configure OpenShift Container Platform for Microsoft Azure in two ways:

- [Using Ansible](#)
- [Manually by modifying the *master-config.yaml* file](#)

21.4.1. Configuring OpenShift Container Platform for Azure using Ansible

You can configure OpenShift Container Platform for Azure at installation time or by running the [Ansible inventory file](#) after installation.

Add the following to the Ansible inventory file located at */etc/ansible/hosts* by default to configure your OpenShift Container Platform environment for Microsoft Azure:

```

[OSEv3:vars]
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=<app_ID> ❶
openshift_cloudprovider_azure_client_secret=<secret> ❷
openshift_cloudprovider_azure_tenant_id=<tenant_ID> ❸

```

```

openshift_cloudprovider_azure_subscription_id=<subscription> 4
openshift_cloudprovider_azure_resource_group=<resource_group> 5
openshift_cloudprovider_azure_location=<location> 6

```

- 1 The app ID value for the service principal.
- 2 The secret containing the password for the service principal.
- 3 The tenant in which the service principal exists.
- 4 The subscription used by the service principal.
- 5 The resource group where the service account exists.
- 6 The Microsoft Azure location where the resource group exists.

Installing with Ansible also creates and configures the following files to fit your Microsoft Azure environment:

- */etc/origin/cloudprovider/azure.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

21.4.2. Manually configuring OpenShift Container Platform for Microsoft Azure

21.4.2.1. Manually configuring master hosts for Microsoft Azure

Perform the following on all master hosts.

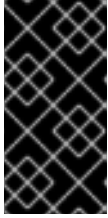
Procedure

1. Edit the master configuration file located at */etc/origin/master/master-config.yaml* by default on all masters and update the contents of the **apiServerArguments** and **controllerArguments** sections:

```

kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/origin/cloudprovider/azure.conf"*
  controllerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/origin/cloudprovider/azure.conf"*

```



IMPORTANT

When triggering a containerized installation, only the */etc/origin* and */var/lib/origin* directories are mounted to the master and node container. Therefore, ensure ***master-config.yaml*** is in the */etc/origin/master* directory instead of */etc/*.

- When you configure OpenShift Container Platform for Microsoft Azure using Ansible, the */etc/origin/cloudprovider/azure.conf* file is created automatically. Because you are manually configuring OpenShift Container Platform for Microsoft Azure, you must create the file on all node instances and include the following:

```
tenantId: <tenant_ID> ❶
subscriptionId: <subscription> ❷
aadClientId: <app_ID> ❸
aadClientSecret: <secret> ❹
aadTenantId: <tenant_ID> ❺
resourceGroup: <resource_group> ❻
location: <location> ❼
```

- ❶ The tenant in which the service principal exists.
- ❷ The subscription used by the service principal.
- ❸ The appID value for the service principal.
- ❹ The secret containing the password for the service principal.
- ❺ The tenant in which the service principal exists.
- ❻ The resource group where the service account exists.
- ❼ The Microsoft Azure location where the resource group exists.

- Restart the OpenShift Container Platform master services:

```
# master-restart api
# master-restart controllers
```

21.4.2.2. Manually configuring node hosts for Microsoft Azure

Perform the following on all node hosts.

Procedure

- Edit the appropriate [node configuration map](#) and update the contents of the **kubeletArguments** section:

```
kubeletArguments:
  cloud-provider:
    - "azure"
  cloud-config:
    - "/etc/origin/cloudprovider/azure.conf"
```



IMPORTANT

The NIC used for accessing the instance must have an internal DNS name set or the node will not be able to rejoin the cluster, display build logs to the console, and will cause **oc rsh** to not work correctly.

- Restart the OpenShift Container Platform services on all nodes:

```
# systemctl restart atomic-openshift-node
```

21.4.3. Configuring the OpenShift Container Platform registry for Microsoft Azure

Microsoft Azure provides object cloud storage that OpenShift Container Platform can use to store container images using the OpenShift Container Platform container registry.

For more information, see [Cloud Storage in the Azure documentation](#).

You can configure the registry either using Ansible or manually by configuring the registry configuration file.

Prerequisites

You must create a storage account to host the registry images before installation. The following command creates a storage account which is used during installation for image storage:

You can use Microsoft Azure blob storage for storing container images. The OpenShift Container Platform registry uses blob storage to allow for the registry to grow dynamically in size without the need for intervention from an administrator.

- Create an Azure storage account:

```
az storage account create
--name <account_name> \
--resource-group <resource_group> \
--location <location> \
--sku Standard_LRS
```

This creates an account key. To view the account key:

```
az storage account keys list \
--account-name <account-name> \
--resource-group <resource-group> \
--output table
```

KeyName	Permissions	Value
key1	Full	<account-key>
key2	Full	<extra-account-key>

Only one account key value is required for the configuration of the OpenShift Container Platform registry.

Option 1: Configuring the OpenShift Container Platform registry for Azure using Ansible

Procedure

- Configure the Ansible inventory for the registry to use the storage account:

```
[OSEv3:vars]
# Azure Registry Configuration
openshift_hosted_registry_replicas=1 ❶
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=
<account_key> ❷
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=
<account_name> ❸
openshift_hosted_registry_storage_azure_blob_container=<registry> ❹
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net
```

- ❶ The number of replicas to configure.
- ❷ The account key associated with the <account-name>.
- ❸ The storage account name.
- ❹ Directory used to store the data. **registry** by default

Option 2: Manually configuring OpenShift Container Platform registry for Microsoft Azure

To use Microsoft Azure object storage, edit the registry's configuration file and mount to the registry pod.

Procedure

1. Export the current **config.yml**:

```
$ oc get secret registry-config \
  -o jsonpath='{.data.config\.yaml}' -n default | base64 -d \
  >> config.yml.old
```

2. Create a new configuration file from the old **config.yml**:

```
$ cp config.yml.old config.yml
```

3. Edit the file to include the Azure parameters:

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: <account-name> ❶
    accountkey: <account-key> ❷
    container: registry ❸
    realm: core.windows.net ❹
```

- ❶ Replace with the storage account name.
- ❷ The account key associated to the <account-name>.

3 Directory used to store the data. **registry** by default

4 Storage realm **core.windows.net** by default

4. Delete the **registry-config** secret:

```
$ oc delete secret registry-config -n default
```

5. Recreate the secret to reference the updated configuration file:

```
$ oc create secret generic registry-config \
  --from-file=config.yml -n default
```

6. Redeploy the registry to read the updated configuration:

```
$ oc rollout latest docker-registry -n default
```

Verifying the registry is using blob object storage

To verify if the registry is using Microsoft Azure blob storage:

Procedure

1. After a successful registry deployment, the registry **deploymentconfig** will always show that the registry is using an **emptydir** instead of Microsoft Azure blob storage:

```
$ oc describe dc docker-registry -n default
...
Mounts:
  ...
  /registry from registry-storage (rw)
Volumes:
  registry-storage:
    Type:          EmptyDir 1
  ...
```

1 The temporary directory that shares a pod's lifetime.

2. Check if the **/registry** mount point is empty. This is the volume Microsoft Azure storage will use:

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l
/registry
total 0
```

3. If it is empty, it is because the Microsoft Azure blob configuration is performed in the **registry-config** secret:

```
$ oc describe secret registry-config
Name:         registry-config
Namespace:    default
```

```
Labels:      <none>
Annotations: <none>

Type: Opaque

Data
====
config.yml: 398 bytes
```

4. The installer creates a **config.yml** file with the desired configuration using the extended registry capabilities as seen in [Storage in the installation documentation](#). To view the configuration file, including the **storage** section where the storage bucket configuration is stored:

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: registry
    accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+QOSxLfo7WT7CLrVPKaqvtNTMgzt
xH7CGjYfpFRNUhvMiA==
    container: registry
    realm: core.windows.net
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
      - name: openshift
    repository:
      - name: openshift
      options:
        pullthrough: True
        acceptschema2: True
        enforcequota: False
  storage:
    - name: openshift
```

Or you can view the secret:

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yaml}' |
base64 -d
version: 0.1
```

```

log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: registry
    accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+Q0SxLfo7WT7CLrVPKaqvtNTMgzt
xH7CGjYfpFRNUhvMiA==
    container: registry
    realm: core.windows.net
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
    options:
      pullthrough: True
      acceptschema2: True
      enforcequota: False
  storage:
    - name: openshift

```

If using an **emptyDir** volume, the **/registry** mountpoint looks like the following:

```

$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        30G   226M   30G   1% /registry

$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 10000000000 10000000000 22 Jun 19 12:24 docker

```

21.4.4. Configuring OpenShift Container Platform to use Microsoft Azure storage

OpenShift Container Platform can use Microsoft Azure storage using persistent volumes mechanisms. OpenShift Container Platform creates the disk in the resource group and attaches the disk to the correct instance.

Procedure

1. The following **storageclass** is created when you configure the Azure cloud provider at installation using the **openshift_cloudprovider_kind=azure** and **openshift_cloud_provider_azure** variables in the Ansible inventory:

```
$ oc get --export storageclass azure-standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
  creationTimestamp: null
  name: azure-standard
parameters:
  kind: Shared
  storageaccounttype: Standard_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

If you did not use Ansible to enable OpenShift Container Platform and Microsoft Azure integration, you can create the **storageclass** manually. See the [xref:\[Dynamic provisioning and creating storage classes section\]](#) for more information.

2. Currently, the default **storageclass** kind is **shared** which means that the Microsoft Azure instances must use unmanaged disks. You can optionally modify this by allowing instances to use managed disks by providing the **openshift_storageclass_parameters={'kind': 'Managed', 'storageaccounttype': 'Premium_LRS'}** or **openshift_storageclass_parameters={'kind': 'Managed', 'storageaccounttype': 'Standard_LRS'}** variables in the Ansible inventory file at installation.



NOTE

Microsoft Azure disks are **ReadWriteOnce** access mode, which means the volume can be mounted as read-write by a single node. See [the Access modes section of the Architecture guide](#) for more information.

21.5. USING THE MICROSOFT AZURE EXTERNAL LOAD BALANCER AS A SERVICE

OpenShift Container Platform can leverage the Microsoft Azure load balancer by exposing services externally using a **LoadBalancer** service. OpenShift Container Platform creates the load balancer in Microsoft Azure and creates the proper firewall rules.



IMPORTANT

Currently, a bug causes extra variables to be included in the Microsoft Azure infrastructure when using it as a cloud provider and when using it as an external load balancer. See the following for more information:

- https://bugzilla.redhat.com/show_bug.cgi?id=1613546

Prerequisites

Ensure the [the Azure configuration file](#) located at `/etc/origin/cloudprovider/azure.conf` is correctly configured with the appropriate objects. See the [Manually configuring OpenShift Container Platform for Microsoft Azure section](#) for an example `/etc/origin/cloudprovider/azure.conf` file.

Once the values are added, restart the OpenShift Container Platform services on all hosts:

```
# systemctl restart atomic-openshift-node
# master-restart api
# master-restart controllers
```

21.5.1. Deploying a sample application using a load balancer

Procedure

1. Create a new application:

```
$ oc new-app openshift/hello-openshift
```

2. Expose the load balancer service:

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --
type='LoadBalancer'
```

This creates a **Loadbalancer** service similar to the following:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-openshift
    name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
    - name: port-1
      nodePort: 30714
      port: 8080
      protocol: TCP
      targetPort: 8080
    - name: port-2
      nodePort: 30122
      port: 8888
      protocol: TCP
      targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer
```

3. Verify that the service has been created:

```
$ oc get svc
```

NAME	EXTERNAL-IP	PORT(S)	TYPE	CLUSTER-IP	AGE
hello-openshift		8080/TCP, 8888/TCP	ClusterIP	172.30.223.255	<none>
hello-openshift-external	40.121.42.180	8080:30714/TCP, 8888:30122/TCP	LoadBalancer	172.30.99.54	4m

The **LoadBalancer** type and **External-IP** fields indicate that the service is using Microsoft Azure load balancers to expose the application.

This creates the following required objects in the Azure infrastructure:

- A load balancer:

```
az network lb list -o table
```

Location	Name	ProvisioningState	ResourceGroup
eastus	kubernetes	Succeeded	refarch-azr
eastus	OcpMasterLB	Succeeded	refarch-azr
eastus	OcpRouterLB	Succeeded	refarch-azr

To verify that the load balancer is properly configured, run the following from an external host:

```
$ curl 40.121.42.180:8080 1
Hello OpenShift!
```

- 1 Replace with the values from the **EXTERNAL-IP** verification step above as well as the port number.

CHAPTER 22. CONFIGURING FOR VMWARE VSPHERE

You can configure OpenShift Container Platform to access [VMware vSphere](#) VMDK Volumes. This includes [using VMware vSphere VMDK Volumes as persistent storage](#) for application data.

The vSphere Cloud Provider allows using vSphere managed storage within OpenShift Container Platform and supports:

- Volumes
- Persistent volumes
- Storage classes and provisioning volumes

22.1. BEFORE YOU BEGIN

22.1.1. VMware vSphere cloud provider prerequisites

Prerequisites

Enabling VMware vSphere requires installing the VMware Tools on each Node VM. See [Installing VMware tools](#) for more information.

Procedure

1. Create a [VM folder](#) and move OpenShift Container Platform Node VMs to this folder.
2. Verify that the Node VM names complies with the regex `[a-z]((?)?[0-9a-z])?(\[a-z0-9\]((\[0-9a-z\])?)?)*`.



IMPORTANT

VM Names cannot:

- Begin with numbers.
- Have any capital letters.
- Have any special characters except -.
- Be shorter than three characters and longer than 63 characters.

3. Set the `disk.EnableUUID` parameter to `true` for each Node VM. This ensures that the VMware vSphere's Virtual Machine Disk (VMDK) always presents a consistent UUID to the VM, allowing the disk to be mounted properly.
For every vSphere virtual machine node that will be participating in the cluster, follow the steps below using the vSphere console:
4. Navigate to **VM properties** → **VM Options** → **Advanced** → **Configuration Parameters** → `disk.enabledUUID=TRUE`
 - a. Set up the GOVC environment:

```
curl -LO
```

```

https://github.com/vmware/govcmomi/releases/download/v0.15.0/govc_
linux_amd64.gz
gunzip govc_linux_amd64.gz
chmod +x govc_linux_amd64
cp govc_linux_amd64 /usr/bin/govc
export GOVC_URL='vCenter IP OR FQDN'
export GOVC_USERNAME='vCenter User'
export GOVC_PASSWORD='vCenter Password'
export GOVC_INSECURE=1

```

- b. Find the Node VM paths:

```
govc ls /datacenter/vm/<vm-folder-name>
```

- c. Set **disk.EnableUUID** to **true** for all VMs:

```
govc vm.change -e="disk.enableUUID=1" -vm='VM Path'
```



NOTE

If OpenShift Container Platform node VMs are created from a template VM, then **disk.EnableUUID=1** can be set on the template VM. VMs cloned from this template inherit this property.

1. Create and assign roles to the vSphere Cloud Provider user and vSphere entities. vSphere Cloud Provider requires the following privileges to interact with vCenter.

Roles	Privileges	Entities	Propagate to Children
manage-k8s-node-vms	Resource.AssignVMT oPool System.Anonymous System.Read System.View VirtualMachine.Config. AddExistingDisk VirtualMachine.Config. AddNewDisk VirtualMachine.Config. AddRemoveDevice VirtualMachine.Config. RemoveDisk VirtualMachine.Invent ory.Create VirtualMachine.Invent ory.Delete	Cluster, Hosts, VM Folder	Yes

Roles	Privileges	Entities	Propagate to Children
manage-k8s-volumes	Datastore.AllocateSpace Datastore.FileManagement System.Anonymous System.Read System.View	Datastore	No
k8s-system-read-and-spbm-profile-view	StorageProfile.View System.Anonymous System.Read System.View	vCenter	No
ReadOnly	System.Anonymous System.Read System.View	Datacenter, Datastore Cluster, Datastore Storage Folder	No

See the [vSphere Documentation Center](#) for steps to create a custom role, user, and role assignment.

22.2. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR VSPHERE

You can configure OpenShift Container Platform for vSphere in two ways:

- [Using Ansible](#)
- [Manually by modifying the *master-config.yaml* file](#)

22.2.1. Option 1: Configuring OpenShift Container Platform for vSphere using Ansible

You can configure OpenShift Container Platform for VMware vSphere (VCP) by modifying the [Ansible inventory file](#) at installation time or after installation.

Procedure

1. Add the following to the Ansible inventory file:

```
[OSEv3:vars]
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username=administrator@vsphere.local
1 openshift_cloudprovider_vsphere_password=<password>
openshift_cloudprovider_vsphere_host=10.x.y.32 2
openshift_cloudprovider_vsphere_datacenter=<Datacenter> 3
openshift_cloudprovider_vsphere_datastore=<Datastore> 4
```

- 1 The user name with the appropriate permissions to create and attach disks in vSphere.

- 2 The vCenter server address.
- 3 The vCenter Datacenter name where the OpenShift Container Platform VMs are located.
- 4 The datastore used for creating VMDKs.

Installing with Ansible also creates and configures the following files to fit your vSphere environment:

- */etc/origin/cloudprovider/vsphere.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

As a reference, a full inventory is shown as follows:

The **openshift_cloudprovider_vsphere_** values are required for OpenShift Container Platform to be able to create **vSphere** resources such as VMDKs on datastores for persistent volumes.

```
$ cat /etc/ansible/hosts

[OSEv3:children]
ansible
masters
infras
apps
etcd
nodes
lb

[OSEv3:vars]
become=yes
ansible_become=yes
ansible_user=root
openshift_release="v3.10"
openshift_version="3.10"
openshift_deployment_type=openshift-enterprise
# Required per https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

VM deployment
openshift_cloudprovider_vsphere_template="rhel75-template"
openshift_cloudprovider_vsphere_vm_network="VM Network"
openshift_cloudprovider_vsphere_vm_netmask="255.255.254.0"
openshift_cloudprovider_vsphere_vm_gateway="10.x.y.254"
openshift_cloudprovider_vsphere_vm_dns="10.x.y.5"

# vSphere Cloud provider
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username="administrator@vsphere.local"
openshift_cloudprovider_vsphere_password="password"
openshift_cloudprovider_vsphere_host="vcsa65-dc1.example.com"
openshift_cloudprovider_vsphere_datacenter=Datacenter
openshift_cloudprovider_vsphere_cluster=Cluster
openshift_cloudprovider_vsphere_resource_pool=ResourcePool
```

```

openshift_cloudprovider_vsphere_datastore="datastore"
openshift_cloudprovider_vsphere_folder="folder"

# Service catalog
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

openshift_master_ldap_ca_file=/home/cloud-user/mycert.crt
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?
(memberOf=cn=ose-user,cn=groups,cn=accounts,dc=openshift,dc=com)'}]]

# Setup vsphere registry storage
openshift_hosted_registry_storage_kind=vsphere
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
openshift_hosted_registry_storage_annotations=
['volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/vsphere-
volume']
openshift_hosted_registry_replicas=1

openshift_hosted_router_replicas=3
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi

default_subdomain=example.com
openshift_master_cluster_hostname=openshift.example.com
openshift_master_cluster_public_hostname=openshift.example.com
openshift_master_default_subdomain=apps.example.com

os_sdn_network_plugin_name='redhat/openshift-ovs-networkpolicy'
osm_use_cockpit=true

# Red Hat subscription name and password
rsub_user=username
rsub_pass=password
rsub_pool=8a85f9815e9b371b015e9b501d081d4b

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=25Gi

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=30Gi
openshift_logging_es_cluster_size=1
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":

```



```

"true"}
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_logging_fluentd_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_logging_storage_kind=dynamic

#registry
openshift_public_hostname=openshift.example.com

[ansible]
localhost

[masters]
master-0.example.com vm_name=master-0 ipv4addr=10.x.y.103
master-1.example.com vm_name=master-1 ipv4addr=10.x.y.104
master-2.example.com vm_name=master-2 ipv4addr=10.x.y.105

[infras]
infra-0.example.com vm_name=infra-0 ipv4addr=10.x.y.100
infra-1.example.com vm_name=infra-1 ipv4addr=10.x.y.101
infra-2.example.com vm_name=infra-2 ipv4addr=10.x.y.102

[apps]
app-0.example.com vm_name=app-0 ipv4addr=10.x.y.106
app-1.example.com vm_name=app-1 ipv4addr=10.x.y.107
app-2.example.com vm_name=app-2 ipv4addr=10.x.y.108

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[lb]
haproxy-0.example.com vm_name=haproxy-0 ipv4addr=10.x.y.200

[nodes]
master-0.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true openshift_hostname=master-0
master-1.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true openshift_hostname=master-1
master-2.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true openshift_hostname=master-2
infra-0.example.com openshift_node_group_name="node-config-infra"
openshift_hostname=infra-0
infra-1.example.com openshift_node_group_name="node-config-infra"
openshift_hostname=infra-1
infra-2.example.com openshift_node_group_name="node-config-infra"
openshift_hostname=infra-2
app-0.example.com openshift_node_group_name="node-config-compute"
openshift_hostname=app-0
app-1.example.com openshift_node_group_name="node-config-compute"

```

```

openshift_hostname=app-1
app-2.example.com openshift_node_group_name="node-config-compute"
openshift_hostname=app-2

```

22.2.2. Option 2: Manually configuring OpenShift Container Platform for vSphere

22.2.2.1. Manually configuring master hosts for vSphere

Perform the following on all master hosts.

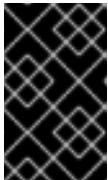
Procedure

1. Edit the master configuration file at `/etc/origin/master/master-config.yaml` by default on all masters and update the contents of the `apiServerArguments` and `controllerArguments` sections:

```

kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"
  controllerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"

```



IMPORTANT

When triggering a containerized installation, only the `/etc/origin` and `/var/lib/origin` directories are mounted to the master and node container. Therefore, **`master-config.yaml`** must be in `/etc/origin/master` rather than `/etc/`.

2. When you configure OpenShift Container Platform for vSphere using Ansible, the `/etc/origin/cloudprovider/vsphere.conf` file is created automatically. Because you are manually configuring OpenShift Container Platform for vSphere, you must create the file and enter the following:

```

[Global] 1
  user = "myusername" 2
  password = "mypassword" 3
  port = "443" 4
  insecure-flag = "1" 5
  datacenter = "mydatacenter" 6

[VirtualCenter "10.10.0.2"] 7
  user = "myvCenterusername"
  password = "password"

[VirtualCenter "10.10.0.3"] 8

```

```

        port = "448"
        insecure-flag = "0"

[Workspace] 9
    server = "10.10.0.2" 10
    datacenter = "mydatacenter"
    folder = "path/to/vms" 11
    datastore = "shared-datastore" 12
    resourcepool-path = "myresourcepoolpath" 13

[Disk]
    scsicontrollertype = pvscsi 14

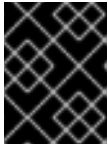
[Network]
    public-network = "VM Network" 15

```

- 1 Any properties set in the **[Global]** section are used for all specified vcenters unless overridden by the settings in the individual **[VirtualCenter]** sections.
- 2 vCenter username for the vSphere cloud provider.
- 3 vCenter password for the specified user.
- 4 Optional. Port number for the vCenter server. Defaults to port **443**.
- 5 Set to **1** if the vCenter uses a self-signed certificate.
- 6 Name of the data center on which Node VMs are deployed.
- 7 Override specific **[Global]** properties for this Virtual Center. Possible setting scan be **[Port]**, **[user]**, **[insecure-flag]**, **[datacenters]**. Any settings not specified are pulled from the **[Global]** section.
- 8 Optional. Additional vCenter server.
- 9 Set any properties used for various vSphere Cloud Provider functionality. For example, dynamic provisioning, Storage Profile Based Volume provisioning, and others.
- 10 IP Address or FQDN for the vCenter server.
- 11 Path to the VM directory for node VMs.
- 12 Set to the name of the datastore to use for provisioning volumes using the storage classes or dynamic provisioning. Prior to OpenShift Container Platform 3.9, if the datastore was located in a storage directory or is a member of a datastore cluster, the full path was required.
- 13 Optional. Set to the path to the resource pool where dummy VMs for Storage Profile Based volume provisioning must be created.
- 14 Type of SCSI controller the VMDK will be attached to the VM as.
- 15 Set to the network port group for vSphere to access the node, which is called VM Network by default. This is the node host's ExternalIP that is registered with Kubernetes.

The cluster installation process configures single-zone or single vCenter by default.

Deploying OpenShift Container Platform in vSphere on different zones can be helpful to avoid single-point-of-failures, but creates the need for shared storage across zones. If an OpenShift Container Platform node host goes down in zone "A" and the pods should be moved to zone "B". See [Multiple zone limitations](#) in the Kubernetes documentation for more information.



IMPORTANT

This ensures that the VMDK always presents a consistent UUID to the VM, allowing the disk to be mounted properly.

For every virtual machine node that will be participating in the cluster: VM properties → VM Options → Advanced → Configuration Parameters → `disk.enabledUUID=TRUE`

Alternatively, the [GOVC tool](#) can be used:

- a. Set up the GOVC environment:

```
export GOVC_URL='vCenter IP OR FQDN'
export GOVC_USERNAME='vCenter User'
export GOVC_PASSWORD='vCenter Password'
export GOVC_INSECURE=1
```

3. Find the Node VM paths:

```
govc ls /datacenter/vm/<vm-folder-name>
```

- a. Set `disk.EnableUUID` to true for all VMs:

```
govc vm.change -e="disk.enableUUID=1" -vm='VM Path'
```



NOTE

If OpenShift Container Platform node VMs are created from a template VM, then **`disk.EnableUUID=1`** can be set on the template VM. VMs cloned from this template inherit this property.

4. Restart the OpenShift Container Platform host services:

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

22.2.2.2. Manually configuring node hosts for vSphere

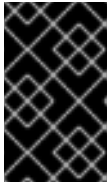
Perform the following on all node hosts.

Procedure

To configure the OpenShift Container Platform nodes for vSphere:

1. Edit the appropriate [node configuration map](#) and update the contents of the `kubeletArguments` section:

```
kubeletArguments:
  cloud-provider:
    - "vsphere"
  cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
```



IMPORTANT

The **nodeName** must match the VM name in vSphere in order for the cloud provider integration to work properly. The name must also be RFC1123 compliant.

2. Restart the OpenShift Container Platform services on all nodes.

```
# systemctl restart atomic-openshift-node
```

22.2.2.3. Applying Configuration Changes

Start or restart OpenShift Container Platform services on all master and node hosts to apply your configuration changes, see [Restarting OpenShift Container Platform services](#):

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

Switching from not using a cloud provider to using a cloud provider produces an error message. Adding the cloud provider tries to delete the node because the node switches from using the **hostname** as the **externalID** (which would have been the case when no cloud provider was being used) to using the cloud provider's **instance-id** (which is what the cloud provider specifies). To resolve this issue:

1. Log in to the CLI as a cluster administrator.
2. Check and back up existing node labels:

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. Delete the nodes:

```
$ oc delete node <node_name>
```

4. On each node host, restart the OpenShift Container Platform service.

```
# systemctl restart atomic-openshift-node
```

5. Add back any [labels on each node](#) that you previously had.

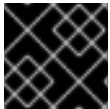
22.2.3. Configuring OpenShift Container Platform to use vSphere storage

OpenShift Container Platform supports VMware vSphere's Virtual Machine Disk (VMDK) volumes. You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [VMware vSphere](#). Some familiarity with Kubernetes and VMware vSphere is assumed.

OpenShift Container Platform creates the disk in vSphere and attaches the disk to the proper instance.

The OpenShift Container Platform [persistent volume \(PV\)](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. vSphere VMDK volumes can be [provisioned dynamically](#).

PVs are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [PV claims](#), however, are specific to a project or namespace and can be requested by users.



IMPORTANT

High availability of storage in the infrastructure is left to the underlying storage provider.

Prerequisites

Before creating PVs using vSphere, ensure your OpenShift Container Platform cluster meets the following requirements:

- OpenShift Container Platform must first be configured for vSphere Cloud Provider
- Each node host in the infrastructure must match the vSphere VM name.
- Each node host must be in the same resource group.

22.2.3.1. Provisioning VMware vSphere volumes

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for vSphere, all that is required for OpenShift Container Platform and vSphere is a VM folder path, file system type, and the **PersistentVolume** API.

22.2.3.1.1. Creating persistent volumes

1. Define a PV object definition, for example ***vsphere-pv.yaml***:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 2Gi ❷
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: ❸
    volumePath: "[datastore1] volumes/myDisk" ❹
    fsType: ext4 ❺
```

- ❶ The name of the volume. This must be how it is identified by [PV claims](#) or from pods.

- 2 The amount of storage allocated to this volume.
- 3 The volume type being used. This example uses **vsphereVolume**, and the label is used to mount a vSphere VMDK volume into pods. The contents of a volume are preserved when it is unmounted. The volume type supports VMFS and VSAN datastore.
- 4 This VMDK volume must exist, and you must include brackets ([]) in the volume definition.
- 5 The file system type to mount. For example, **ext4**, **xfs**, or other file-systems.



IMPORTANT

Changing the value of the **fsType** parameter after the volume is formatted and provisioned can result in data loss and pod failure.

2. Create the PV:

```
$ oc create -f vsphere-pv.yaml
persistentvolume "pv0001" created
```

3. Verify that the PV was created:

```
$ oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM    REASON  AGE
pv0001        <none>    2Gi       RWO          Available         2s
```

Now you can [request storage using PV claims](#), which can now use your PV.



IMPORTANT

PV claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a PV from a different namespace causes the pod to fail.

22.2.3.1.2. Formatting VMware vSphere volumes

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that the volume contains a file system as specified by the **fsType** parameter in the PV definition. If the device is not formatted with the file system, all data from the device is erased, and the device is automatically formatted with the given file system.

This allows unformatted vSphere volumes to be used as PVs, because OpenShift Container Platform formats them before the first use.

22.2.3.2. Provisioning VMware vSphere volumes via a Storage Class

1. OpenShift Container Platform creates the following **storageclass** when you use the **vsphere-volume** provisioner and if you use the **openshift_cloudprovider_kind=vsphere** and **openshift_vsphere_*** variables in the Ansible inventory. Otherwise, you can create it manually:

■

```
$ oc get --export storageclass vsphere-standard -o yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: "vsphere-standard" ❶
provisioner: kubernetes.io/vsphere-volume ❷
parameters:
  diskformat: zeroedthick ❸
  datastore: "ose3-vmware" ❹
reclaimPolicy: Delete
```

- ❶ The name of the storage class.
- ❷ The type of storage provisioner: vsphere-volume
- ❸ The type of disk: zeroedthick, thin.
- ❹ The source datastore where the disk will be created.

2. After you request a PV and using the storageclass shown in the previous step, OpenShift Container Platform creates VMDK disks in the vSphere infrastructure. To verify that the disks were created:

```
$ ls /vmfs/volumes/ose3-vmware/kubevols | grep kubernetes
kubernetes-dynamic-pvc-790615e8-a22a-11e8-bc85-0050568e2982.vmdk
```



NOTE

vSphere-volume disks are **ReadWriteOnce** access mode, which means the volume can be mounted as read-write by a single node. See [the Access modes section of the Architecture guide](#) for more information.

22.2.4. Configuring the OpenShift Container Platform registry for vSphere

The following steps define the manual process of storage creation, which is used to create storage for the registry if a storage class is unavailable or not used.

```
# VMFS
cd /vmfs/volumes/datastore1/
mkdir kubevols # Not needed but good hygiene

# VSAN
cd /vmfs/volumes/vsanDatastore/
/usr/lib/vmware/osfs/bin/osfs-mkdir kubevols # Needed

cd kubevols

vmkfstools -c 25G registry.vmdk
```

22.2.4.1. Configuring the OpenShift Container Platform registry for vSphere using Ansible

Procedure

To configure the Ansible inventory for the registry to use a vSphere volume:

```
[OSEv3:vars]
# vSphere Provider Configuration
openshift_hosted_registry_storage_kind=vsphere ❶
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce'] ❷
openshift_hosted_registry_storage_annotations=[
  'volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/vsphere-
  volume' ] ❸
openshift_hosted_registry_replicas=1 ❹
```

- ❶ The storage type.
- ❷ vSphere volumes only support **RWO**.
- ❸ The annotation for the volume.
- ❹ The number of replicas to configure.



NOTE

The brackets in the configuration file above are required.

22.2.4.2. Manually configuring OpenShift Container Platform registry for vSphere

To use vSphere volume storage, edit the registry's configuration file and mount to the registry pod.

Procedure

1. Create a new configuration file from the vSphere volume:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: vsphere-registry-storage
  annotations:
    volume.beta.kubernetes.io/storage-class: vsphere-standard
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 30Gi
```

2. Create the file in OpenShift Container Platform:

```
$ oc create -f pvc-registry.yaml
```

3. Update the volume configuration to use the new PVC:

```
$ oc volume dc docker-registry --add --name=registry-storage -t \
pvc --claim-name=vsphere-registry-storage --overwrite
```

4. Redeploy the registry to read the updated configuration:

```
$ oc rollout latest docker-registry -n default
```

5. Verify the volume has been assigned:

```
$ oc volume dc docker-registry -n default
```

22.3. BACKUP OF PERSISTENT VOLUMES

OpenShift Container Platform provisions new volumes as **independent persistent disks** to freely attach and detach the volume on any node in the cluster. As a consequence, [it is not possible to back up volumes that use snapshots](#).

To create a backup of PVs:

1. Stop the application using the PV.
2. Clone the persistent disk.
3. Restart the application.
4. Create a backup of the cloned disk.
5. Delete the cloned disk.

CHAPTER 23. CONFIGURING LOCAL VOLUMES

23.1. OVERVIEW

OpenShift Container Platform can be configured to access [local volumes](#) for application data.

Local volumes are persistent volumes (PV) that represent locally-mounted file systems. As of OpenShift Container Platform 3.10, this also includes raw block devices. A raw device offers a more direct route to the physical device and allows an application more control over the timing of I/O operations to that physical device. This makes raw devices suitable for complex applications such as database management systems that typically do their own caching. Local volumes have a few unique features. Any pod that uses a local volume PV is scheduled on the node where the local volume is mounted.

In addition, local volumes include a provisioner that automatically creates PVs for locally-mounted devices. This provisioner currently scans only pre-configured directories. This provisioner cannot dynamically provision volumes, but this feature might be implemented in a future release.

The local volume provisioner allows using local storage within OpenShift Container Platform and supports:

- Volumes
- PVs



IMPORTANT

Local volumes is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

23.2. MOUNTING LOCAL VOLUMES



NOTE

All local volumes must be manually mounted before they can be consumed by OpenShift Container Platform as PVs.

To mount local volumes:

1. Mount all volumes into the `/mnt/local-storage/<storage-class-name>/<volume>` path. Administrators must create local devices as needed using any method such as disk partition or LVM, create suitable file systems on these devices, and mount these devices using a script or `/etc/fstab` entries, for example:

```
# device name      # mount point          # FS      # options #
extra
/dev/sdb1          /mnt/local-storage/ssd/disk1 ext4      defaults 1 2
/dev/sdb2          /mnt/local-storage/ssd/disk2 ext4      defaults 1 2
```

```

/dev/sdb3      /mnt/local-storage/ssd/disk3 ext4      defaults 1 2
/dev/sdc1      /mnt/local-storage/hdd/disk1 ext4      defaults 1 2
/dev/sdc2      /mnt/local-storage/hdd/disk2 ext4      defaults 1 2

```

2. Make all volumes accessible to the processes running within the Docker containers. You can change the labels of mounted file systems to allow this, for example:

```

---
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-
storage/
---

```

23.3. CONFIGURING THE LOCAL PROVISIONER

OpenShift Container Platform depends on an external provisioner to create PVs for local devices and to clean up PVs when they are not in use to enable reuse.



NOTE

- The local volume provisioner is different from most provisioners and does not support dynamic provisioning.
- The local volume provisioner requires administrators to preconfigure the local volumes on each node and mount them under discovery directories. The provisioner then manages the volumes by creating and cleaning up PVs for each volume.

To configure the local provisioner:

1. Configure the external provisioner using a ConfigMap to relate directories with storage classes. This configuration must be created before the provisioner is deployed, for example:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
  storageClassMap: |
    local-ssd: ❶
                hostDir: /mnt/local-storage/ssd ❷
                mountDir: /mnt/local-storage/ssd ❸
    local-hdd:
                hostDir: /mnt/local-storage/hdd
                mountDir: /mnt/local-storage/hdd

```

- ❶ Name of the storage class.
- ❷ Path to the directory on the host. It must be a subdirectory of **/mnt/local-storage**.
- ❸ Path to the directory in the provisioner pod. We recommend using the same directory structure as used on the host and **mountDir** can be omitted in this case.

2. (Optional) Create a standalone namespace for the local volume provisioner and its configuration, for example: **oc new-project local-storage**.

With this configuration, the provisioner creates:

- One PV with storage class **local-ssd** for every subdirectory mounted in the **/mnt/local-storage/ssd** directory
- One PV with storage class **local-hdd** for every subdirectory mounted in the **/mnt/local-storage/hdd** directory



WARNING

The syntax of the ConfigMap has changed between OpenShift Container Platform 3.9 and 3.10. Since this feature is in Technology Preview, the ConfigMap is not automatically converted during the update.

23.4. DEPLOYING THE LOCAL PROVISIONER



NOTE

Before starting the provisioner, mount all local devices and create a ConfigMap with storage classes and their directories.

To deploy the local provisioner:

1. Install the local provisioner from the [local-storage-provisioner-template.yaml](#) file.
2. Create a service account that allows running pods as a root user, using hostPath volumes, and using any SELinux context to monitor, manage, and clean local volumes:

```
$ oc create serviceaccount local-storage-admin
$ oc adm policy add-scc-to-user privileged -z local-storage-admin
```

To allow the provisioner pod to delete content on local volumes created by any pod, root privileges and any SELinux context are required. hostPath is required to access the **/mnt/local-storage** path on the host.

3. Install the template:

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/release-
3.10/examples/storage-examples/local-examples/local-storage-
provisioner-template.yaml
```

4. Instantiate the template by specifying values for the **CONFIGMAP**, **SERVICE_ACCOUNT**, **NAMESPACE**, and **PROVISIONER_IMAGE** parameters:

```
$ oc new-app -p CONFIGMAP=local-volume-config \
```

```
-p SERVICE_ACCOUNT=local-storage-admin \
-p NAMESPACE=local-storage \
-p PROVISIONER_IMAGE=registry.access.redhat.com/openshift3/local-
storage-provisioner:v3.9 \ 1
local-storage-provisioner
```

1 Provide your OpenShift Container Platform version number, such as **v3.10**.

5. Add the necessary storage classes:

```
$ oc create -f ./storage-class-ssd.yaml
$ oc create -f ./storage-class-hdd.yaml
```

For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-ssd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-hdd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

See the [local storage provisioner template](#) for other configurable options. This template creates a DaemonSet that runs a pod on every node. The pod watches the directories that are specified in the ConfigMap and automatically creates PVs for them.

The provisioner runs with root permissions because it removes all data from the modified directories when a PV is released.

23.5. ADDING NEW DEVICES

Adding a new device is semi-automatic. The provisioner periodically checks for new mounts in configured directories. Administrators must create a new subdirectory, mount a device, and allow pods to use the device by applying the SELinux label, for example:

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-
storage/
```



IMPORTANT

Omitting any of these steps may result in the wrong PV being created.

23.6. CONFIGURING RAW BLOCK DEVICES

It is possible to statically provision raw block devices using the local volume provisioner. This feature is disabled by default and requires additional configuration.

To configure raw block devices:

1. Enable the **BlockVolume** feature gate on all masters. Edit or create the master configuration file on all masters (`/etc/origin/master/master-config.yaml` by default) and add **BlockVolume=true** under the **apiServerArguments** and **controllerArguments** sections:

```
apiServerArguments:
  feature-gates:
    - BlockVolume=true
...

controllerArguments:
  feature-gates:
    - BlockVolume=true
...
```

2. Enable the feature gate on all nodes by editing the node configuration ConfigMap:

```
$ oc edit configmap node-config-compute --namespace openshift-node
$ oc edit configmap node-config-master --namespace openshift-node
$ oc edit configmap node-config-infra --namespace openshift-node
```

3. Ensure that all ConfigMaps contain **BlockVolume=true** in the feature gates array of the **kubeletArguments**, for example:

node configmap feature-gates setting

```
kubeletArguments:
  feature-gates:
    -
    RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true,BlockVolume=true
```

4. Restart the master. The nodes restart automatically after the configuration change. This may take several minutes.

23.6.1. Preparing raw block devices

Before you start the provisioner, link all the raw block devices that pods can use to the `/mnt/local-storage/<storage class>` directory structure. For example, to make directory `/dev/dm-36` available:

1. Create a directory for the device's storage class in `/mnt/local-storage`:

```
$ mkdir -p /mnt/local-storage/block-devices
```

2. Create a symbolic link that points to the device:

```
$ ln -s /dev/dm-36 dm-uuid-LVM-1234
```

**NOTE**

To avoid possible name conflicts, use the same name for the symbolic link and the link from the `/dev/disk/by-uuid` or `/dev/disk/by-id` directory .

3. Create or update the ConfigMap that configures the provisioner:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
  storageClassMap: |
    block-devices: 1
      hostDir: /mnt/local-storage/block-devices 2
      mountDir: /mnt/local-storage/block-devices 3
```

- 1** Name of the storage class.
- 2** Path to the directory on the host. It must be a subdirectory of `/mnt/local-storage`.
- 3** Path to the directory in the provisioner pod. If you use the directory structure that the host uses, which is recommended, omit the `mountDir` parameter.

4. Change the **SELinux** label of the device and the `/mnt/local-storage/`:

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
$ chcon unconfined_u:object_r:svirt_sandbox_file_t:s0 /dev/dm-36
```

5. Create a storage class for the raw block devices:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: block-devices
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

The block device `/dev/dm-36` is now ready to be used by the provisioner and provisioned as a PV.

23.6.2. Deploying raw block device provisioners

Deploying the provisioner for raw block devices is similar to deploying the provisioner on local volumes. There are two differences:

1. The provisioner must run in a privileged container.
2. The provisioner must have access to the `/dev` file system from the host.

To deploy the provisioner for raw block devices:

1. Download the template from the [local-storage-provisioner-template.yaml](#) file.

2. Edit the template:

- a. Set the **privileged** attribute of the **securityContext** of the container spec to **true**:

```
...
  containers:
  ...
    name: provisioner
  ...
    securityContext:
      privileged: true
  ...
```

- b. Mount the host **/dev** file system to the container using **hostPath**:

```
...
  containers:
  ...
    name: provisioner
  ...
    volumeMounts:
    - mountPath: /dev
      name: dev
  ...
  volumes:
  - hostPath:
    path: /dev
    name: dev
  ...
```

3. Create the template from the modified YAML file:

```
$ oc create -f local-storage-provisioner-template.yaml
```

4. Start the provisioner:

```
$ oc new-app -p CONFIGMAP=local-volume-config \
  -p SERVICE_ACCOUNT=local-storage-admin \
  -p NAMESPACE=local-storage \
  -p
  PROVISIONER_IMAGE=registry.access.redhat.com/openshift3/local-
  storage-provisioner:v3.10 \
  local-storage-provisioner
```

23.6.3. Using raw block device persistent volumes

To use the raw block device in the pod, create a persistent volume claim (PVC) with **volumeMode**: set to **Block** and **storageClassName** set to **block-devices**, for example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
```

```
storageClassName: block-devices
accessModes:
  - ReadWriteOnce
volumeMode: Block
resources:
  requests:
    storage: 1Gi
```

Pod using the raw block device PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-test
  labels:
    name: busybox-test
spec:
  restartPolicy: Never
  containers:
    - resources:
        limits :
          cpu: 0.5
      image: gcr.io/google_containers/busybox
      command:
        - "/bin/sh"
        - "-c"
        - "while true; do date; sleep 1; done"
      name: busybox
      volumeDevices:
        - name: vol
          devicePath: /dev/xvda
  volumes:
    - name: vol
      persistentVolumeClaim:
        claimName: block-pvc
```



NOTE

The volume is not mounted in the pod but is exposed as the **/dev/xvda** raw block device.

CHAPTER 24. CONFIGURING PERSISTENT STORAGE

24.1. OVERVIEW

The Kubernetes [persistent volume](#) framework allows you to provision an OpenShift Container Platform cluster with persistent storage using networked storage available in your environment. This can be done after completing the initial OpenShift Container Platform installation depending on your application needs, giving users a way to request those resources without having any knowledge of the underlying infrastructure.

These topics show how to configure persistent volumes in OpenShift Container Platform using the following supported volume plug-ins:

- [NFS](#)
- [GlusterFS](#)
- [OpenStack Cinder](#)
- [Ceph RBD](#)
- [AWS Elastic Block Store \(EBS\)](#)
- [GCE Persistent Disk](#)
- [iSCSI](#)
- [Fibre Channel](#)
- [Azure Disk](#)
- [Azure File](#)
- [FlexVolume](#)
- [VMWare vSphere](#)
- [Container Storage Interface \(CSI\)](#)
- [Dynamic Provisioning and Creating Storage Classes](#)
- [Volume Security](#)
- [Selector-Label Volume Binding](#)

24.2. PERSISTENT STORAGE USING NFS

24.2.1. Overview

OpenShift Container Platform clusters can be provisioned with [persistent storage](#) using NFS. Persistent volumes (PVs) and persistent volume claims (PVCs) provide a convenient method for sharing a volume across a project. While the NFS-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

This topic covers the specifics of using the NFS persistent storage type. Some familiarity with OpenShift Container Platform and [NFS](#) is beneficial. See the [Persistent Storage](#) concept topic for details on the OpenShift Container Platform persistent volume (PV) framework in general.

24.2.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. To provision NFS volumes, a list of NFS servers and export paths are all that is required.

You must first create an object definition for the PV:

Example 24.1. PV Object Definition Using NFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
    - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Recycle 7
```

- 1 The name of the volume. This is the PV identity in various **oc <command> pod** commands.
- 2 The amount of storage allocated to this volume.
- 3 Though this appears to be related to controlling access to the volume, it is actually used similarly to labels and used to match a PVC to a PV. Currently, no access rules are enforced based on the **accessModes**.
- 4 The volume type being used, in this case the **nfs** plug-in.
- 5 The path that is exported by the NFS server.
- 6 The host name or IP address of the NFS server.
- 7 The reclaim policy for the PV. This defines what happens to a volume when released from its claim. Valid options are **Retain** (default) and **Recycle**. See [Reclaiming Resources](#).



NOTE

Each NFS volume must be mountable by all schedulable nodes in the cluster.

Save the definition to a file, for example **nfs-pv.yaml**, and create the PV:

```
$ oc create -f nfs-pv.yaml
persistentvolume "pv0001" created
```

Verify that the PV was created:

```
# oc get pv
NAME                                LABELS            CAPACITY          ACCESSMODES        STATUS
CLAIM          REASON          AGE
pv0001                                     <none>           5368709120        RWO                Available
31s
```

The next step can be to create a PVC, which binds to the new PV:

Example 24.2. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce ❶
  resources:
    requests:
      storage: 1Gi ❷
```

- ❶ As mentioned above for PVs, the **accessModes** do not enforce security, but rather act as labels to match a PV to a PVC.
- ❷ This claim looks for PVs offering **1Gi** or greater capacity.

Save the definition to a file, for example **nfs-claim.yaml**, and create the PVC:

```
# oc create -f nfs-claim.yaml
```

24.2.3. Enforcing Disk Quotas

You can use disk partitions to enforce disk quotas and size constraints. Each partition can be its own export. Each export is one PV. OpenShift Container Platform enforces unique names for PVs, but the uniqueness of the NFS volume's server and path is up to the administrator.

Enforcing quotas in this way allows the developer to request persistent storage by a specific amount (for example, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

24.2.4. NFS Volume Security

This section covers NFS volume security, including matching permissions and SELinux considerations. The user is expected to understand the basics of POSIX permissions, process UIDs, supplemental groups, and SELinux.

**NOTE**

See the full [Volume Security](#) topic before implementing NFS volumes.

Developers request NFS storage by referencing, in the **volumes** section of their pod definition, either a PVC by name or the NFS volume plug-in directly.

The **/etc/exports** file on the NFS server contains the accessible NFS directories. The target NFS directory has POSIX owner and group IDs. The OpenShift Container Platform NFS plug-in mounts the container's NFS directory with the same POSIX ownership and permissions found on the exported NFS directory. However, the container is not run with its effective UID equal to the owner of the NFS mount, which is the desired behavior.

As an example, if the target NFS directory appears on the NFS server as:

```
# ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0 /opt/nfs

# id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

Then the container must match SELinux labels, and either run with a UID of **65534** (**nfsnobody** owner) or with **5555** in its supplemental groups in order to access the directory.

**NOTE**

The owner ID of 65534 is used as an example. Even though NFS's **root_squash** maps **root** (0) to **nfsnobody** (65534), NFS exports can have arbitrary owner IDs. Owner 65534 is not required for NFS exports.

24.2.4.1. Group IDs

The recommended way to handle NFS access (assuming it is not an option to change permissions on the NFS export) is to use supplemental groups. Supplemental groups in OpenShift Container Platform are used for shared storage, of which NFS is an example. In contrast, block storage, such as Ceph RBD or iSCSI, use the **fsGroup** SCC strategy and the **fsGroup** value in the pod's **securityContext**.

**NOTE**

It is generally preferable to use supplemental group IDs to gain access to persistent storage versus using [user IDs](#). Supplemental groups are covered further in the full [Volume Security](#) topic.

Because the group ID on the [example target NFS directory](#) shown above is 5555, the pod can define that group ID using **supplementalGroups** under the pod-level **securityContext** definition. For example:

```
spec:
  containers:
    - name:
      ...
  securityContext: ❶
    supplementalGroups: [5555] ❷
```

- 1 **securityContext** must be defined at the pod level, not under a specific container.
- 2 An array of GIDs defined for the pod. In this case, there is one element in the array; additional GIDs would be comma-separated.

Assuming there are no custom SCCs that might satisfy the pod's requirements, the pod likely matches the **restricted** SCC. This SCC has the **supplementalGroups** strategy set to **RunAsAny**, meaning that any supplied group ID is accepted without range checking.

As a result, the above pod passes admissions and is launched. However, if group ID range checking is desired, a custom SCC, as described in [pod security and custom SCCs](#), is the preferred solution. A custom SCC can be created such that minimum and maximum group IDs are defined, group ID range checking is enforced, and a group ID of 5555 is allowed.



NOTE

To use a custom SCC, you must first add it to the appropriate service account. For example, use the **default** service account in the given project unless another has been specified on the pod specification. See [Add an SCC to a User, Group, or Project](#) for details.

24.2.4.2. User IDs

User IDs can be defined in the container image or in the pod definition. The full [Volume Security](#) topic covers controlling storage access based on user IDs, and should be read prior to setting up NFS persistent storage.



NOTE

It is generally preferable to use [supplemental group IDs](#) to gain access to persistent storage versus using user IDs.

In the [example target NFS directory](#) shown above, the container needs its UID set to 65534 (ignoring group IDs for the moment), so the following can be added to the pod definition:

```
spec:
  containers: 1
  - name:
    ...
    securityContext:
      runAsUser: 65534 2
```

- 1 Pods contain a **securityContext** specific to each container (shown here) and a pod-level **securityContext** which applies to all containers defined in the pod.
- 2 65534 is the **nfsnobody** user.

Assuming the **default** project and the **restricted** SCC, the pod's requested user ID of 65534 is not allowed, and therefore the pod fails. The pod fails for the following reasons:

- It requests 65534 as its user ID.

- All SCCs available to the pod are examined to see which SCC allows a user ID of 65534 (actually, all policies of the SCCs are checked but the focus here is on user ID).
- Because all available SCCs use **MustRunAsRange** for their **runAsUser** strategy, UID range checking is required.
- 65534 is not included in the SCC or project's user ID range.

It is generally considered a good practice not to modify the predefined SCCs. The preferred way to fix this situation is to create a custom SCC, as described in the full [Volume Security](#) topic. A custom SCC can be created such that minimum and maximum user IDs are defined, UID range checking is still enforced, and the UID of 65534 is allowed.



NOTE

To use a custom SCC, you must first add it to the appropriate service account. For example, use the **default** service account in the given project unless another has been specified on the pod specification. See [Add an SCC to a User, Group, or Project](#) for details.

24.2.4.3. SELinux



NOTE

See the full [Volume Security](#) topic for information on controlling storage access in conjunction with using SELinux.

By default, SELinux does not allow writing from a pod to a remote NFS server. The NFS volume mounts correctly, but is read-only.

To enable writing to NFS volumes with SELinux enforcing on each node, run:

```
# setsebool -P virt_use_nfs 1
```

The **-P** option above makes the bool persistent between reboots.

The **virt_use_nfs** boolean is defined by the **docker-selinux** package. If an error is seen indicating that this bool is not defined, ensure this package has been installed.

24.2.4.4. Export Settings

In order to enable arbitrary container users to read and write the volume, each exported volume on the NFS server should conform to the following conditions:

- Each export must be:

```
/<example_fs> *(rw,root_squash)
```

- The firewall must be configured to allow traffic to the mount point.
 - For NFSv4, configure the default port **2049 (nfs)** and port **111 (portmapper)**.

NFSv4

■


```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- For NFSv3, there are three ports to configure: **2049** (**nfs**), **20048** (**mountd**), and **111** (**portmapper**).

NFSv3

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- The NFS export and directory must be set up so that it is accessible by the target pods. Either set the export to be owned by the container's primary UID, or supply the pod group access using **supplementalGroups**, as shown in [Group IDs](#) above. See the full [Volume Security](#) topic for additional pod security information as well.

24.2.5. Reclaiming Resources

NFS implements the OpenShift Container Platform **Recyclable** plug-in interface. Automatic processes handle reclamation tasks based on policies set on each persistent volume.

By default, PVs are set to **Retain**. NFS volumes which are set to **Recycle** are scrubbed (i.e., **rm -rf** is run on the volume) after being released from their claim (i.e, after the user's **PersistentVolumeClaim** bound to the volume is deleted). Once recycled, the NFS volume can be bound to a new claim.

Once claim to a PV is released (that is, the PVC is deleted), the PV object should not be re-used. Instead, a new PV should be created with the same basic volume details as the original.

For example, the administrator creates a PV named **nfs1**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

The user creates **PVC1**, which binds to **nfs1**. The user then deletes **PVC1**, releasing claim to **nfs1**, which causes **nfs1** to be **Released**. If the administrator wishes to make the same NFS share available, they should create a new PV with the same NFS server details, but a different PV name:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
```

```

storage: 1Mi
accessModes:
  - ReadWriteMany
nfs:
  server: 192.168.1.1
  path: "/"

```

Deleting the original PV and re-creating it with the same name is discouraged. Attempting to manually change the status of a PV from **Released** to **Available** causes errors and potential data loss.



NOTE

A PV with retention policy of **Recycle** scrubs (**rm -rf**) the data and marks it as **Available** for claim. The **Recycle** retention policy is deprecated starting in OpenShift Container Platform 3.6 and should be avoided. Anyone using recycler should use dynamic provision and volume deletion instead.

24.2.6. Automation

Clusters can be provisioned with persistent storage using NFS in the following ways:

- [Enforce storage quotas](#) using disk partitions.
- Enforce security by [restricting volumes](#) to the project that has a claim to them.
- Configure [reclamation of discarded resources](#) for each PV.

There are many ways that you can use scripts to automate the above tasks. You can use an [example Ansible playbook](#) to help you get started.

24.2.7. Additional Configuration and Troubleshooting

Depending on what version of NFS is being used and how it is configured, there may be additional configuration steps needed for proper export and security mapping. The following are some that may apply:

NFSv4 mount incorrectly shows all files with ownership of nobody:nobody	<ul style="list-style-type: none"> • Could be attributed to the ID mapping settings (/etc/idmapd.conf) on your NFS • See this Red Hat Solution.
Disabling ID mapping on NFSv4	<ul style="list-style-type: none"> • On both the NFS client and server, run: <pre># echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable _idmapping</pre>

24.3. PERSISTENT STORAGE USING RED HAT GLUSTER STORAGE

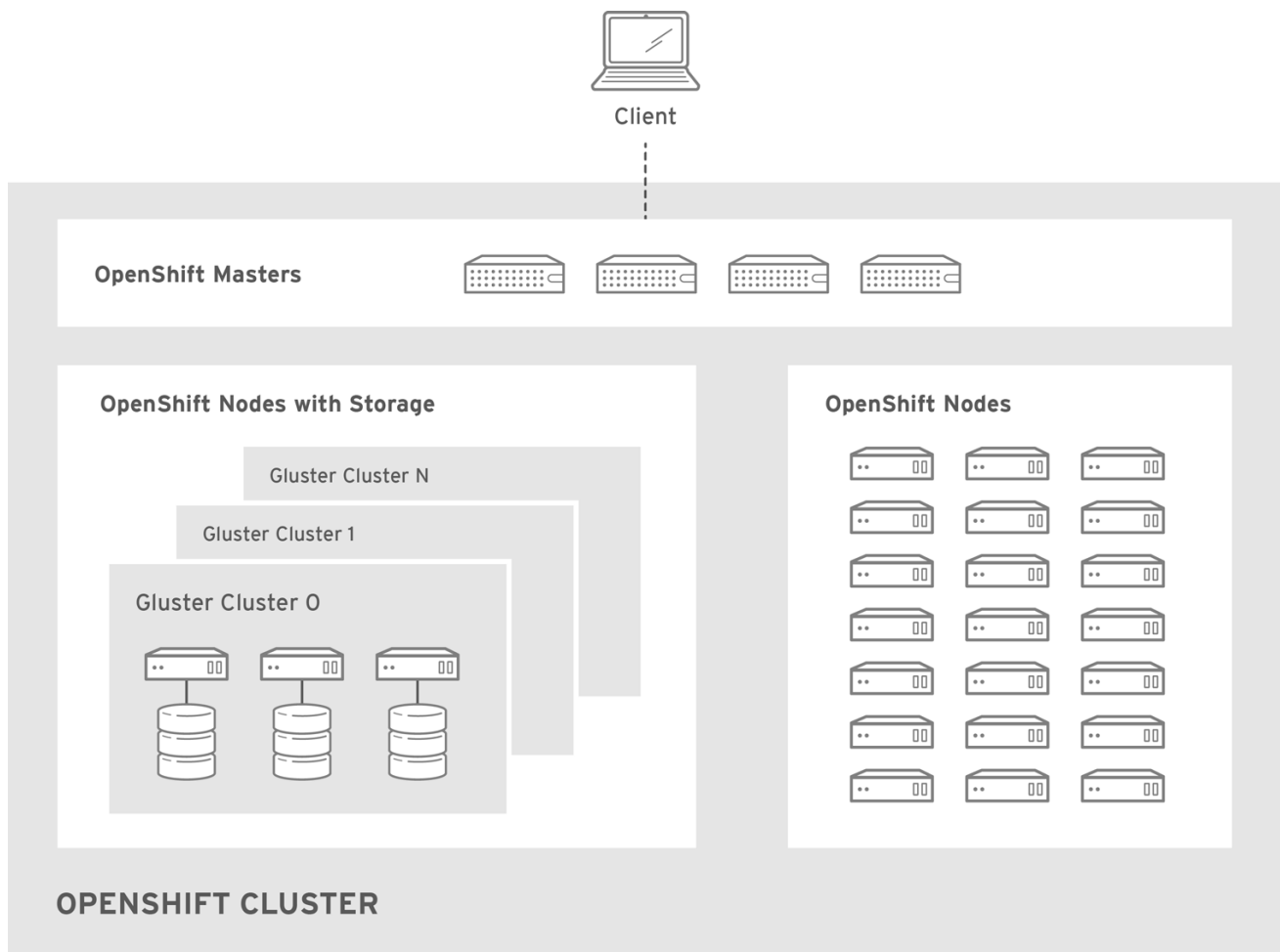
24.3.1. Overview

Red Hat Gluster Storage can be configured to provide persistent storage and dynamic provisioning for OpenShift Container Platform. It can be used both containerized within OpenShift Container Platform (**converged mode**) and non-containerized on its own nodes (**independent mode**).

24.3.1.1. converged mode

With converged mode, Red Hat Gluster Storage runs containerized directly on OpenShift Container Platform nodes. This allows for compute and storage instances to be scheduled and run from the same set of hardware.

Figure 24.1. Architecture - converged mode



OPENSIFT_412816_0716

converged mode is available starting with Red Hat Gluster Storage 3.1 update 3. See [converged mode for OpenShift Container Platform](#) for additional documentation.

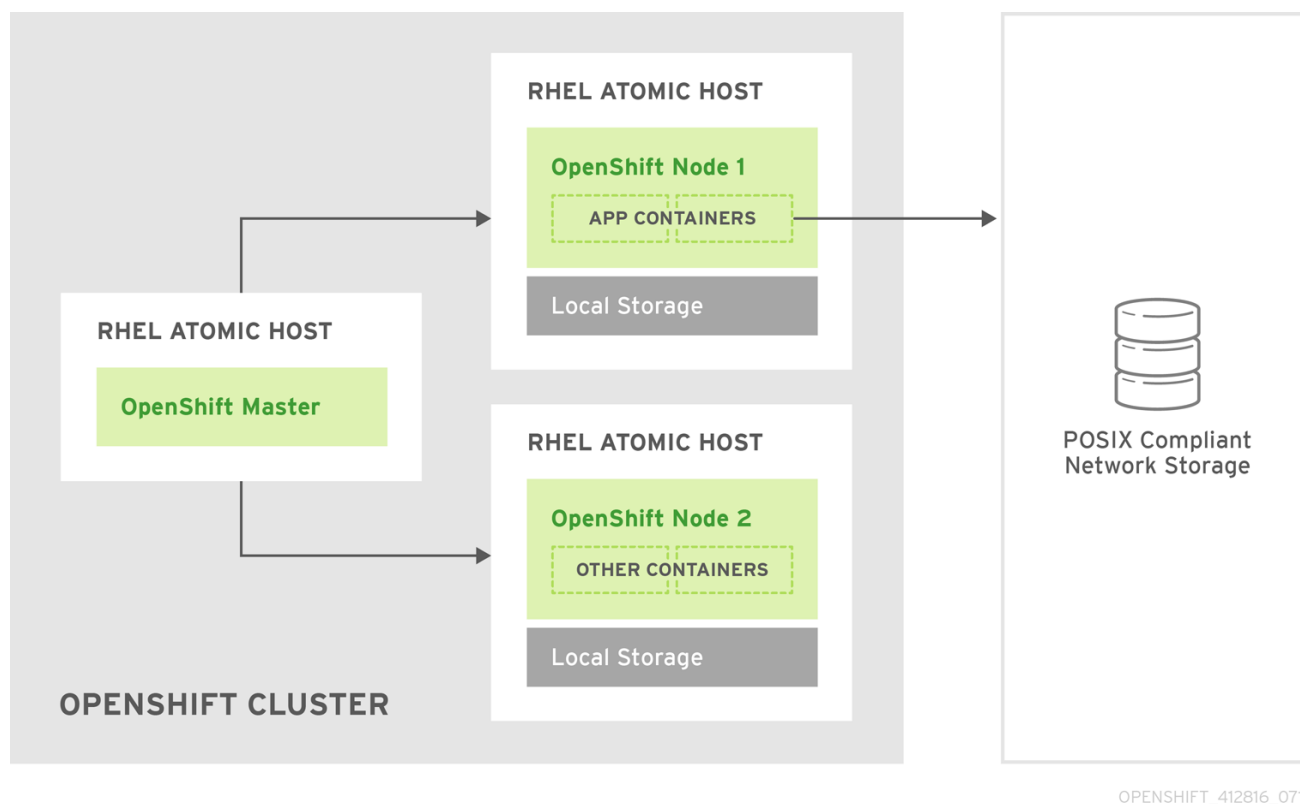
24.3.1.2. independent mode

With independent mode, Red Hat Gluster Storage runs on its own dedicated nodes and is managed by an instance of [heketi](#), the GlusterFS volume management REST service. This heketi service can run either standalone or containerized. Containerization allows for an easy mechanism to provide high-availability to the service. This documentation will focus on the configuration where heketi is containerized.

24.3.1.3. Standalone Red Hat Gluster Storage

If you have a standalone Red Hat Gluster Storage cluster available in your environment, you can make use of volumes on that cluster using OpenShift Container Platform's GlusterFS volume plug-in. This solution is a conventional deployment where applications run on dedicated compute nodes, an OpenShift Container Platform cluster, and storage is provided from its own dedicated nodes.

Figure 24.2. Architecture - Standalone Red Hat Gluster Storage Cluster Using OpenShift Container Platform's GlusterFS Volume Plug-in



See the [Red Hat Gluster Storage Installation Guide](#) and the [Red Hat Gluster Storage Administration Guide](#) for more on Red Hat Gluster Storage.



IMPORTANT

High availability of storage in the infrastructure is left to the underlying storage provider.

24.3.1.4. GlusterFS Volumes

GlusterFS volumes present a POSIX-compliant filesystem and are comprised of one or more "bricks" across one or more nodes in their cluster. A brick is just a directory on a given storage node and is typically the mount point for a block storage device. GlusterFS handles distribution and replication of files across a given volume's bricks per that volume's configuration.

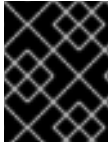
It is recommended to use heketi for most common volume management operations such as create, delete, and resize. OpenShift Container Platform expects heketi to be present when using the GlusterFS provisioner. heketi by default will create volumes that are three-ray replica, that is volumes where each file has three copies across three different nodes. As such it is recommended that any Red Hat Gluster Storage clusters which will be used by heketi have at least three nodes available.

There are many features available for GlusterFS volumes, but they are beyond the scope of this documentation.

24.3.1.5. gluster-block Volumes

gluster-block volumes are volumes that can be mounted over iSCSI. This is done by creating a file on an existing GlusterFS volume and then presenting that file as a block device via an iSCSI target. Such GlusterFS volumes are called block-hosting volumes.

gluster-block volumes present a sort of trade-off. Being consumed as iSCSI targets, gluster-block volumes can only be mounted by one node/client at a time which is in contrast to GlusterFS volumes which can be mounted by multiple nodes/clients. Being files on the backend, however, allows for operations which are typically costly on GlusterFS volumes (e.g. metadata lookups) to be converted to ones which are typically much faster on GlusterFS volumes (e.g. reads and writes). This leads to potentially substantial performance improvements for certain workloads.

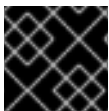


IMPORTANT

At this time, it is recommended to only use gluster-block volumes for OpenShift Logging and OpenShift Metrics storage.

24.3.1.6. Gluster S3 Storage

The Gluster S3 service allows user applications to access GlusterFS storage via an S3 interface. The service binds to two GlusterFS volumes, one for object data and one for object metadata, and translates incoming S3 REST requests into filesystem operations on the volumes. It is recommended to run the service as a pod inside OpenShift Container Platform.



IMPORTANT

At this time, use and installation of the Gluster S3 service is in tech preview.

24.3.2. Considerations

This section covers a few topics that should be taken into consideration when using Red Hat Gluster Storage with OpenShift Container Platform.

24.3.2.1. Software Prerequisites

To access GlusterFS volumes, the **mount.glusterfs** command must be available on all schedulable nodes. For RPM-based systems, the **glusterfs-fuse** package must be installed:

```
# yum install glusterfs-fuse
```

This package comes installed on every RHEL system. However, it is recommended to update to the latest available version from Red Hat Gluster Storage if your servers use x86_64 architecture. To do this, the following RPM repository must be enabled:

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

If **glusterfs-fuse** is already installed on the nodes, ensure that the latest version is installed:

```
# yum update glusterfs-fuse
```

24.3.2.2. Hardware Requirements

Any nodes used in a converged mode or independent mode cluster are considered storage nodes. Storage nodes can be grouped into distinct cluster groups, though a single node can not be in multiple groups. For each group of storage nodes:

- A minimum of three storage nodes per group is required.
- Each storage node must have a minimum of 8 GB of RAM. This is to allow running the Red Hat Gluster Storage pods, as well as other applications and the underlying operating system.
 - Each GlusterFS volume also consumes memory on every storage node in its storage cluster, which is about 30 MB. The total amount of RAM should be determined based on how many concurrent volumes are desired or anticipated.
- Each storage node must have at least one raw block device with no present data or metadata. These block devices will be used in their entirety for GlusterFS storage. Make sure the following are not present:
 - Partition tables (GPT or MSDOS)
 - Filesystems or residual filesystem signatures
 - LVM2 signatures of former Volume Groups and Logical Volumes
 - LVM2 metadata of LVM2 physical volumes

If in doubt, `wipefs -a <device>` should clear any of the above.



IMPORTANT

It is recommended to plan for two clusters: one dedicated to storage for infrastructure applications (such as an OpenShift Container Registry) and one dedicated to storage for general applications. This would require a total of six storage nodes. This recommendation is made to avoid potential impacts on performance in I/O and volume creation.

24.3.2.3. Storage Sizing

Every GlusterFS cluster must be sized based on the needs of the anticipated applications that will use its storage. For example, there are sizing guides available for both [OpenShift Logging](#) and [OpenShift Metrics](#).

Some additional things to consider are:

- For each converged mode or independent mode cluster, the default behavior is to create GlusterFS volumes with three-way replication. As such, the total storage to plan for should be the desired capacity times three.
 - As an example, each `heketi` instance creates a `heketi.dbstorage` volume that is 2 GB in size, requiring a total of 6 GB of raw storage across three nodes in the storage cluster. This capacity is always required and should be taken into consideration for sizing calculations.
 - Applications like an integrated OpenShift Container Registry share a single GlusterFS volume across multiple instances of the application.
- gluster-block volumes require the presence of a GlusterFS block-hosting volume with enough capacity to hold the full size of any given block volume's capacity.

- By default, if no such block-hosting volume exists, one will be automatically created at a set size. The default for this size is 100 GB. If there is not enough space in the cluster to create the new block-hosting volume, the creation of the block volume will fail. Both the auto-create behavior and the auto-created volume size are configurable.
- Applications with multiple instances that use gluster-block volumes, such as OpenShift Logging and OpenShift Metrics, will use one volume per instance.
- The Gluster S3 service binds to two GlusterFS volumes. In a [default cluster installation](#), these volumes are 1 GB each, consuming a total of 6 GB of raw storage.

24.3.2.4. Volume Operation Behaviors

Volume operations, such as create and delete, can be impacted by a variety of environmental circumstances and can in turn affect applications as well.

- If the application pod requests a dynamically provisioned GlusterFS persistent volume claim (PVC), then extra time might have to be considered for the volume to be created and bound to the corresponding PVC. This effects the startup time for an application pod.



NOTE

Creation time of GlusterFS volumes scales linearly depending on the number of volumes. As an example, given 100 volumes in a cluster using recommended hardware specifications, each volume took approximately 6 seconds to be created, allocated, and bound to a pod.

- When a PVC is deleted, that action will trigger the deletion of the underlying GlusterFS volume. While PVCs will disappear immediately from the **oc get pvc** output, this does not mean the volume has been fully deleted. A GlusterFS volume can only be considered deleted when it does not show up in the command-line outputs for **heketi-cli volume list** and **gluster volume list**.



NOTE

The time to delete the GlusterFS volume and recycle its storage depends on and scales linearly with the number of active GlusterFS volumes. While pending volume deletes do not affect running applications, storage administrators should be aware of and be able to estimate how long they will take, especially when tuning resource consumption at scale.

24.3.2.5. Volume Security

This section covers Red Hat Gluster Storage volume security, including Portable Operating System Interface [for Unix] (POSIX) permissions and SELinux considerations. Understanding the basics of [Volume Security](#), POSIX permissions, and SELinux is presumed.

24.3.2.5.1. POSIX Permissions

Red Hat Gluster Storage volumes present POSIX-compliant file systems. As such, access permissions can be managed using standard command-line tools such as **chmod** and **chown**.

For converged mode and independent mode, it is also possible to specify a group ID that will own the root of the volume at volume creation time. For static provisioning, this is specified as part of the **heketi-cli** volume creation command:

```
$ heketi-cli volume create --size=100 --gid=10001000
```



WARNING

The PersistentVolume that will be associated with this volume must be annotated with the group ID so that pods consuming the PersistentVolume can have access to the file system. This annotation takes the form of:

```
pv.beta.kubernetes.io/gid: "<GID>" ---
```

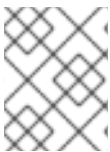
For dynamic provisioning, the provisioner automatically generates and applies a group ID. It is possible to control the range from which this group ID is selected using the **gidMin** and **gidMax** StorageClass parameters (see [Dynamic Provisioning](#)). The provisioner also takes care of annotating the generated PersistentVolume with the group ID.

24.3.2.5.2. SELinux

By default, SELinux does not allow writing from a pod to a remote Red Hat Gluster Storage server. To enable writing to Red Hat Gluster Storage volumes with SELinux on, run the following on each node running GlusterFS:

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

1 The **-P** option makes the boolean persistent between reboots.



NOTE

The **virt_sandbox_use_fusefs** boolean is defined by the **docker-selinux** package. If you get an error saying it is not defined, ensure that this package is installed.



NOTE

If you use Atomic Host, the SELinux booleans are cleared when you upgrade Atomic Host. When you upgrade Atomic Host, you must set these boolean values again.

24.3.3. Support Requirements

The following requirements must be met to create a supported integration of Red Hat Gluster Storage and OpenShift Container Platform.

For independent mode or standalone Red Hat Gluster Storage:

- Minimum version: Red Hat Gluster Storage 3.1.3

- All Red Hat Gluster Storage nodes must have valid subscriptions to Red Hat Network channels and Subscription Manager repositories.
- Red Hat Gluster Storage nodes must adhere to the requirements specified in the [Planning Red Hat Gluster Storage Installation](#).
- Red Hat Gluster Storage nodes must be completely up to date with the latest patches and upgrades. Refer to the [Red Hat Gluster Storage Installation Guide](#) to upgrade to the latest version.
- A fully-qualified domain name (FQDN) must be set for each Red Hat Gluster Storage node. Ensure that correct DNS records exist, and that the FQDN is resolvable via both forward and reverse DNS lookup.

24.3.4. Installation

For standalone Red Hat Gluster Storage, there is no component installation required to use it with OpenShift Container Platform. OpenShift Container Platform comes with a built-in GlusterFS volume driver, allowing it to make use of existing volumes on existing clusters. See [provisioning](#) for more on how to make use of existing volumes.

For converged mode and independent mode, it is recommended to use the [cluster installation](#) process to install the required components.

24.3.4.1. independent mode: Installing Red Hat Gluster Storage Nodes

For independent mode, each Red Hat Gluster Storage node must have the appropriate system configurations (e.g. firewall ports, kernel modules) and the Red Hat Gluster Storage services must be running. The services should not be further configured, and should not have formed a Trusted Storage Pool.

The installation of Red Hat Gluster Storage nodes is beyond the scope of this documentation. For more information, see [Setting Up independent mode](#).

24.3.4.2. Using the Installer

The [cluster installation](#) process can be used to install one or both of two GlusterFS node groups:

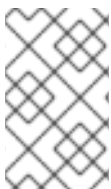
- **glusterfs**: A general storage cluster for use by user applications.
- **glusterfs-registry**: A dedicated storage cluster for use by infrastructure applications such as an integrated OpenShift Container Registry.

It is recommended to deploy both groups to avoid potential impacts on performance in I/O and volume creation. Both of these are defined in the inventory hosts file.

The definition of the clusters is done by including the relevant names in the **[OSEv3:children]** group, creating similarly named groups, and then populating the groups with the node information. The clusters can then be configured through a variety of variables in the **[OSEv3:vars]** group. **glusterfs** variables begin with **openshift_storage_glusterfs_** and **glusterfs-registry** variables begin with **openshift_storage_glusterfs_registry_**. A few other variables, such as **openshift_hosted_registry_storage_kind**, interact with the GlusterFS clusters.

To prevent Red Hat Gluster Storage pods from upgrading after an outage leading to a cluster with different Red Hat Gluster Storage versions, it is recommended to specify the image name and version tags for all containerized components. The relevant variables are:

- `openshift_storage_glusterfs_image`
- `openshift_storage_glusterfs_block_image`
- `openshift_storage_glusterfs_s3_image`
- `openshift_storage_glusterfs_heketi_image`
- `openshift_storage_glusterfs_registry_image`
- `openshift_storage_glusterfs_registry_block_image`
- `openshift_storage_glusterfs_registry_s3_image`
- `openshift_storage_glusterfs_registry_heketi_image`



NOTE

The image variables for ***gluster-block*** and ***gluster-s3*** are only necessary if the corresponding deployment variables (the variables ending in ***_block_deploy*** and ***_s3_deploy***) are true.

The following are the recommended values for this release of OpenShift Container Platform:

- `openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-server-rhel7:v3.10`
- `openshift_storage_glusterfs_block_image=registry.access.redhat.com/rhgs3/rhgs-gluster-block-prov-rhel7:v3.10`
- `openshift_storage_glusterfs_s3_image=registry.access.redhat.com/rhgs3/rhgs-s3-server-rhel7:v3.10`
- `openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/rhgs-volmanager-rhel7:v3.10`



NOTE

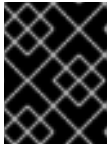
The values are same for the corresponding ***registry*** variables.

For a complete list of variables, see the [GlusterFS role README](#) on GitHub.

Once the variables are configured, there are several playbooks available depending on the circumstances of the installation:

- The main playbook for cluster installations can be used to deploy the GlusterFS clusters in tandem with an initial installation of OpenShift Container Platform.
 - This includes deploying an integrated OpenShift Container Registry that uses GlusterFS storage.

- This does not include OpenShift Logging or OpenShift Metrics, as that is currently still a separate step. See [converged mode for OpenShift Logging and Metrics](#) for more information.
- **playbooks/openshift-glusterfs/config.yml** can be used to deploy the clusters onto an existing OpenShift Container Platform installation.
- **playbooks/openshift-glusterfs/registry.yml** can be used to deploy the clusters onto an existing OpenShift Container Platform installation. In addition, this will deploy an integrated OpenShift Container Registry which uses GlusterFS storage.

**IMPORTANT**

There must not be a pre-existing registry in the OpenShift Container Platform cluster.

- **playbooks/openshift-glusterfs/uninstall.yml** can be used to remove existing clusters matching the configuration in the inventory hosts file. This is useful for cleaning up the OpenShift Container Platform environment in the case of a failed deployment due to configuration errors.

**NOTE**

The GlusterFS playbooks are not guaranteed to be idempotent.

**NOTE**

Running the playbooks more than once for a given installation is currently not supported without deleting the entire GlusterFS installation (including disk data) and starting over.

24.3.4.2.1. Example: Basic converged mode Installation

1. In your inventory file, include the following variables in the **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
```

2. Add **glusterfs** in the **[OSEv3:children]** section to enable the **[glusterfs]** group:

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. Add a **[glusterfs]** section with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Specifying the variable takes the form:

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "  
</path/to/device2/>", ... ]'
```

For example:

```
[glusterfs]  
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'  
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'  
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. Add the hosts listed under **[glusterfs]** to the **[nodes]** group:

```
[nodes]  
...  
node11.example.com openshift_node_group_name="node-config-compute"  
node12.example.com openshift_node_group_name="node-config-compute"  
node13.example.com openshift_node_group_name="node-config-compute"
```



NOTE

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

5. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```
ansible-playbook -i <path_to_inventory_file>  
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml  
  
ansible-playbook -i <path_to_inventory_file>  
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>  
/usr/share/ansible/openshift-ansible/playbooks/openshift-  
glusterfs/config.yml
```

24.3.4.2.2. Example: Basic independent mode Installation

1. In your inventory file, include the following variables in the **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]  
...  
openshift_storage_glusterfs_namespace=app-storage
```

```

openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"

```

2. Add **glusterfs** in the **[OSEv3:children]** section to enable the **[glusterfs]** group:

```

[OSEv3:children]
masters
nodes
glusterfs

```

3. Add a **[glusterfs]** section with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Also, set **glusterfs_ip** to the IP address of the node. Specifying the variable takes the form:

```

<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "
</path/to/device1/>', "</path/to/device2/>", ... ]'

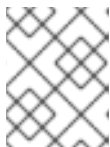
```

For example:

```

[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'

```



NOTE

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

4. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml

```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

24.3.4.2.3. Example: converged mode with an Integrated OpenShift Container Registry

1. In your inventory file, set the following variable under **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs 1
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'
```

- 1 Running the integrated OpenShift Container Registry, on infrastructure nodes is recommended. Infrastructure node are nodes dedicated to running applications deployed by administrators to provide services for the OpenShift Container Platform cluster.

2. Add **glusterfs_registry** in the **[OSEv3:children]** section to enable the **[glusterfs_registry]** group:

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

3. Add a **[glusterfs_registry]** section with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Specifying the variable takes the form:

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

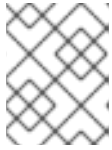
For example:

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. Add the hosts listed under **[glusterfs_registry]** to the **[nodes]** group:

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
```

```
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

**NOTE**

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

5. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

24.3.4.2.4. Example: converged mode for OpenShift Logging and Metrics

1. In your inventory file, set the following variables under **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]
...

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❶
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❷
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❸
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" ❹

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❺
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❻
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} ❼
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi ❽
```

```
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block" 9
```

```
openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
```

- 1 2 3 5 6 7 It is recommended to run the integrated OpenShift Container Registry, Logging, and Metrics on nodes dedicated to "infrastructure" applications, that is applications deployed by administrators to provide services for the OpenShift Container Platform cluster.

- 4 9 Specify the StorageClass to be used for Logging and Metrics. This name is generated from the name of the target GlusterFS cluster (e.g., **glusterfs-`<name>-block`**). In this example, this defaults to **registry**.

- 8 OpenShift Logging requires that a PVC size be specified. The supplied value is only an example, not a recommendation.



NOTE

See the [GlusterFS role README](#) for details on these and other variables.

2. Add **glusterfs_registry** in the **[OSEv3:children]** section to enable the **[glusterfs_registry]** group:

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

3. Add a **[glusterfs_registry]** section with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Specifying the variable takes the form:

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2/>", ... ]'
```

For example:

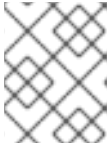
```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. Add the hosts listed under **[glusterfs_registry]** to the **[nodes]** group:

```
[nodes]
```



```
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

**NOTE**

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

5. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

24.3.4.2.5. Example: converged mode for Applications, Registry, Logging, and Metrics

1. In your inventory file, set the following variables under **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs ❶
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❷
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❸
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❹
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" ❺

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❻
```

```

openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 9
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block" 10

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=fals
e

```

- 1 2 3 4 6 7 8** Running the integrated OpenShift Container Registry, Logging, and Metrics on infrastructure nodes is recommended. Infrastructure nodes are nodes dedicated to running applications deployed by administrators to provide services for the OpenShift Container Platform cluster.
- 5 10** Specify the StorageClass to be used for Logging and Metrics. This name is generated from the name of the target GlusterFS cluster, for example **glusterfs-`<name>`-block**. In this example, `<name>` defaults to **registry**.
- 9** Specifying a PVC size is required for OpenShift Logging. The supplied value is only an example, not a recommendation.

2. Add **glusterfs** and **glusterfs_registry** in the **[OSEv3:children]** section to enable the **[glusterfs]** and **[glusterfs_registry]** groups:

```

[OSEv3:children]
...
glusterfs
glusterfs_registry

```

3. Add **[glusterfs]** and **[glusterfs_registry]** sections with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Specifying the variable takes the form:

```

<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'

```

For example:

```
[glusterfs]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
node14.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node15.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node16.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. Add the hosts listed under **[glusterfs]** and **[glusterfs_registry]** to the **[nodes]** group:

```
[nodes]
...
node11.example.com openshift_node_group_name='node-config-compute'
1 node12.example.com openshift_node_group_name='node-config-compute'
2 node13.example.com openshift_node_group_name='node-config-compute'
3 node14.example.com openshift_node_group_name='node-config-infra' 4
node15.example.com openshift_node_group_name='node-config-infra' 5
node16.example.com openshift_node_group_name='node-config-infra' 6
```

- 1 2 3 4 5 6 The nodes are marked to denote whether they will allow general applications or infrastructure applications to be scheduled on them. It is up to the administrator to configure how applications will be constrained.



NOTE

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

5. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

24.3.4.2.6. Example: independent mode for Applications, Registry, Logging, and Metrics

1. In your inventory file, set the following variables under **[OSEv3:vars]** section, and adjust them as required for your configuration:

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs ❶
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❷
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❸
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❹
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" ❺

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❻
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} ❼
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} ❽
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi ❾
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block" ❿

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=fals
e
```

```

openshift_storage_glusterfs_registry_is_native=false
openshift_storage_glusterfs_registry_heketi_is_native=true
openshift_storage_glusterfs_registry_heketi_executor=ssh
openshift_storage_glusterfs_registry_heketi_ssh_port=22
openshift_storage_glusterfs_registry_heketi_ssh_user=root
openshift_storage_glusterfs_registry_heketi_ssh_sudo=false
openshift_storage_glusterfs_registry_heketi_ssh_keyfile="/root/.ssh/
id_rsa"

```

- 1 2 3 4 6 7 8** It is recommended to run the integrated OpenShift Container Registry on nodes dedicated to "infrastructure" applications, that is applications deployed by administrators to provide services for the OpenShift Container Platform cluster. It is up to the administrator to select and label nodes for infrastructure applications.
- 5 10** Specify the StorageClass to be used for Logging and Metrics. This name is generated from the name of the target GlusterFS cluster (e.g., **glusterfs-`<name>-block`**). In this example, this defaults to **registry**.
- 9** OpenShift Logging requires that a PVC size be specified. The supplied value is only an example, not a recommendation.

2. Add **glusterfs** and **glusterfs_registry** in the **[OSEv3:children]** section to enable the **[glusterfs]** and **[glusterfs_registry]** groups:

```

[OSEv3:children]
...
glusterfs
glusterfs_registry

```

3. Add **[glusterfs]** and **[glusterfs_registry]** sections with entries for each storage node that will host the GlusterFS storage. For each node, set **glusterfs_devices** to a list of raw block devices that will be completely managed as part of a GlusterFS cluster. There must be at least one device listed. Each device must be bare, with no partitions or LVM PVs. Also, set **glusterfs_ip** to the IP address of the node. Specifying the variable takes the form:

```

<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "
</path/to/device1/>", "</path/to/device2>", ... ]'

```

For example:

```

[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
gluster4.example.com glusterfs_ip=192.168.10.14 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster5.example.com glusterfs_ip=192.168.10.15 glusterfs_devices='[

```

```
"/dev/xvdc", "/dev/xvdd" ]'
gluster6.example.com glusterfs_ip=192.168.10.16 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
```

**NOTE**

The preceding steps only provide some of the options that must be added to the inventory file. Use the complete inventory file to deploy Red Hat Gluster Storage.

4. Run the installation playbook and provide the relative path for the inventory file as an option.

- For a new OpenShift Container Platform installation:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- For an installation onto an existing OpenShift Container Platform cluster:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

24.3.5. Uninstall converged mode

For converged mode, an OpenShift Container Platform install comes with a playbook to uninstall all resources and artifacts from the cluster. To use the playbook, provide the original inventory file that was used to install the target instance of converged mode and run the following playbook:

```
# ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/uninstall.yml
```

In addition, the playbook supports the use of a variable called **openshift_storage_glusterfs_wipe** which, when enabled, destroys any data on the block devices that were used for Red Hat Gluster Storage backend storage. To use the **openshift_storage_glusterfs_wipe** variable:

```
# ansible-playbook -i <path_to_inventory_file> -e
"openshift_storage_glusterfs_wipe=true"
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/uninstall.yml
```

**WARNING**

This procedure destroys data. Proceed with caution.

24.3.6. Provisioning

GlusterFS volumes can be provisioned either statically or dynamically. Static provisioning is available with all configurations. Only converged mode and independent mode support dynamic provisioning.

24.3.6.1. Static Provisioning

1. To enable static provisioning, first create a GlusterFS volume. See the [Red Hat Gluster Storage Administration Guide](#) for information on how to do this using the **gluster** command-line interface or the [heketi project site](#) for information on how to do this using **heketi-cli**. For this example, the volume will be named **myVol1**.
2. Define the following Service and Endpoints in **gluster-endpoints.yaml**:

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster 1
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster 2
subsets:
- addresses:
  - ip: 192.168.122.221 3
  ports:
  - port: 1 4
- addresses:
  - ip: 192.168.122.222 5
  ports:
  - port: 1 6
- addresses:
  - ip: 192.168.122.223 7
  ports:
  - port: 1 8
```

1 2 These names must match.

3 5 7 The **ip** values must be the actual IP addresses of a Red Hat Gluster Storage server, not hostnames.

4 6 8 The port number is ignored.

3. From the OpenShift Container Platform master host, create the Service and Endpoints:

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. Verify that the Service and Endpoints were created:

```
$ oc get services
NAME                                CLUSTER_IP          EXTERNAL_IP  PORT(S)
SELECTOR      AGE
glusterfs-cluster  172.30.205.34      <none>      1/TCP
<none>         44s

$ oc get endpoints
NAME                                ENDPOINTS
AGE
docker-registry  10.1.0.3:5000
4h
glusterfs-cluster
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1  11s
kubernetes       172.16.35.3:8443
4d
```



NOTE

Endpoints are unique per project. Each project accessing the GlusterFS volume needs its own Endpoints.

5. In order to access the volume, the container must run with either a user ID (UID) or group ID (GID) that has access to the file system on the volume. This information can be discovered in the following manner:

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0  myVol1 1 2
```

1 The UID is 592.

2 The GID is 590.

6. Define the following PersistentVolume (PV) in **gluster-pv.yaml**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
  annotations:
    pv.beta.kubernetes.io/gid: "590" 2
spec:
  capacity:
    storage: 2Gi 3
  accessModes: 4
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster 5
```



```

    path: myVol1 ❹
    readOnly: false
    persistentVolumeReclaimPolicy: Retain

```

- ❶ The name of the volume.
- ❷ The GID on the root of the GlusterFS volume.
- ❸ The amount of storage allocated to this volume.
- ❹ **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control.
- ❺ The Endpoints resource previously created.
- ❻ The GlusterFS volume that will be accessed.

7. From the OpenShift Container Platform master host, create the PV:

```
$ oc create -f gluster-pv.yaml
```

8. Verify that the PV was created:

```

$ oc get pv
NAME                                LABELS            CAPACITY          ACCESSMODES
STATUS      CLAIM          REASON    AGE
gluster-default-volume  <none>         2147483648        RWX
Available                                     2s

```

9. Create a PersistentVolumeClaim (PVC) that will bind to the new PV in **gluster-claim.yaml**:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim ❶
spec:
  accessModes:
    - ReadWriteMany ❷
  resources:
    requests:
      storage: 1Gi ❸

```

- ❶ The claim name is referenced by the pod under its **volumes** section.
- ❷ Must match the **accessModes** of the PV.
- ❸ This claim will look for PVs offering **1Gi** or greater capacity.

10. From the OpenShift Container Platform master host, create the PVC:

```
$ oc create -f gluster-claim.yaml
```

11. Verify that the PV and PVC are bound:

```
$ oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS    CLAIM
REASON    AGE
gluster-pv    <none>         1Gi       RWX           Available
gluster-claim          37s

$ oc get pvc
NAME          LABELS          STATUS    VOLUME          CAPACITY
ACCESSMODES  AGE
gluster-claim <none>         Bound     gluster-pv       1Gi
24s          RWX
```



NOTE

PVCs are unique per project. Each project accessing the GlusterFS volume needs its own PVC. PVs are not bound to a single project, so PVCs across multiple projects may refer to the same PV.

24.3.6.2. Dynamic Provisioning

1. To enable dynamic provisioning, first create a **StorageClass** object definition. The definition below is based on the minimum requirements needed for this example to work with OpenShift Container Platform. See [Dynamic Provisioning and Creating Storage Classes](#) for additional parameters and specification definitions.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" ❶
  restauthenabled: "false" ❷
```

- ❶ The heketi server URL.
- ❷ Since authentication is not turned on in this example, set to **false**.

2. From the OpenShift Container Platform master host, create the StorageClass:

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. Create a PVC using the newly-created StorageClass. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster1
spec:
  accessModes:
```

```
- ReadWriteMany
resources:
  requests:
    storage: 30Gi
  storageClassName: glusterfs
```

4. From the OpenShift Container Platform master host, create the PVC:

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

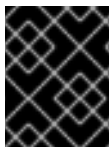
5. View the PVC to see that the volume was dynamically created and bound to the PVC:

```
# oc get pvc
NAME                STATUS VOLUME
CAPACITY            ACCESSMODES  STORAGECLASS  AGE
gluster1            Bound pvc-78852230-d8e2-11e6-a3fa-0800279cf26f
30Gi               RWX          gluster-dyn  42s
```

24.4. PERSISTENT STORAGE USING OPENSTACK CINDER

24.4.1. Overview

You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [OpenStack Cinder](#). Some familiarity with Kubernetes and OpenStack is assumed.



IMPORTANT

Before you create persistent volumes (PVs) using Cinder, [configured OpenShift Container Platform for OpenStack](#).

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. You can [provision](#) OpenStack Cinder volumes dynamically.

Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [Persistent volume claims](#), however, are specific to a project or namespace and can be requested by users.



IMPORTANT

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.4.2. Provisioning Cinder PVs

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring that OpenShift Container Platform is [configured for OpenStack](#), all that is required for Cinder is a Cinder volume ID and the **PersistentVolume** API.

24.4.2.1. Creating the Persistent Volume

**NOTE**

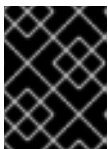
Cinder does not support the 'Recycle' reclaim policy.

You must define your PV in an object definition before creating it in OpenShift Container Platform:

1. Save your object definition to a file, for example ***cinder-pv.yaml***:

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  cinder: ❸
    fsType: "ext3" ❹
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" ❺
```

- ❶ The name of the volume that is used by [persistent volume claims](#) or pods.
- ❷ The amount of storage allocated to this volume.
- ❸ The volume type, in this case **cinder**.
- ❹ File system type to mount.
- ❺ The Cinder volume to use.

**IMPORTANT**

Do not change the **fstype** parameter value after the volume is formatted and provisioned. Changing this value can result in data loss and pod failure.

2. Create the persistent volume:

```
# oc create -f cinder-pv.yaml

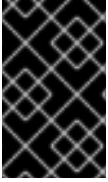
persistentvolume "pv0001" created
```

3. Verify that the persistent volume exists:

```
# oc get pv
```

NAME	LABELS	CAPACITY	ACCESSMODES	STATUS	CLAIM
REASON	AGE				
pv0001	<none>	5Gi	RW0	Available	
2s					

Users can then [request storage using persistent volume claims](#), which can now utilize your new persistent volume.

**IMPORTANT**

Persistent volume claims exist only in the user's namespace and can be referenced by a pod within that same namespace. Any attempt to access a persistent volume claim from a different namespace causes the pod to fail.

24.4.2.2. Cinder PV format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted Cinder volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

24.4.2.3. Cinder volume security

If you use Cinder PVs in your application, configure security for their deployment configurations.

**NOTE**

Review the [Volume Security](#) information before implementing Cinder volumes.

1. Create an [SCC](#) that uses the appropriate **fsGroup** strategy.
2. Create a service account and add it to the SCC:

```
[source,bash]
$ oc create serviceaccount <service_account>
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n
<project>
```

3. In your application's deployment configuration, provide the service account name and **securityContext**:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1
  selector:
    name: frontend
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
        - image: openshift/hello-openshift
          name: helloworld
          ports:
            - containerPort: 8080
```

```

    protocol: TCP
    restartPolicy: Always
    serviceAccountName: <service_account> 6
    securityContext:
      fsGroup: 7777 7

```

- 1 The number of copies of the pod to run.
- 2 The label selector of the pod to run.
- 3 A template for the pod the controller creates.
- 4 The labels on the pod must include labels from the label selector.
- 5 The maximum name length after expanding any parameters is 63 characters.
- 6 Specify the service account you created.
- 7 Specify an **fsGroup** for the pods.

24.5. PERSISTENT STORAGE USING CEPH RADOS BLOCK DEVICE (RBD)

24.5.1. Overview

OpenShift Container Platform clusters can be provisioned with [persistent storage](#) using Ceph RBD.

Persistent volumes (PVs) and [persistent volume claims \(PVCs\)](#) can share volumes across a single project. While the Ceph RBD-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

This topic presumes some familiarity with OpenShift Container Platform and [Ceph RBD](#). See the [Persistent Storage](#) concept topic for details on the OpenShift Container Platform persistent volume (PV) framework in general.



NOTE

Project and *namespace* are used interchangeably throughout this document. See [Projects and Users](#) for details on the relationship.



IMPORTANT

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.5.2. Provisioning

To provision Ceph volumes, the following are required:

- An existing storage device in your underlying infrastructure.
- The Ceph key to be used in an OpenShift Container Platform secret object.

- The Ceph image name.
- The file system type on top of the block storage (e.g., ext4).
- **ceph-common** installed on each schedulable OpenShift Container Platform node in your cluster:

```
# yum install ceph-common
```

24.5.2.1. Creating the Ceph Secret

Define the authorization key in a secret configuration, which is then converted to base64 for use by OpenShift Container Platform.



NOTE

In order to use Ceph storage to back a persistent volume, the secret must be created in the same project as the PVC and pod. The secret cannot simply be in the default project.

1. Run **ceph auth get-key** on a Ceph MON node to display the key value for the **client.admin** user:

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFB0FF2S1ZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ==
```

2. Save the secret definition to a file, for example **ceph-secret.yaml**, then create the secret:

```
$ oc create -f ceph-secret.yaml
```

3. Verify that the secret was created:

```
# oc get secret ceph-secret
NAME          TYPE          DATA      AGE
ceph-secret   Opaque        1          23d
```

24.5.2.2. Creating the Persistent Volume



NOTE

Ceph RBD does not support the 'Recycle' reclaim policy.

Developers request Ceph RBD storage by referencing either a PVC, or the Gluster volume plug-in directly in the **volumes** section of a pod specification. A PVC exists only in the user's namespace and can be referenced only by pods within that same namespace. Any attempt to access a PV from a different namespace causes the pod to fail.

1. Define the PV in an object definition before creating it in OpenShift Container Platform:

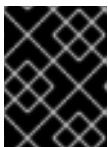
Example 24.3. Persistent Volume Object Definition Using Ceph RBD

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv ❶
spec:
  capacity:
    storage: 2Gi ❷
  accessModes:
    - ReadWriteOnce ❸
  rbd: ❹
    monitors: ❺
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret ❻
    fsType: ext4 ❼
    readOnly: false
  persistentVolumeReclaimPolicy: Retain

```

- ❶ The name of the PV that is referenced in pod definitions or displayed in various **oc** volume commands.
- ❷ The amount of storage allocated to this volume.
- ❸ **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control. All block storage is defined to be single user (non-shared storage).
- ❹ The volume type being used, in this case the **rbd** plug-in.
- ❺ An array of Ceph monitor IP addresses and ports.
- ❻ The Ceph secret used to create a secure connection from OpenShift Container Platform to the Ceph server.
- ❼ The file system type mounted on the Ceph RBD block device.



IMPORTANT

Changing the value of the **fstype** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

2. Save your definition to a file, for example **ceph-pv.yaml**, and create the PV:

```
# oc create -f ceph-pv.yaml
```

3. Verify that the persistent volume was created:

```
# oc get pv
```


NAME		LABELS	CAPACITY	ACCESSMODES
STATUS	CLAIM	REASON	AGE	
ceph-pv		<none>	2147483648	RWO
Available			2s	

4. Create a PVC that will bind to the new PV:

Example 24.4. PVC Object Definition

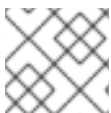
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

- 1** The **accessModes** do not enforce access right, but instead act as labels to match a PV to a PVC.
- 2** This claim looks for PVs offering **2Gi** or greater capacity.

5. Save the definition to a file, for example ***ceph-claim.yaml***, and create the PVC:

```
# oc create -f ceph-claim.yaml
```

24.5.3. Ceph Volume Security



NOTE

See the full [Volume Security](#) topic before implementing Ceph RBD volumes.

A significant difference between shared volumes (NFS and GlusterFS) and block volumes (Ceph RBD, iSCSI, and most cloud storage), is that the user and group IDs defined in the pod definition or container image are applied to the target physical storage. This is referred to as managing ownership of the block device. For example, if the Ceph RBD mount has its owner set to **123** and its group ID set to **567**, and if the pod defines its **runAsUser** set to **222** and its **fsGroup** to be **7777**, then the Ceph RBD physical mount's ownership will be changed to **222:7777**.



NOTE

Even if the user and group IDs are not defined in the pod specification, the resulting pod may have defaults defined for these IDs based on its matching SCC, or its project. See the full [Volume Security](#) topic which covers storage aspects of SCCs and defaults in greater detail.

A pod defines the group ownership of a Ceph RBD volume using the **fsGroup** stanza under the pod's **securityContext** definition:

```
spec:
  containers:
    - name:
      ...
  securityContext: 1
    fsGroup: 7777 2
```

- 1 The **securityContext** must be defined at the pod level, not under a specific container.
- 2 All containers in the pod will have the same fsGroup ID.

24.6. PERSISTENT STORAGE USING AWS ELASTIC BLOCK STORE

24.6.1. Overview

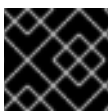
OpenShift Container Platform supports AWS Elastic Block Store volumes (EBS). You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [AWS EC2](#). Some familiarity with Kubernetes and AWS is assumed.



IMPORTANT

Before creating persistent volumes using AWS, OpenShift Container Platform must first be properly [configured for AWS ElasticBlockStore](#).

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. AWS Elastic Block Store volumes can be [provisioned dynamically](#). Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [Persistent volume claims](#), however, are specific to a project or namespace and can be requested by users.



IMPORTANT

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.6.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift is [configured for AWS Elastic Block Store](#), all that is required for OpenShift and AWS is an AWS EBS volume ID and the **PersistentVolume** API.

24.6.2.1. Creating the Persistent Volume



NOTE

AWS does not support the 'Recycle' reclaim policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

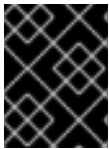
Example 24.5. Persistent Volume Object Definition Using AWS

```

apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  awsElasticBlockStore: ❸
    fsType: "ext4" ❹
    volumeID: "vol-f37a03aa" ❺

```

- ❶ The name of the volume. This will be how it is identified via [persistent volume claims](#) or from pods.
- ❷ The amount of storage allocated to this volume.
- ❸ This defines the volume type being used, in this case the **awsElasticBlockStore** plug-in.
- ❹ File system type to mount.
- ❺ This is the AWS volume that will be used.

**IMPORTANT**

Changing the value of the **fstype** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

Save your definition to a file, for example **aws-pv.yaml**, and create the persistent volume:

```

# oc create -f aws-pv.yaml
persistentvolume "pv0001" created

```

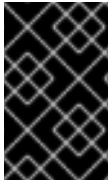
Verify that the persistent volume was created:

```

# oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS      CLAIM          REASON
AGE
pv0001        <none>          5Gi       RWO           Available
2s

```

Users can then [request storage using persistent volume claims](#), which can now utilize your new persistent volume.

**IMPORTANT**

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

24.6.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted AWS volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

24.6.2.3. Maximum Number of EBS Volumes on a Node

By default, OpenShift Container Platform supports a maximum of 39 EBS volumes attached to one node. This limit is consistent with the [AWS Volume Limits](#).

OpenShift Container Platform can be configured to have a higher limit by setting the environment variable **KUBE_MAX_PD_VOLS**. However, AWS requires a particular naming scheme ([AWS Device Naming](#)) for attached devices, which only supports a maximum of 52 volumes. This limits the number of volumes that can be attached to a node via OpenShift Container Platform to 52.

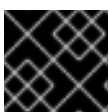
24.7. PERSISTENT STORAGE USING GCE PERSISTENT DISK**24.7.1. Overview**

OpenShift Container Platform supports GCE Persistent Disk volumes (gcePD). You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [GCE](#). Some familiarity with Kubernetes and GCE is assumed.

**IMPORTANT**

Before creating persistent volumes using GCE, OpenShift Container Platform must first be properly [configured for GCE Persistent Disk](#).

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. GCE Persistent Disk volumes can be [provisioned dynamically](#). Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [Persistent volume claims](#), however, are specific to a project or namespace and can be requested by users.

**IMPORTANT**

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.7.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift

Container Platform. After ensuring OpenShift Container Platform is [configured for GCE PersistentDisk](#), all that is required for OpenShift Container Platform and GCE is an GCE Persistent Disk volume ID and the **PersistentVolume** API.

24.7.2.1. Creating the Persistent Volume



NOTE

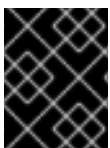
GCE does not support the 'Recycle' reclaim policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 24.6. Persistent Volume Object Definition Using GCE

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  gcePersistentDisk: 3
    fsType: "ext4" 4
    pdName: "pd-disk-1" 5
```

- 1 The name of the volume. This will be how it is identified via [persistent volume claims](#) or from pods.
- 2 The amount of storage allocated to this volume.
- 3 This defines the volume type being used, in this case the **gcePersistentDisk** plug-in.
- 4 File system type to mount.
- 5 This is the GCE Persistent Disk volume that will be used.



IMPORTANT

Changing the value of the **fsType** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

Save your definition to a file, for example **gce-pv.yaml**, and create the persistent volume:

```
# oc create -f gce-pv.yaml
persistentvolume "pv0001" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS   CLAIM          REASON
AGE
pv0001        <none>          5Gi       RWO           Available
2s
```

Users can then [request storage using persistent volume claims](#), which can now utilize your new persistent volume.



IMPORTANT

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

24.7.2.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows using unformatted GCE volumes as persistent volumes, because OpenShift Container Platform formats them before the first use.

24.8. PERSISTENT STORAGE USING ISCSI

24.8.1. Overview

You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [iSCSI](#). Some familiarity with Kubernetes and iSCSI is assumed.

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.



IMPORTANT

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.8.2. Provisioning

Verify that the storage exists in the underlying infrastructure before mounting it as a volume in OpenShift Container Platform. All that is required for the iSCSI is the iSCSI target portal, a valid iSCSI Qualified Name (IQN), a valid LUN number, the filesystem type, and the **PersistentVolume** API.

Optionally, multipath portals and Challenge Handshake Authentication Protocol (CHAP) configuration can be provided.



NOTE

iSCSI does not support the 'Recycle' reclaim policy.

Example 24.7. Persistent Volume Object Definition

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.16.154.81:3260
    portals: ['10.16.154.82:3260', '10.16.154.83:3260']
    iqn: iqn.2014-12.example.server:storage.target00
    lun: 0
    fsType: 'ext4'
    readOnly: false
    chapAuthDiscovery: true
    chapAuthSession: true
    secretRef:
      name: chap-secret

```

24.8.2.1. Enforcing Disk Quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is one persistent volume. Kubernetes enforces unique names for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (e.g, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

24.8.2.2. iSCSI Volume Security

Users request storage with a **PersistentVolumeClaim**. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each iSCSI LUN must be accessible by all nodes in the cluster.

24.8.2.3. iSCSI Multipathing

For iSCSI-based storage, you can configure multiple paths by using the same IQN for more than one target portal IP address. Multipathing ensures access to the persistent volume when one or more of the components in a path fail.

To specify multi-paths in pod specification use the **portals** field. For example:

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi_pv
spec:
  capacity:

```

```

    storage: 1Gi
    accessModes:
      - ReadWriteOnce
    iscsi:
      targetPortal: 10.0.0.1:3260
      portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] 1
      iqn: iqn.2016-04.test.com:storage.target00
      lun: 0
      fsType: ext4
      readOnly: false

```

- 1** Add additional target portals using the **portals** field.

24.8.2.4. iSCSI Custom Initiator IQN

Configure the custom initiator iSCSI Qualified Name (IQN) if the iSCSI targets are restricted to certain IQNs, but the nodes that the iSCSI PVs are attached to are not guaranteed to have these IQNs.

To specify custom initiator IQN, use **initiatorName** field.

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi_pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn 1
    fsType: ext4
    readOnly: false

```

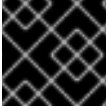
- 1** To add an additional custom initiator IQN, use **initiatorName** field.

24.9. PERSISTENT STORAGE USING FIBRE CHANNEL

24.9.1. Overview

You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [Fibre Channel](#). Some familiarity with Kubernetes and Fibre Channel is assumed.

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

**IMPORTANT**

High-availability of storage in the infrastructure is left to the underlying storage provider.

24.9.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. All that is required for Fibre Channel persistent storage is the **targetWWNs** (array of Fibre Channel target's World Wide Names), a valid **LUN** number, filesystem type, and the **PersistentVolume** API. Persistent volume and a LUN have one-to-one mapping between them.

**NOTE**

Fibre Channel does not support the 'Recycle' reclaim policy.

Persistent Volumes Object Definition

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] 1
    lun: 2
    fsType: ext4
```

- 1** Fibre Channel WWNs are identified as `/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>`, but you do not need to provide any part of the path leading up to the **WWN**, including the **0x**, and anything after, including the **-** (hyphen).

**IMPORTANT**

Changing the value of the **fstype** parameter after the volume has been formatted and provisioned can result in data loss and pod failure.

24.9.2.1. Enforcing Disk Quotas

Use LUN partitions to enforce disk quotas and size constraints. Each LUN is one persistent volume. Kubernetes enforces unique names for persistent volumes.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (e.g, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

24.9.2.2. Fibre Channel Volume Security

Users request storage with a **PersistentVolumeClaim**. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each Fibre Channel LUN must be accessible by all nodes in the cluster.

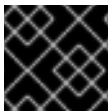
24.10. PERSISTENT STORAGE USING AZURE DISK

24.10.1. Overview

OpenShift Container Platform supports [Microsoft Azure Disk](#) volumes. You can provision your OpenShift Container Platform cluster with [persistent storage](#) using Azure. Some familiarity with Kubernetes and Azure is assumed.

The Kubernetes [persistent volume](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Azure Disk volumes can be [provisioned dynamically](#). Persistent volumes are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [Persistent volume claims](#), however, are specific to a project or namespace and can be requested by users.



IMPORTANT

High availability of storage in the infrastructure is left to the underlying storage provider.

24.10.2. Prerequisites

Before creating persistent volumes using Azure, ensure your OpenShift Container Platform cluster meets the following requirements:

- OpenShift Container Platform must first be [configured for Azure Disk](#).
- Each node host in the infrastructure must match the Azure virtual machine name.
- Each node host must be in the same resource group.

24.10.3. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is [configured for Azure Disk](#), all that is required for OpenShift Container Platform and Azure is an Azure Disk Name and Disk URI and the **PersistentVolume** API.

24.10.4. Configuring Azure Disk for regional cloud

Azure has multiple regions on which to deploy an instance. To specify a desired region, add the following to the **azure.conf** file:

```
cloud: <region>
```

The region can be any of the following:

- German cloud: **AZUREGERMANCLOUD**

- China cloud: **AZURECHINACLOUD**
- Public cloud: **AZUREPUBLICCLOUD**
- US cloud: **AZUREUSGOVERNMENTCLOUD**

24.10.4.1. Creating the Persistent Volume



NOTE

Azure does not support the **Recycle** reclaim policy.

You must define your persistent volume in an object definition before creating it in OpenShift Container Platform:

Example 24.8. Persistent Volume Object Definition Using Azure

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  azureDisk: ❸
    diskName: test2.vhd ❹
    diskURI: https://someaccount.blob.core.windows.net/vhds/test2.vhd ❺
    cachingMode: ReadWrite ❻
    fsType: ext4 ❼
    readOnly: false ❽
```

- ❶ The name of the volume. This will be how it is identified via [persistent volume claims](#) or from pods.
- ❷ The amount of storage allocated to this volume.
- ❸ This defines the volume type being used (**azureDisk** plug-in, in this example).
- ❹ The name of the data disk in the blob storage.
- ❺ The URI of the data disk in the blob storage.
- ❻ Host caching mode: **None**, **ReadOnly**, or **ReadWrite**.
- ❼ File system type to mount (for example, **ext4**, **xfs**, and so on).
- ❽ Defaults to **false** (read/write). **ReadOnly** here will force the **ReadOnly** setting in **VolumeMounts**.

**IMPORTANT**

Changing the value of the **fsType** parameter after the volume is formatted and provisioned can result in data loss and pod failure.

1. Save your definition to a file, for example **azure-pv.yaml**, and create the persistent volume:

```
# oc create -f azure-pv.yaml
persistentvolume "pv0001" created
```

2. Verify that the persistent volume was created:

```
# oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS      CLAIM
REASON        AGE
pv0001        <none>          5Gi       RW0           Available
2s
```

Now you can [request storage using persistent volume claims](#), which can now use your new persistent volume.

**IMPORTANT**

For a pod that has a mounted volume through an Azure disk PVC, scheduling the pod to a new node takes a few minutes. Wait for two to three minutes to complete the *Disk Detach* operation, and then start a new deployment. If a new pod creation request is started before completing the *Disk Detach* operation, the *Disk Attach* operation initiated by the pod creation fails, resulting in pod creation failure.

**IMPORTANT**

Persistent volume claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume from a different namespace causes the pod to fail.

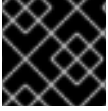
24.10.4.2. Volume Format

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that it contains a file system as specified by the **fsType** parameter in the persistent volume definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows unformatted Azure volumes to be used as persistent volumes because OpenShift Container Platform formats them before the first use.

24.11. PERSISTENT STORAGE USING AZURE FILE**24.11.1. Overview**

OpenShift Container Platform supports [Microsoft Azure File](#) volumes. You can provision your OpenShift Container Platform cluster with [persistent storage](#) using Azure. Some familiarity with Kubernetes and Azure is assumed.



IMPORTANT

High availability of storage in the infrastructure is left to the underlying storage provider.

24.11.2. Before you begin

1. Install **samba-client**, **samba-common**, and **cifs-utils** on all nodes:

```
$ sudo yum install samba-client samba-common cifs-utils
```

2. Enable SELinux booleans on all nodes:

```
$ /usr/sbin/setsebool -P virt_use_samba on
$ /usr/sbin/setsebool -P virt_sandbox_use_samba on
```

3. Run the **mount** command to check **dir_mode** and **file_mode** permissions, for example:

```
$ mount
```

If the **dir_mode** and **file_mode** permissions are set to **0755**, change the default value **0755** to **0777** or **0775**. This manual step is required because the default **dir_mode** and **file_mode** permissions changed from **0777** to **0755** in OpenShift Container Platform 3.9. The following examples show configuration files with the changed values.

Considerations when using MySQL and PostgreSQL with Azure file

- The owner UID of the Azure File mounted directory is different from the UID of a container.
- MySQL containers change the file owner permission in the mounted directory. Because of the mismatch between the owner UID and container process UID, this operation fails. Therefore to run MySQL with Azure File:
 - Specify the Azure File mounted directory UID in the **runAsUser** variable in the PV configuration file.

```
spec:
  containers:
    ...
  securityContext:
    runAsUser: <mounted_dir_uid>
```

- Specify the container process UID under **mountOptions** in the PV configuration file.

```
mountOptions:
  - dir_mode=0700
  - file_mode=0600
  - uid=<container_process_uid>
  - gid=0
```

- PostgreSQL does not work with Azure file. This is because PostgreSQL requires hard links in the Azure File directory, and since Azure File does not support [hard links](#) the pod fails to start.

PV configuration file example

```
# azf-pv.yml
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "azpv"
spec:
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteMany"
  azureFile:
    secretName: azure-secret
    shareName: azftest
    readOnly: false
  mountOptions:
    - dir_mode=0777
    - file_mode=0777
```

Storage class configuration file example

```
$ azure-file-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
parameters:
  storageAccount: ocp39str
  location: centralus
```

24.11.3. Configuring Azure File for regional cloud

While [Azure Disk is compatible with multiple regional clouds](#), Azure File supports only the Azure public cloud, because the endpoint is hard-coded.

24.11.4. Creating the PV



NOTE

Azure File does not support the **Recycle** reclaim policy.

24.11.5. Creating the Azure Storage Account secret

Define the Azure Storage Account name and key in a secret configuration, which is then converted to base64 for use by OpenShift Container Platform.

1. Obtain an Azure Storage Account name and key and encode to base64:

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: azure-secret
type: Opaque
data:
  azurestorageaccountname: azhzdGVzdA==
  azurestorageaccountkey:
eElGMXpKYm5ub2pGTE1Ta0JwNTBteDAyckhzTUsc2pVN21GdDRMMTNob0I3ZHJBYUo4
akQ2K0E0NDNqSm9nVjd5MkZVT2hRQ1dQbU02WWF0SHk3cWc9PQ==

```

2. Save the secret definition to a file, for example ***azure-secret.yaml***, then create the secret:

```
$ oc create -f azure-secret.yaml
```

3. Verify that the secret was created:

```

$ oc get secret azure-secret
NAME          TYPE          DATA      AGE
azure-secret  Opaque        1          23d

```

4. Define the PV in an object definition before creating it in OpenShift Container Platform:

PV object definition using Azure File example

```

apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteMany"
  azureFile: ❸
    secretName: azure-secret ❹
    shareName: example ❺
    readOnly: false ❻

```

- ❶ The name of the volume. This is how it is identified via [PV claims](#) or from pods.
- ❷ The amount of storage allocated to this volume.
- ❸ This defines the volume type being used: **azureFile** plug-in.
- ❹ The name of the secret used.
- ❺ The name of the file share.
- ❻ Defaults to **false** (read/write). **ReadOnly** here forces the **ReadOnly** setting in **VolumeMounts**.

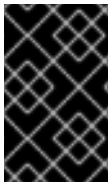
5. Save your definition to a file, for example ***azure-file-pv.yaml***, and create the PV:

```
$ oc create -f azure-file-pv.yaml
persistentvolume "pv0001" created
```

6. Verify that the PV was created:

```
$ oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS      CLAIM
REASON        AGE
pv0001        <none>          5Gi       RWM           Available
2s
```

You can now [request storage using PV claims](#), which can now use your new PV.



IMPORTANT

PV claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a PV from a different namespace causes the pod to fail.

24.12. PERSISTENT STORAGE USING FLEXVOLUME PLUG-INS

24.12.1. Overview

OpenShift Container Platform has built-in [volume plug-ins](#) to use different storage technologies. To use storage from a back-end that does not have a built-in plug-in, you can extend OpenShift Container Platform through FlexVolume drivers and provide [persistent storage](#) to applications.

24.12.2. FlexVolume drivers

A FlexVolume driver is an executable file that resides in a well-defined directory on all machines in the cluster, both masters and nodes. OpenShift Container Platform calls the FlexVolume driver whenever it needs to attach, detach, mount, or unmount a volume represented by a **PersistentVolume** with **flexVolume** as the source.

The first command-line argument of the driver is always an operation name. Other parameters are specific to each operation. Most of the operations take a JavaScript Object Notation (JSON) string as a parameter. This parameter is a complete JSON string, and not the name of a file with the JSON data.

The FlexVolume driver contains:

- All **flexVolume.options**.
- Some options from **flexVolume** prefixed by **kubernetes.io/**, such as **fsType** and **readOnly**.
- The content of the referenced secret, if specified, prefixed by **kubernetes.io/secret/**.

FlexVolume driver JSON input example

```
{
  "fooServer": "192.168.0.1:1234", 1
  "fooVolumeName": "bar",
  "kubernetes.io/fsType": "ext4", 2
}
```



```
"kubernetes.io/readwrite": "ro", 3
"kubernetes.io/secret/<key name>": "<key value>", 4
"kubernetes.io/secret/<another key name>": "<another key value>",
}
```

- 1 All options from `flexVolume.options`.
- 2 The value of `flexVolume.fsType`.
- 3 `ro/rw` based on `flexVolume.readOnly`.
- 4 All keys and their values from the secret referenced by `flexVolume.secretRef`.

OpenShift Container Platform expects JSON data on standard output of the driver. When not specified, the output describes the result of the operation.

FlexVolume Driver Default Output

```
{
  "status": "<Success/Failure/Not supported>",
  "message": "<Reason for success/failure>"
}
```

Exit code of the driver should be `0` for success and `1` for error.

Operations should be idempotent, which means that the attachment of an already attached volume or the mounting of an already mounted volume should result in a successful operation.

The FlexVolume driver can work in two modes:

- with the master-initiated `attach/detach` operation, or
- without the master-initiated `attach/detach` operation.

The **attach/detach** operation is used by the OpenShift Container Platform master to attach a volume to a node and to detach it from a node. This is useful when a node becomes unresponsive for any reason. Then, the master can kill all pods on the node, detach all volumes from it, and attach the volumes to other nodes to resume the applications while the original node is still not reachable.



IMPORTANT

Not all storage back-end supports master-initiated detachment of a volume from another machine.

24.12.2.1. FlexVolume drivers with master-initiated attach/detach

A FlexVolume driver that supports master-controlled `attach/detach` must implement the following operations:

init

Initializes the driver. It is called during initialization of masters and nodes.

- Arguments: none

- Executed on: master, node
- Expected output: default JSON

getvolumename

Returns the unique name of the volume. This name must be consistent among all masters and nodes, because it is used in subsequent **detach** call as **<volume-name>**. Any / characters in the **<volume-name>** are automatically replaced by ~.

- Arguments: **<json>**
- Executed on: master, node
- Expected output: default JSON + **volumeName**:

```
{
  "status": "Success",
  "message": "",
  "volumeName": "foo-volume-bar" 1
}
```

1 The unique name of the volume in storage back-end **foo**.

attach

Attaches a volume represented by the JSON to a given node. This operation should return the name of the device on the node if it is known, that is, if it has been assigned by the storage back-end before it runs. If the device is not known, the device must be found on the node by the subsequent **waitforattach** operation.

- Arguments: **<json> <node-name>**
- Executed on: master
- Expected output: default JSON + **device**, if known:

```
{
  "status": "Success",
  "message": "",
  "device": "/dev/xvda" 1
}
```

1 The name of the device on the node, if known.

waitforattach

Waits until a volume is fully attached to a node and its device emerges. If the previous **attach** operation has returned **<device-name>**, it is provided as an input parameter. Otherwise, **<device-name>** is empty and the operation must find the device on the node.

- Arguments: **<device-name> <json>**
- Executed on: node

- Expected output: default JSON + **device**

```
{
  "status": "Success",
  "message": "",
  "device": "/dev/xvda" 1
}
```

- 1** The name of the device on the node.

detach

Detaches the given volume from a node. **<volume-name>** is the name of the device returned by the **getvolumename** operation. Any / characters in the **<volume-name>** are automatically replaced by ~.

- Arguments: **<volume-name>** **<node-name>**
- Executed on: master
- Expected output: default JSON

isattached

Checks that a volume is attached to a node.

- Arguments: **<json>** **<node-name>**
- Executed on: master
- Expected output: default JSON + **attached**

```
{
  "status": "Success",
  "message": "",
  "attached": true 1
}
```

- 1** The status of attachment of the volume to the node.

mountdevice

Mounts a volume's device to a directory. **<device-name>** is name of the device as returned by the previous **waitforattach** operation.

- Arguments: **<mount-dir>** **<device-name>** **<json>**
- Executed on: node
- Expected output: default JSON

unmountdevice

Unmounts a volume's device from a directory.

- Arguments: **<mount-dir>**

• Arguments: `<mount-dir>`

- Executed on: node

All other operations should return JSON with `{"status": "Not supported"}` and exit code **1**.



NOTE

Master-initiated attach/detach operations are enabled by default in OpenShift Container Platform 3.6. They may work in older versions, but must be explicitly enabled. See [Enabling Controller-managed Attachment and Detachment](#). When not enabled, the attach/detach operations are initiated by a node where the volume should be attached to or detached from. Syntax and all parameters of FlexVolume driver invocations are the same in both cases.

24.12.2.2. FlexVolume drivers without master-initiated attach/detach

FlexVolume drivers that do not support master-controlled attach/detach are executed only on the node and must implement these operations:

init

Initializes the driver. It is called during initialization of all nodes.

- Arguments: none
- Executed on: node
- Expected output: default JSON

mount

Mounts a volume to directory. This can include anything that is necessary to mount the volume, including attaching the volume to the node, finding the its device, and then mounting the device.

- Arguments: `<mount-dir> <json>`
- Executed on: node
- Expected output: default JSON

unmount

Unmounts a volume from a directory. This can include anything that is necessary to clean up the volume after unmounting, such as detaching the volume from the node.

- Arguments: `<mount-dir>`
- Executed on: node
- Expected output: default JSON

All other operations should return JSON with `{"status": "Not supported"}` and exit code **1**.

24.12.3. Installing FlexVolume drivers

To install the FlexVolume driver:

1. Ensure that the executable file exists on all masters and nodes in the cluster.
2. Place the executable file at the volume plug-in path: `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>`.

For example, to install the FlexVolume driver for the storage **foo**, place the executable file at: `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo`.

In OpenShift Container Platform 3.10, since **controller-manager** runs as a static pod, the FlexVolume binary file that performs the attach and detach operations must be a self-contained executable file with no external dependencies.

On Atomic hosts, the default location of the FlexVolume plug-in directory is `/etc/origin/kubelet-plugins/`. You must place the FlexVolume executable file in the `/etc/origin/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>` directory on all master and nodes in the cluster.

24.12.4. Consuming storage using FlexVolume drivers

Use the **PersistentVolume** object to reference the installed storage. Each **PersistentVolume** object in OpenShift Container Platform represents one storage asset, typically a volume, in the storage back-end.

Persistent volume object definition using FlexVolume drivers example

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo ❸
    fsType: "ext4" ❹
    secretRef: foo-secret ❺
    readOnly: true ❻
    options: ❼
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

- ❶ The name of the volume. This is how it is identified through [persistent volume claims](#) or from pods. This name can be different from the name of the volume on back-end storage.
- ❷ The amount of storage allocated to this volume.
- ❸ The name of the driver. This field is mandatory.
- ❹ The file system that is present on the volume. This field is optional.
- ❺ The reference to a secret. Keys and values from this secret are provided to the FlexVolume driver on invocation. This field is optional.

- 6 The read-only flag. This field is optional.
- 7 The additional options for the FlexVolume driver. In addition to the flags specified by the user in the **options** field, the following flags are also passed to the executable:

```
"fsType": "<FS type>",
"readwrite": "<rw>",
"secret/key1": "<secret1>"
...
"secret/keyN": "<secretN>"
```

**NOTE**

Secrets are passed only to mount/unmount call-outs.

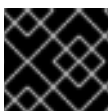
24.13. USING VMWARE VSPHERE VOLUMES FOR PERSISTENT STORAGE

24.13.1. Overview

OpenShift Container Platform supports VMware vSphere's Virtual Machine Disk (VMDK) volumes. You can provision your OpenShift Container Platform cluster with [persistent storage](#) using [VMware vSphere](#). Some familiarity with Kubernetes and VMware vSphere is assumed.

The OpenShift Container Platform [persistent volume \(PV\)](#) framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure. vSphere VMDK volumes can be [provisioned dynamically](#).

PVs are not bound to a single project or namespace; they can be shared across the OpenShift Container Platform cluster. [PV claims](#), however, are specific to a project or namespace and can be requested by users.

**IMPORTANT**

High availability of storage in the infrastructure is left to the underlying storage provider.

Prerequisites

Before creating PVs using vSphere, ensure your OpenShift Container Platform cluster meets the following requirements:

- OpenShift Container Platform must first be [configured for vSphere](#).
- Each node host in the infrastructure must match the vSphere VM name.
- Each node host must be in the same resource group.

IMPORTANT

Create VMDK using one of the following methods before using them.

- Create using **vmkfstools**:

Access ESX through Secure Shell (SSH) and then use following command to create a VMDK volume:

```
vmkfstools -c 2G
/vmfs/volumes/DatastoreName/volumes/myDisk.vmdk
```

- Create using **vmware-vdiskmanager**:

```
shell vmware-vdiskmanager -c -t 0 -s 40GB -a lsilogic
myDisk.vmdk
```

24.13.2. Provisioning VMware vSphere volumes

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is [configured for vSphere](#), all that is required for OpenShift Container Platform and vSphere is a VM folder path, file system type, and the **PersistentVolume** API.

24.13.2.1. Creating persistent volumes

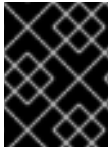
You must define your PV in an object definition before creating it in OpenShift Container Platform:

PV object definition using VMware vSphere example

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 2Gi ❷
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: ❸
    volumePath: "[datastore1] volumes/myDisk" ❹
    fsType: ext4 ❺
```

- ❶ The name of the volume. This must be how it is identified by [PV claims](#) or from pods.
- ❷ The amount of storage allocated to this volume.
- ❸ This defines the volume type being used (**vsphereVolume** plug-in, in this example). The **vsphereVolume** label is used to mount a vSphere VMDK volume into pods. The contents of a volume are preserved when it is unmounted. The volume type supports VMFS and VSAN datastore.
- ❹ This VMDK volume must exist, and you must include brackets ([]) in the volume definition.

- 5 The file system type to mount (for example, **ext4**, **xfs**, and other file-systems).



IMPORTANT

Changing the value of the **fsType** parameter after the volume is formatted and provisioned can result in data loss and pod failure.

To create persistent volumes:

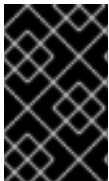
1. Save your definition to a file, for example **vsphere-pv.yaml**, and create the PV:

```
$ oc create -f vsphere-pv.yaml
persistentvolume "pv0001" created
```

2. Verify that the PV was created:

```
$ oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM    REASON
AGE
pv0001        <none>    2Gi       RWO           Available
2s
```

Now you can [request storage using PV claims](#), which can now use your PV.



IMPORTANT

PV claims only exist in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a PV from a different namespace causes the pod to fail.

24.13.2.2. Formatting VMware vSphere volumes

Before OpenShift Container Platform mounts the volume and passes it to a container, it checks that the volume contains a file system as specified by the **fsType** parameter in the PV definition. If the device is not formatted with the file system, all data from the device is erased and the device is automatically formatted with the given file system.

This allows unformatted vSphere volumes to be used as PVs, because OpenShift Container Platform formats them before the first use.

24.14. PERSISTENT STORAGE USING LOCAL VOLUME

24.14.1. Overview

OpenShift Container Platform clusters can be provisioned with [persistent storage](#) by using local volumes. Local persistent volume allows you to access local storage devices such as a disk, partition or directory by using the standard PVC interface.

Local volumes can be used without manually scheduling pods to nodes, because the system is aware of the volume's node constraints. However, local volumes are still subject to the availability of the underlying node and are not suitable for all applications.

**NOTE**

Local volumes is an alpha feature and may change in a future release of OpenShift Container Platform. See [Feature Status\(Local Volume\)](#) section for details on known issues and workarounds.

**WARNING**

Local volumes can only be used as a statically created Persistent Volume.

24.14.2. Provisioning

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. Ensure that OpenShift Container Platform is configured for [Local Volumes](#), before using the **PersistentVolume** API.

24.14.3. Creating Local Persistent Volume Claim

Define the persistent volume claim in an object definition.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: example-local-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi 1
  storageClassName: local-storage 2
```

- 1 The required size of storage volume.
- 2 The name of storage class, which is used for local PVs.

24.14.4. Feature Status**What Works:**

- Creating a PV by specifying a directory with node affinity.
- A Pod using the PVC that is bound to the previously mentioned PV always get scheduled to that node.
- External static provisioner daemonset that discovers local directories, creates, cleans up and deletes PVs.

What does not work:

- Multiple local PVCs in a single pod.
- PVC binding does not consider pod scheduling requirements and may make sub-optimal or incorrect decisions.
 - Workarounds:
 - Run those pods first, which requires local volume.
 - Give the pods high priority.
 - Run a workaround controller that unbinds PVCs for pods that are stuck pending.
- If mounts are added after the external provisioner is started, then external provisioner cannot detect the correct capacity of mounts.
 - Workarounds:
 - Before adding any new mount points, first stop the daemonset, add the new mount points, and then start the daemonset.
- **fsgroup** conflict occurs if multiple pods using the same PVC specify different **fsgroup** 's.

24.15. PERSISTENT STORAGE USING CONTAINER STORAGE INTERFACE (CSI)

24.15.1. Overview

Container Storage Interface (CSI) allows OpenShift Container Platform to consume storage from storage backends that implement the [CSI interface](#) as [persistent storage](#).



IMPORTANT

CSI volumes are currently in Technology Preview and not for production workloads. CSI volumes may change in a future release of OpenShift Container Platform. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See the [Red Hat Technology Preview features support scope](#) for more information.



NOTE

OpenShift Container Platform does not ship with any CSI drivers. It is recommended to use the CSI drivers provided by [community or storage vendors](#).

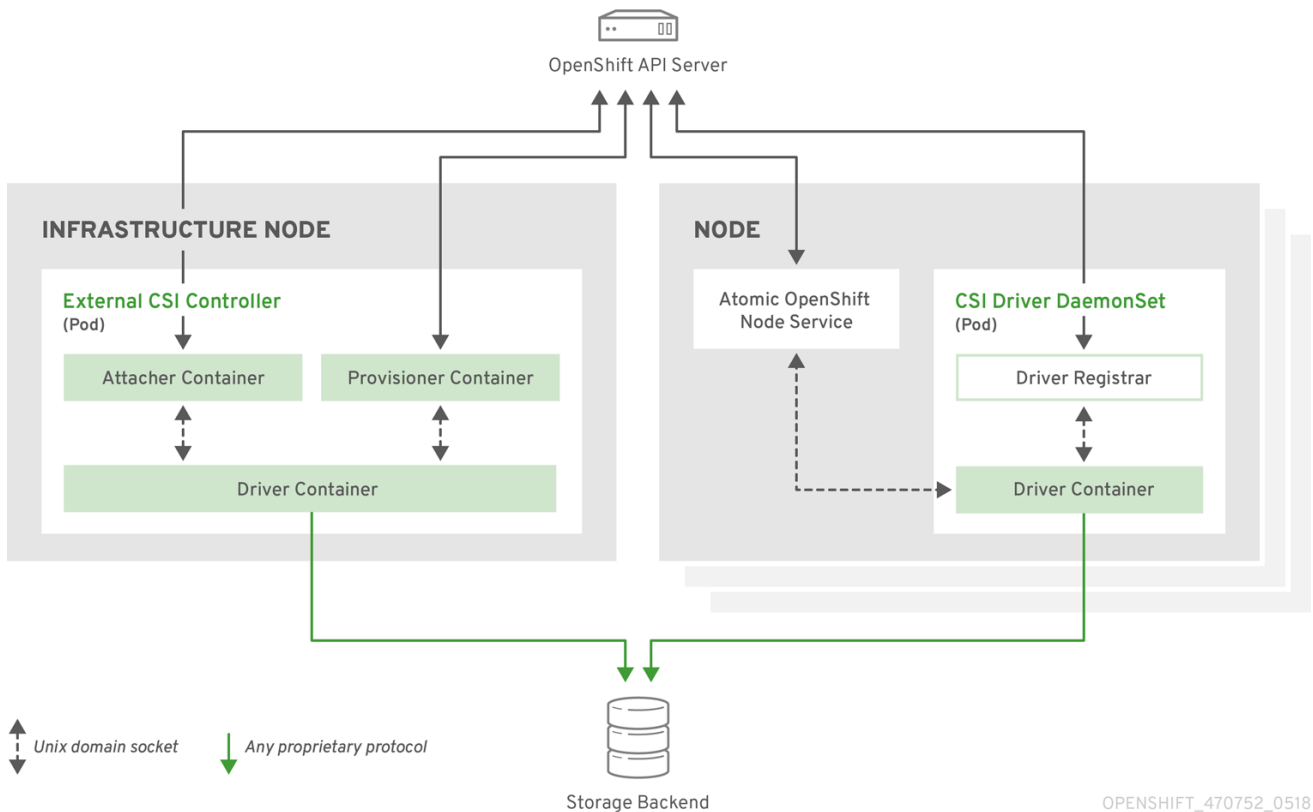
OpenShift Container Platform 3.10 supports version 0.2.0 of the [CSI specification](#).

24.15.2. Architecture

CSI drivers are typically shipped as container images. These containers are not aware of OpenShift Container Platform where they run. To use CSI-compatible storage backend in OpenShift Container Platform, the cluster administrator must deploy several components that serve as a bridge between

OpenShift Container Platform and the storage driver.

The following diagram provides a high-level overview about the components running in pods in the OpenShift Container Platform cluster.



OPENSIFT_470752_0518

It is possible to run multiple CSI drivers for different storage backends. Each driver needs its own external controllers' deployment and DaemonSet with the driver and CSI registrar.

24.15.2.1. External CSI Controllers

External CSI Controllers is a deployment that deploys one or more pods with two containers:

- External CSI attacher container that translates **attach** and **detach** calls from OpenShift Container Platform to respective **ControllerPublish** and **ControllerUnpublish** calls to CSI driver
- External CSI provisioner container that translates **provision** and **delete** calls from OpenShift Container Platform to respective **CreateVolume** and **DeleteVolume** calls to CSI driver
- CSI driver container

The CSI attacher and CSI provisioner containers talk to the CSI driver container using UNIX Domain Sockets, ensuring that no CSI communication leaves the pod. The CSI driver is not accessible from outside of the pod.

**NOTE**

attach, **detach**, **provision**, and **delete** operations typically require the CSI driver to use credentials to the storage backend. Run the CSI controller pods on [infrastructure nodes](#) so the credentials never leak to user processes, even in the event of a catastrophic security breach on a compute node.

**NOTE**

The external attacher must also run for CSI drivers that do not support third-party attach/detach operations. The external attacher will not issue any **ControllerPublish** or **ControllerUnpublish** operations to the CSI driver. However, it still must run to implement the necessary OpenShift Container Platform attachment API.

24.15.2.2. CSI Driver DaemonSet

Finally, the CSI driver DaemonSet runs a pod on every node that allows OpenShift Container Platform to mount storage provided by the CSI driver to the node and use it in user workloads (pods) as persistent volumes (PVs). The pod with the CSI driver installed contains the following containers:

- CSI driver registrar, which registers the CSI driver into the **openshift-node** service running on the node. The **openshift-node** process running on the node then directly connects with the CSI driver using the UNIX Domain Socket available on the node.
- CSI driver.

The CSI driver deployed on the node should have as few credentials to the storage backend as possible. OpenShift Container Platform will only use the node plug-in set of CSI calls such as **NodePublish/NodeUnpublish** and **NodeStage/NodeUnstage** (if implemented).

24.15.3. Example Deployment

Since OpenShift Container Platform does not ship with any CSI driver installed, this example shows how to deploy a community driver for OpenStack Cinder in OpenShift Container Platform.

1. Create a new project where the CSI components will run and a new service account that will run the components. Explicit node selector is used to run the Daemonset with the CSI driver also on master nodes.

```
# oc adm new-project csi --node-selector=""
Now using project "csi" on server "https://example.com:8443".

# oc create serviceaccount cinder-csi
serviceaccount "cinder-csi" created

# oc adm policy add-scc-to-user privileged
system:serviceaccount:csi:cinder-csi
scc "privileged" added to: ["system:serviceaccount:csi:cinder-csi"]
```

2. Apply this YAML file to create the deployment with the external CSI attacher and provisioner and DaemonSet with the CSI driver.

```
# This YAML file contains all API objects that are necessary to run
Cinder CSI
# driver.
```

```

#
# In production, this needs to be in separate files, e.g. service
# account and
# role and role binding needs to be created once.
#
# It server as an example how to use external attacher and external
# provisioner
# images shipped with {product-title} with a community CSI driver.

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["create", "delete", "get", "list", "watch", "update",
"patch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]

---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
subjects:
  - kind: ServiceAccount
    name: cinder-csi
    namespace: csi
roleRef:
  kind: ClusterRole
  name: cinder-csi-role
  apiGroup: rbac.authorization.k8s.io

---
apiVersion: v1
data:
  cloud.conf:

```

```

W0dsb2JhbF0KYXV0aC11cmwgPSBodHRwczovL2V4YW1wbGUuY29tOjEzMdAwL3YyLjAv
CnVzZXJuYW1lID0gYWxhZGRpbGpwYXNzd29yZCA9IG9wZW5zZXNhbwUKdGVuYW50LWlk
ID0gZTBmYTg1YjZhMDY0NDM5NTlkMmQzYjQ5NzE3NGJlZDYKcmVnaW9uID0gcmVnaW9u
T25lCg== 1
kind: Secret
metadata:
  creationTimestamp: null
  name: cloudconfig
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cinder-csi-controller
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cinder-csi-controllers
  template:
    metadata:
      labels:
        app: cinder-csi-controllers
    spec:
      serviceAccount: cinder-csi
      containers:
        - name: csi-attacher
          image: registry.access.redhat.com/openshift3/csi-
attacher:v3.10
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election"
            - "--leader-election-namespace=$(MY_NAMESPACE)"
            - "--leader-election-identity=$(MY_NAME)"
          env:
            - name: MY_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: MY_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-provisioner
          image: registry.access.redhat.com/openshift3/csi-
provisioner:v3.10
          args:
            - "--v=5"
            - "--provisioner=csi-cinderplugin"
            - "--csi-address=$(ADDRESS)"
          env:

```

```

      - name: ADDRESS
        value: /csi/csi.sock
    volumeMounts:
      - name: socket-dir
        mountPath: /csi
  - name: cinder-driver
    image: quay.io/jsafrane/cinder-csi-plugin
    command: [ "/bin/cinder-csi-plugin" ]
    args:
      - "--nodeid=$(NODEID)"
      - "--endpoint=unix://$(ADDRESS)"
      - "--cloud-config=/etc/cloudconfig/cloud.conf"
    env:
      - name: NODEID
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      - name: ADDRESS
        value: /csi/csi.sock
    volumeMounts:
      - name: socket-dir
        mountPath: /csi
      - name: cloudconfig
        mountPath: /etc/cloudconfig
  volumes:
    - name: socket-dir
      emptyDir: {}
    - name: cloudconfig
      secret:
        secretName: cloudconfig
---

kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: cinder-csi-ds
spec:
  selector:
    matchLabels:
      app: cinder-csi-driver
  template:
    metadata:
      labels:
        app: cinder-csi-driver
    spec:
      nodeSelector:
        role: node
      serviceAccount: cinder-csi
      containers:
        - name: csi-driver-registrar
          image: registry.access.redhat.com/openshift3/csi-driver-
registrar:v3.10
          securityContext:
            privileged: true
          args:

```

```

- "--v=5"
- "--csi-address=$(ADDRESS)"
env:
- name: ADDRESS
  value: /csi/csi.sock
- name: KUBE_NODE_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
volumeMounts:
- name: socket-dir
  mountPath: /csi
- name: cinder-driver
securityContext:
  privileged: true
  capabilities:
    add: ["SYS_ADMIN"]
  allowPrivilegeEscalation: true
image: quay.io/jsafrane/cinder-csi-plugin
command: [ "/bin/cinder-csi-plugin" ]
args:
- "--nodeid=$(NODEID)"
- "--endpoint=unix://$(ADDRESS)"
- "--cloud-config=/etc/cloudconfig/cloud.conf"
env:
- name: NODEID
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: ADDRESS
  value: /csi/csi.sock
volumeMounts:
- name: socket-dir
  mountPath: /csi
- name: cloudconfig
  mountPath: /etc/cloudconfig
- name: mountpoint-dir
  mountPath:
/var/lib/origin/openshift.local.volumes/pods/
  mountPropagation: "Bidirectional"
- name: cloud-metadata
  mountPath: /var/lib/cloud/data/
- name: dev
  mountPath: /dev
volumes:
- name: cloud-metadata
  hostPath:
    path: /var/lib/cloud/data/
- name: socket-dir
  hostPath:
    path: /var/lib/kubelet/plugins/csi-cinderplugin
    type: DirectoryOrCreate
- name: mountpoint-dir
  hostPath:
    path: /var/lib/origin/openshift.local.volumes/pods/
    type: Directory

```



```

- name: cloudconfig
  secret:
    secretName: cloudconfig
- name: dev
  hostPath:
    path: /dev

```

- 1 Replace with **cloud.conf** for your OpenStack deployment, as described in [OpenStack configuration](#). For example, the Secret can be generated using the **oc create secret generic cloudconfig --from-file cloud.conf --dry-run -o yaml**.

24.15.4. Dynamic Provisioning

Dynamic provisioning of persistent storage depends on the capabilities of the CSI driver and underlying storage backend. The provider of the CSI driver should document how to create a StorageClass in OpenShift Container Platform and the parameters available for configuration.

As seen in the OpenStack Cinder example, you can deploy this StorageClass to enable dynamic provisioning. The following example creates a new default storage class that ensures that all PVCs that do not require any special storage class are provisioned by the installed CSI driver:

```

# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cinder
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-cinderplugin
parameters:
EOF

```

24.15.5. Usage

Once the CSI driver is deployed and the StorageClass for dynamic provisioning is created, OpenShift Container Platform is ready to use CSI. The following example installs a default MySQL template without any changes to the template:

```

# oc new-app oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

# oc get pvc
[root@host-172-16-120-9 ~]# oc get pvc
NAME                STATUS      VOLUME
CAPACITY  ACCESS MODES  STORAGECLASS  AGE
mysql          Bound        kubernetes-dynamic-pv-3271ffcb4e1811e8    1Gi
RWO                cinder        3s

```

24.16. PERSISTENT STORAGE USING OPENSTACK MANILA

24.16.1. Overview



IMPORTANT

Persistent volume (PV) provisioning using OpenStack Manila is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

OpenShift Container Platform is capable of provisioning PVs using the [OpenStack Manila](#) shared file system service.

It is assumed the OpenStack Manila service has been correctly set up and is accessible from the OpenShift Container Platform cluster. Only the NFS share types can be provisioned.

Familiarity with [PVs](#), [persistent volume claims \(PVCs\)](#), [dynamic provisioning](#), and [RBAC authorization](#) is recommended.

24.16.2. Installation and Setup

The feature is provided by an external provisioner. You must install and configure it in the OpenShift Container Platform cluster.

24.16.2.1. Starting the External Provisioner

The external provisioner service is distributed as a container image and can be run in the OpenShift Container Platform cluster as usual.

To allow the containers managing the API objects, configure the required role-based access control (RBAC) rules as a cluster administrator:

1. Create a **ServiceAccount**:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: manila-provisioner-runner
```

2. Create a **ClusterRole**:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: manila-provisioner-role
rules:
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["get", "list", "watch", "create", "delete"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
```

```

resources: ["storageclasses"]
verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["events"]
  verbs: ["list", "watch", "create", "update", "patch"]

```

3. Bind the rules via **ClusterRoleBinding**:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manila-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: manila-provisioner-role
subjects:
- kind: ServiceAccount
  name: manila-provisioner-runner
  namespace: default

```

4. Create a new **StorageClass**:

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "manila-share"
provisioner: "externalstorage.k8s.io/manila"
parameters:
  type: "default" ❶
  zones: "nova" ❷

```

❶ The [Manila share type](#) the provisioner will create for the volume.

❷ Set of Manila availability zones that the volume might be created in.

Configure the provisioner to connect, authenticate, and authorize to the Manila service using environment variables. Select the appropriate combination of environment variables for your installation from the following list:

```

OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_NAME
OS_TENANT_NAME

```

```

OS_USERID
OS_PASSWORD
OS_AUTH_URL
OS_TENANT_ID

```

```

OS_USERNAME
OS_PASSWORD

```

```
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_NAME
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_ID
```

To pass the variables to the provisioner, use a **Secret**. The following example shows a **Secret** configured for the first variables combination

```
apiVersion: v1
kind: Secret
metadata:
  name: manila-provisioner-env
type: Opaque
data:
  os_username: <base64 encoded Manila username>
  os_password: <base64 encoded password>
  os_auth_url: <base64 encoded OpenStack Keystone URL>
  os_domain_name: <base64 encoded Manila service Domain>
  os_tenant_name: <base64 encoded Manila service Tenant/Project name>
```



NOTE

Newer OpenStack versions use "project" instead of "tenant," however, the environment variables used by the provisioner must use **TENANT** in their names.

The last step is to start the provisioner itself, for example, using a deployment:

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: manila-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: manila-provisioner
    spec:
      serviceAccountName: manila-provisioner-runner
      containers:
        - image: "registry.access.redhat.com:8888/openshift3/manila-
provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          name: manila-provisioner
          env:
            - name: "OS_USERNAME"
              valueFrom:
```

```

      secretKeyRef:
        name: manila-provisioner-env
        key: os_username
- name: "OS_PASSWORD"
  valueFrom:
    secretKeyRef:
      name: manila-provisioner-env
      key: os_password
- name: "OS_AUTH_URL"
  valueFrom:
    secretKeyRef:
      name: manila-provisioner-env
      key: os_auth_url
- name: "OS_DOMAIN_NAME"
  valueFrom:
    secretKeyRef:
      name: manila-provisioner-env
      key: os_domain_name
- name: "OS_TENANT_NAME"
  valueFrom:
    secretKeyRef:
      name: manila-provisioner-env
      key: os_tenant_name

```

24.16.3. Usage

After the provisioner is running, you can provision PVs using a PVC and the corresponding `StorageClass`:

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: manila-nfs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G
  storageClassName: manila-share

```

The **PersistentVolumeClaim** is then bound to a **PersistentVolume** backed by the newly provisioned Manila share. When the **PersistentVolumeClaim** and subsequently the **PersistentVolume** are deleted, the provisioner deletes and unexports the Manila share.

24.17. DYNAMIC PROVISIONING AND CREATING STORAGE CLASSES

24.17.1. Overview

The *StorageClass* resource object describes and classifies storage that can be requested, as well as provides a means for passing parameters for **dynamically provisioned storage** on demand. *StorageClass* objects can also serve as a management mechanism for controlling different levels of

storage and access to the storage. Cluster Administrators (**cluster-admin**) or Storage Administrators (**storage-admin**) define and create the *StorageClass* objects that users can request without needing any intimate knowledge about the underlying storage volume sources.

The OpenShift Container Platform [persistent volume](#) framework enables this functionality and allows administrators to provision a cluster with persistent storage. The framework also gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Many storage types are available for use as persistent volumes in OpenShift Container Platform. While all of them can be statically provisioned by an administrator, some types of storage are created dynamically using the built-in provider and plug-in APIs.



NOTE

To enable dynamic provisioning, add the **openshift_master_dynamic_provisioning_enabled** variable to the **[OSEv3:vars]** section of the Ansible inventory file and set its value to **True**.

```
[OSEv3:vars]
```

```
openshift_master_dynamic_provisioning_enabled=True
```

24.17.2. Available dynamically provisioned plug-ins

OpenShift Container Platform provides the following *provisioner plug-ins*, which have generic implementations for dynamic provisioning that use the cluster's configured provider's API to create new storage resources:

Storage Type	Provisioner Plug-in Name	Required Configuration	Notes
OpenStack Cinder	kubernetes.io/cinder	Configuring for OpenStack	
AWS Elastic Block Store (EBS)	kubernetes.io/aws-ebs	Configuring for AWS	For dynamic provisioning when using multiple clusters in different zones, tag each node with Key=kubernetes.io/cluster/xxxx, Value=clusterid where xxxx and clusterid are unique per cluster. In versions prior to 3.6, this was Key=KubernetesCluster, Value=clusterid .

Storage Type	Provisioner Plug-in Name	Required Configuration	Notes
GCE Persistent Disk (gcePD)	kubernetes.io/gce-pd	Configuring for GCE	In multi-zone configurations, it is advisable to run one Openshift cluster per GCE project to avoid PVs from getting created in zones where no node from current cluster exists.
GlusterFS	kubernetes.io/glusterfs	Configuring GlusterFS	
Ceph RBD	kubernetes.io/rbd	Configuring Ceph RBD	
Trident from NetApp	netapp.io/trident	Configuring for Trident	Storage orchestrator for NetApp ONTAP, SolidFire, and E-Series storage.
VMWare vSphere	kubernetes.io/vsphere-volume	Getting Started with vSphere and Kubernetes	
Azure Disk	kubernetes.io/azure-disk	Configuring for Azure	



IMPORTANT

Any chosen provisioner plug-in also requires configuration for the relevant cloud, host, or third-party provider as per the relevant documentation.

24.17.3. Defining a StorageClass

StorageClass objects are currently a globally scoped object and need to be created by **cluster-admin** or **storage-admin** users.



NOTE

For GCE and AWS, a default *StorageClass* is created during OpenShift Container Platform installation. You can [change the default StorageClass](#) or delete it.

There are currently six plug-ins that are supported. The following sections describe the basic object definition for a *StorageClass* and specific examples for each of the supported plug-in types.

24.17.3.1. Basic StorageClass object definition

StorageClass Basic object definition

StorageClass Basic Object definition

```

kind: StorageClass ❶
apiVersion: storage.k8s.io/v1 ❷
metadata:
  name: foo ❸
  annotations: ❹
  ...
provisioner: kubernetes.io/plug-in-type ❺
parameters: ❻
  param1: value
  ...
  paramN: value

```

- ❶ (required) The API object type.
- ❷ (required) The current apiVersion.
- ❸ (required) The name of the StorageClass.
- ❹ (optional) Annotations for the StorageClass
- ❺ (required) The type of provisioner associated with this storage class.
- ❻ (optional) The parameters required for the specific provisioner, this will change from plug-in to plug-in.

24.17.3.2. StorageClass annotations

To set a *StorageClass* as the cluster-wide default:

```
storageclass.kubernetes.io/is-default-class: "true"
```

This enables any Persistent Volume Claim (PVC) that does not specify a specific volume to automatically be provisioned through the *default* StorageClass

**NOTE**

Beta annotation **storageclass.beta.kubernetes.io/is-default-class** is still working. However it will be removed in a future release.

To set a *StorageClass* description:

```
kubernetes.io/description: My StorageClass Description
```

24.17.3.3. OpenStack Cinder object definition**cinder-storageclass.yaml**

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:

```



```

name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast ❶
  availability: nova ❷
  fsType: ext4 ❸

```

- ❶ Volume type created in Cinder. Default is empty.
- ❷ Availability Zone. If not specified, volumes are generally round-robin across all active zones where the OpenShift Container Platform cluster has a node.
- ❸ File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

24.17.3.4. AWS ElasticBlockStore (EBS) object definition

aws-ebs-storageclass.yaml

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 ❶
  zone: us-east-1d ❷
  iopsPerGB: "10" ❸
  encrypted: "true" ❹
  kmsKeyId: keyvalue ❺
  fsType: ext4 ❻

```

- ❶ Select from **io1**, **gp2**, **sc1**, **st1**. The default is **gp2**. See [AWS documentation](#) for valid Amazon Resource Name (ARN) values.
- ❷ AWS zone. If no zone is specified, volumes are generally round-robin across all active zones where the OpenShift Container Platform cluster has a node. Zone and zones parameters must not be used at the same time.
- ❸ Only for **io1** volumes. I/O operations per second per GiB. The AWS volume plug-in multiplies this with the size of the requested volume to compute IOPS of the volume. The value cap is 20,000 IOPS, which is the maximum supported by AWS. See [AWS documentation](#) for further details.
- ❹ Denotes whether to encrypt the EBS volume. Valid values are **true** or **false**.
- ❺ Optional. The full ARN of the key to use when encrypting the volume. If none is supplied, but **encrypted** is set to **true**, then AWS generates a key. See [AWS documentation](#) for a valid ARN value.
- ❻ File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

24.17.3.5. GCE PersistentDisk (gcePD) object definition

gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard ❶
  zone: us-central1-a ❷
  zones: us-central1-a, us-central1-b, us-east1-b ❸
  fsType: ext4 ❹
```

- ❶ Select either **pd-standard** or **pd-ssd**. The default is **pd-ssd**.
- ❷ GCE zone. If no zone is specified, volumes are generally round-robin across all active zones where the OpenShift Container Platform cluster has a node. Zone and zones parameters must not be used at the same time.
- ❸ A comma-separated list of GCE zone(s). If no zone is specified, volumes are generally round-robin across all active zones where the OpenShift Container Platform cluster has a node. Zone and zones parameters must not be used at the same time.
- ❹ File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

24.17.3.6. GlusterFS object definition

glusterfs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://127.0.0.1:8081" ❶
  restuser: "admin" ❷
  secretName: "heketi-secret" ❸
  secretNamespace: "default" ❹
  gidMin: "40000" ❺
  gidMax: "50000" ❻
```

- ❶ **heketi** (volume management REST service for Gluster) URL that provisions GlusterFS volumes on demand. The general format should be **{http/https}://{IPaddress}:{Port}**. This is a mandatory parameter for the GlusterFS dynamic provisioner. If the heketi service is exposed as a routable service in the OpenShift Container Platform, it will have a resolvable fully qualified domain name (FQDN) and heketi service URL.

- 2 heketi user who has access to create volumes. Usually "admin".
- 3 Identification of a Secret that contains a user password to use when talking to heketi. Optional; an empty password will be used when both **secretNamespace** and **secretName** are omitted. The provided secret must be of type **"kubernetes.io/glusterfs"**.
- 4 The namespace of mentioned **secretName**. Optional; an empty password will be used when both **secretNamespace** and **secretName** are omitted. The provided secret must be of type **"kubernetes.io/glusterfs"**.
- 5 Optional. The minimum value of the GID range for volumes of this StorageClass.
- 6 Optional. The maximum value of the GID range for volumes of this StorageClass.



NOTE

When the **gidMin** and **gidMax** values are not specified, their defaults are 2000 and 2147483647, respectively. Each dynamically provisioned volume will be given a GID in this range (**gidMin-gidMax**). This GID is released from the pool when the respective volume is deleted. The GID pool is per StorageClass. If two or more storage classes have GID ranges that overlap there may be duplicate GIDs dispatched by the provisioner.

When heketi authentication is used, a Secret containing the admin key should also exist:

heketi-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlwYXNzd29yZA== 1
  type: kubernetes.io/glusterfs
```

- 1 base64 encoded password, for example: **echo -n "mypassword" | base64**



NOTE

When the PVs are dynamically provisioned, the GlusterFS plug-in automatically creates an Endpoints and a headless Service named **gluster-dynamic-**<claimname>****. When the PVC is deleted, these dynamic resources are deleted automatically.

24.17.3.7. Ceph RBD object definition

ceph-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
```

```

provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.16.153.105:6789 ❶
  adminId: admin ❷
  adminSecretName: ceph-secret ❸
  adminSecretNamespace: kube-system ❹
  pool: kube ❺
  userId: kube ❻
  userSecretName: ceph-secret-user ❼
  fsType: ext4 ❽

```

- ❶ Ceph monitors, comma-delimited. It is required.
- ❷ Ceph client ID that is capable of creating images in the pool. Default is "admin".
- ❸ Secret Name for **adminId**. It is required. The provided secret must have type "kubernetes.io/rbd".
- ❹ The namespace for **adminSecret**. Default is "default".
- ❺ Ceph RBD pool. Default is "rbd".
- ❻ Ceph client ID that is used to map the Ceph RBD image. Default is the same as **adminId**.
- ❼ The name of Ceph Secret for **userId** to map Ceph RBD image. It must exist in the same namespace as PVCs. It is required.
- ❽ File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

24.17.3.8. Trident object definition

trident.yaml

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: netapp.io/trident ❶
parameters: ❷
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"

```

Trident uses the parameters as selection criteria for the different pools of storage that are registered with it. Trident itself is configured separately.

- ❶ For more information about installing Trident with OpenShift Container Platform, see the [Trident documentation](#).
- ❷ For more information about supported parameters, see the [storage attributes](#) section of the Trident documentation.

24.17.3.9. VMware vSphere object definition

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume ❶
parameters:
  diskformat: thin ❷
```

- ❶ For more information about using VMWare vSphere with OpenShift Container Platform, see the [VMWare vSphere documentation](#).
- ❷ **diskformat: thin, zeroedthick and eagerzeroedthick**. See vSphere docs for details. Default: **thin**

24.17.3.10. Azure File object definition

To configure Azure file dynamic provisioning:

1. Create the role in the user's project:

```
$ cat azf-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project name>
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "get", "delete"]
```

2. Create the role binding to the **persistent-volume-binder** service account in the **kube-system** project:

```
$ cat azf-rolebind.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: system:controller:persistent-volume-binder
subjects:
- kind: ServiceAccount
  name: persistent-volume-binder
  namespace: kube-system
```

3. Add the service account as **admin** to the user's project:

```
$ oc policy add-role-to-user admin system:serviceaccount:kube-
system:persistent-volume-binder -n <user's project>
```

4. Create a storage class for the Azure file:

```
$ cat azfsc.yaml | oc create -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azfsc
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
```

The user can now create a PVC that uses this storage class.

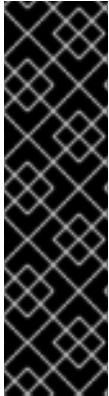
24.17.3.11. Azure Disk object definition

azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name ❶
  storageaccounttype: Standard_LRS ❷
  kind: Dedicated ❸
```

- ❶ Azure storage account name. This must reside in the same resource group as the cluster. If a storage account is specified, the **location** is ignored. If a storage account is not specified, a new storage account gets created in the same resource group as the cluster. If you are specifying a **storageAccount**, the value for **kind** must be **Dedicated**.
- ❷ Azure storage account SKU tier. Default is empty. **Note:** Premium VM can attach both *Standard_LRS* and *Premium_LRS* disks, Standard VM can only attach *Standard_LRS* disks, Managed VM can only attach managed disks, and unmanaged VM can only attach unmanaged disks.
- ❸ Possible values are **Shared** (default), **Dedicated**, and **Managed**.
 - a. If **kind** is set to **Shared**, Azure creates all unmanaged disks in a few shared storage accounts in the same resource group as the cluster.
 - b. If **kind** is set to **Managed**, Azure creates new managed disks.
 - c. If **kind** is set to **Dedicated** and a **storageAccount** is specified, Azure uses the specified storage account for the new unmanaged disk in the same resource group as the cluster. For this to work:
 - a. The specified storage account must be in the same region

- The specified storage account must be in the same region.
 - Azure Cloud Provider must have a write access to the storage account.
- d. If **kind** is set to **Dedicated** and a **storageAccount** is not specified, Azure creates a new dedicated storage account for the new unmanaged disk in the same resource group as the cluster.



IMPORTANT

Azure StorageClass is revised in OpenShift Container Platform version 3.7. If you upgraded from a previous version, either:

- specify the property **kind: dedicated** to continue using the Azure StorageClass created before the upgrade. Or,
- add the location parameter (for example, "**location**": "**southcentralus**",) in the **azure.conf** file to use the default property **kind: shared**. Doing this creates new storage accounts for future use.

24.17.4. Changing the default StorageClass

If you are using GCE and AWS, use the following process to change the default StorageClass:

1. List the StorageClass:

```
$ oc get storageclass
```

NAME	TYPE
gp2 (default)	kubernetes.io/aws-ebs 1
standard	kubernetes.io/gce-pd

1 (**default**) denotes the default StorageClass.

2. Change the value of the annotation **storageclass.kubernetes.io/is-default-class** to **false** for the default StorageClass:

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": \
  {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. Make another StorageClass the default by adding or modifying the annotation as **storageclass.kubernetes.io/is-default-class=true**.

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": \
  {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. Verify the changes:

```
$ oc get storageclass
```

NAME	TYPE
gp2	kubernetes.io/aws-ebs
standard (default)	kubernetes.io/gce-pd



24.17.5. Additional information and examples

- [Examples and uses of StorageClasses for Dynamic Provisioning](#)
- [Examples and uses of StorageClasses without Dynamic Provisioning](#)

24.18. VOLUME SECURITY

24.18.1. Overview

This topic provides a general guide on pod security as it relates to volume security. For information on pod-level security in general, see [Managing Security Context Constraints \(SCC\)](#) and the [Security Context Constraint](#) concept topic. For information on the OpenShift Container Platform persistent volume (PV) framework in general, see the [Persistent Storage](#) concept topic.

Accessing persistent storage requires coordination between the cluster and/or storage administrator and the end developer. The cluster administrator creates PVs, which abstract the underlying physical storage. The developer creates pods and, optionally, PVCs, which bind to PVs, based on matching criteria, such as capacity.

Multiple persistent volume claims (PVCs) within the same project can bind to the same PV. However, once a PVC binds to a PV, that PV cannot be bound by a claim outside of the first claim's project. If the underlying storage needs to be accessed by multiple projects, then each project needs its own PV, which can point to the same physical storage. In this sense, a bound PV is tied to a project. For a detailed PV and PVC example, see the guide for [WordPress and MySQL using NFS](#).

For the cluster administrator, granting pods access to PVs involves:

- knowing the group ID and/or user ID assigned to the actual storage,
- understanding SELinux considerations, and
- ensuring that these IDs are allowed in the range of legal IDs defined for the project and/or the SCC that matches the requirements of the pod.

Group IDs, the user ID, and SELinux values are defined in the **SecurityContext** section in a pod definition. Group IDs are global to the pod and apply to all containers defined in the pod. User IDs can also be global, or specific to each container. Four sections control access to volumes:

- [supplementalGroups](#)
- [fsGroup](#)
- [runAsUser](#)
- [selinuxOptions](#)

24.18.2. SCCs, Defaults, and Allowed Ranges

SCCs influence whether or not a pod is given a default user ID, **fsGroup** ID, supplemental group ID, and SELinux label. They also influence whether or not IDs supplied in the pod definition (or in the image) will be validated against a range of allowable IDs. If validation is required and fails, then the pod will also fail.

SCCs define strategies, such as **runAsUser**, **supplementalGroups**, and **fsGroup**. These strategies help decide whether the pod is authorized. Strategy values set to **RunAsAny** are essentially stating that the pod can do what it wants regarding that strategy. Authorization is skipped for that strategy and no OpenShift Container Platform default is produced based on that strategy. Therefore, IDs and SELinux labels in the resulting container are based on container defaults instead of OpenShift Container Platform policies.

For a quick summary of **RunAsAny**:

- Any ID defined in the pod definition (or image) is allowed.
- Absence of an ID in the pod definition (and in the image) results in the container assigning an ID, which is **root** (0) for Docker.
- No SELinux labels are defined, so Docker will assign a unique label.

For these reasons, SCCs with **RunAsAny** for ID-related strategies should be protected so that ordinary developers do not have access to the SCC. On the other hand, SCC strategies set to **MustRunAs** or **MustRunAsRange** trigger ID validation (for ID-related strategies), and cause default values to be supplied by OpenShift Container Platform to the container when those values are not supplied directly in the pod definition or image.

CAUTION

Allowing access to SCCs with a **RunAsAny fsGroup** strategy can also prevent users from accessing their block devices. Pods need to specify an **fsGroup** in order to take over their block devices. Normally, this is done when the SCC **fsGroup** strategy is set to **MustRunAs**. If a user's pod is assigned an SCC with a **RunAsAny fsGroup** strategy, then the user may face **permission denied** errors until they discover that they need to specify an **fsGroup** themselves.

SCCs may define the range of allowed IDs (user or groups). If range checking is required (for example, using **MustRunAs**) and the allowable range is not defined in the SCC, then the project determines the ID range. Therefore, projects support ranges of allowable ID. However, unlike SCCs, projects do not define strategies, such as **runAsUser**.

Allowable ranges are helpful not only because they define the boundaries for container IDs, but also because the minimum value in the range becomes the default value for the ID in question. For example, if the SCC ID strategy value is **MustRunAs**, the minimum value of an ID range is **100**, and the ID is absent from the pod definition, then 100 is provided as the default for this ID.

As part of pod admission, the SCCs available to a pod are examined (roughly, in priority order followed by most restrictive) to best match the requests of the pod. Setting a SCC's strategy type to **RunAsAny** is less restrictive, whereas a type of **MustRunAs** is more restrictive. All of these strategies are evaluated. To see which SCC was assigned to a pod, use the **oc get pod** command:

```
# oc get pod <pod_name> -o yaml
...
metadata:
  annotations:
    openshift.io/scc: nfs-scc ❶
    name: nfs-pod1 ❷
    namespace: default ❸
...
```

- 1 Name of the SCC that the pod used (in this case, a custom SCC).
- 2 Name of the pod.
- 3 Name of the project. "Namespace" is interchangeable with "project" in OpenShift Container Platform. See [Projects and Users](#) for details.

It may not be immediately obvious which SCC was matched by a pod, so the command above can be very useful in understanding the UID, supplemental groups, and SELinux relabeling in a live container.

Any SCC with a strategy set to **RunAsAny** allows specific values for that strategy to be defined in the pod definition (and/or image). When this applies to the user ID (**runAsUser**) it is prudent to restrict access to the SCC to prevent a container from being able to run as root.

Because pods often match the **restricted** SCC, it is worth knowing the security this entails. The **restricted** SCC has the following characteristics:

- User IDs are constrained due to the **runAsUser** strategy being set to **MustRunAsRange**. This forces user ID validation.
- Because a range of allowable user IDs is not defined in the SCC (see **oc export scc restricted** for more details), the project's **openshift.io/sa.scc.uid-range** range will be used for range checking and for a default ID, if needed.
- A default user ID is produced when a user ID is not specified in the pod definition and the matching SCC's **runAsUser** is set to **MustRunAsRange**.
- An SELinux label is required (**seLinuxContext** set to **MustRunAs**), which uses the project's default MCS label.
- **fsGroup** IDs are constrained to a single value due to the **FSGroup** strategy being set to **MustRunAs**, which dictates that the value to use is the minimum value of the first range specified.
- Because a range of allowable **fsGroup** IDs is not defined in the SCC, the minimum value of the project's **openshift.io/sa.scc.supplemental-groups** range (or the same range used for user IDs) will be used for validation and for a default ID, if needed.
- A default **fsGroup** ID is produced when a **fsGroup** ID is not specified in the pod and the matching SCC's **FSGroup** is set to **MustRunAs**.
- Arbitrary supplemental group IDs are allowed because no range checking is required. This is a result of the **supplementalGroups** strategy being set to **RunAsAny**.
- Default supplemental groups are not produced for the running pod due to **RunAsAny** for the two group strategies above. Therefore, if no groups are defined in the pod definition (or in the image), the container(s) will have no supplemental groups predefined.

The following shows the **default** project and a custom SCC (**my-custom-scc**), which summarizes the interactions of the SCC and the project:

```
$ oc get project default -o yaml 1
...
metadata:
  annotations: 2
```

```

openshift.io/sa.scc.mcs: s0:c1,c0 3
openshift.io/sa.scc.supplemental-groups: 1000000000/10000 4
openshift.io/sa.scc.uid-range: 1000000000/10000 5

$ oc get scc my-custom-scc -o yaml
...
fsGroup:
  type: MustRunAs 6
  ranges:
    - min: 5000
      max: 6000
runAsUser:
  type: MustRunAsRange 7
  uidRangeMin: 1000100000
  uidRangeMax: 1000100999
seLinuxContext: 8
  type: MustRunAs
  SELinuxOptions: 9
    user: <selinux-user-name>
    role: ...
    type: ...
    level: ...
supplementalGroups:
  type: MustRunAs 10
  ranges:
    - min: 5000
      max: 6000

```

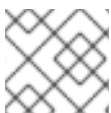
- 1 **default** is the name of the project.
- 2 Default values are only produced when the corresponding SCC strategy is not **RunAsAny**.
- 3 SELinux default when not defined in the pod definition or in the SCC.
- 4 Range of allowable group IDs. ID validation only occurs when the SCC strategy is **RunAsAny**. There can be more than one range specified, separated by commas. See below for [supported formats](#).
- 5 Same as <4> but for user IDs. Also, only a single range of user IDs is supported.
- 6 10 **MustRunAs** enforces group ID range checking and provides the container's groups default. Based on this SCC definition, the default is 5000 (the minimum ID value). If the range was omitted from the SCC, then the default would be 1000000000 (derived from the project). The other supported type, **RunAsAny**, does not perform range checking, thus allowing any group ID, and produces no default groups.
- 7 **MustRunAsRange** enforces user ID range checking and provides a UID default. Based on this SCC, the default UID is 1000100000 (the minimum value). If the minimum and maximum range were omitted from the SCC, the default user ID would be 1000000000 (derived from the project). **MustRunAsNonRoot** and **RunAsAny** are the other supported types. The range of allowed IDs can be defined to include any user IDs required for the target storage.
- 8 When set to **MustRunAs**, the container is created with the SCC's SELinux options, or the MCS default defined in the project. A type of **RunAsAny** indicates that SELinux context is not required, and, if not defined in the pod, is not set in the container.

- 9 The SELinux user name, role name, type, and labels can be defined here.

Two formats are supported for allowed ranges:

1. **M/N**, where **M** is the starting ID and **N** is the count, so the range becomes **M** through (and including) **M+N-1**.
2. **M-N**, where **M** is again the starting ID and **N** is the ending ID. The default group ID is the starting ID in the first range, which is **1000000000** in this project. If the SCC did not define a minimum group ID, then the project's default ID is applied.

24.18.3. Supplemental Groups



NOTE

Read [SCCs, Defaults, and Allowed Ranges](#) before working with supplemental groups.

TIP

It is generally preferable to use group IDs (supplemental or [fsGroup](#)) to gain access to persistent storage versus using [user IDs](#).

Supplemental groups are regular Linux groups. When a process runs in Linux, it has a UID, a GID, and one or more supplemental groups. These attributes can be set for a container's main process. The **supplementalGroups** IDs are typically used for controlling access to shared storage, such as NFS and GlusterFS, whereas [fsGroup](#) is used for controlling access to block storage, such as Ceph RBD and iSCSI.

The OpenShift Container Platform shared storage plug-ins mount volumes such that the POSIX permissions on the mount match the permissions on the target storage. For example, if the target storage's owner ID is **1234** and its group ID is **5678**, then the mount on the host node and in the container will have those same IDs. Therefore, the container's main process must match one or both of those IDs in order to access the volume.

For example, consider the following NFS export.

On an OpenShift Container Platform node:



NOTE

showmount requires access to the ports used by **rpcbind** and **rpc.mount** on the NFS server

```
# showmount -e <nfs-server-ip-or-hostname>
Export list for f21-nfs.vm:
/opt/nfs *
```

On the NFS server:

```
# cat /etc/exports
/opt/nfs *(rw,sync,root_squash)
...
```

```
# ls -lZ /opt/nfs -d
drwx----- . 1000100001 5555 unconfined_u:object_r:usr_t:s0 /opt/nfs
```

The `/opt/nfs/` export is accessible by UID **1000100001** and the group **5555**. In general, containers should not run as root. So, in this NFS example, containers which are not run as UID **1000100001** and are not members the group **5555** will not have access to the NFS export.

Often, the SCC matching the pod does not allow a specific user ID to be specified, thus using supplemental groups is a more flexible way to grant storage access to a pod. For example, to grant NFS access to the export above, the group **5555** can be defined in the pod definition:

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: ...
    volumeMounts:
    - name: nfs 1
      mountPath: /usr/share/... 2
  securityContext: 3
    supplementalGroups: [5555] 4
  volumes:
  - name: nfs 5
    nfs:
      server: <nfs_server_ip_or_host>
      path: /opt/nfs 6
```

- 1** Name of the volume mount. Must match the name in the **volumes** section.
- 2** NFS export path as seen in the container.
- 3** Pod global security context. Applies to all containers inside the pod. Each container can also define its **securityContext**, however group IDs are global to the pod and cannot be defined for individual containers.
- 4** Supplemental groups, which is an array of IDs, is set to 5555. This grants group access to the export.
- 5** Name of the volume. Must match the name in the **volumeMounts** section.
- 6** Actual NFS export path on the NFS server.

All containers in the above pod (assuming the matching SCC or project allows the group **5555**) will be members of the group **5555** and have access to the volume, regardless of the container's user ID. However, the assumption above is critical. Sometimes, the SCC does not define a range of allowable group IDs but instead requires group ID validation (a result of **supplementalGroups** set to **MustRunAs**). Note that this is **not** the case for the **restricted** SCC. The project will not likely allow a group ID of **5555**, unless the project has been customized to access this NFS export. So, in this scenario, the above pod will fail because its group ID of **5555** is not within the SCC's or the project's range of allowed group IDs.

Supplemental Groups and Custom SCCs

To remedy the situation in [the previous example](#), a custom SCC can be created such that:

- a minimum and max group ID are defined,
- ID range checking is enforced, and
- the group ID of **5555** is allowed.

It is often better to create a new SCC rather than modifying a predefined SCC, or changing the range of allowed IDs in the predefined projects.

The easiest way to create a new SCC is to export an existing SCC and customize the YAML file to meet the requirements of the new SCC. For example:

1. Use the **restricted** SCC as a template for the new SCC:

```
$ oc export scc restricted > new-scc.yaml
```

2. Edit the **new-scc.yaml** file to your desired specifications.
3. Create the new SCC:

```
$ oc create -f new-scc.yaml
```



NOTE

The **oc edit scc** command can be used to modify an instantiated SCC.

Here is a fragment of a new SCC named **nfs-scc**:

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false ❶
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc ❷
priority: 9 ❸
...
supplementalGroups:
  type: MustRunAs ❹
  ranges:
    - min: 5000 ❺
      max: 6000
  ...
```

- ❶ The **allow** booleans are the same as for the **restricted** SCC.
- ❷ Name of the new SCC.
- ❸ Numerically larger numbers have greater priority. Nil or omitted is the lowest priority. Higher priority SCCs sort before lower priority SCCs and thus have a better chance of matching a new pod.

- 4 **supplementalGroups** is a strategy and it is set to **MustRunAs**, which means group ID checking is required.
- 5 Multiple ranges are supported. The allowed group ID range here is 5000 through 5999, with the default supplemental group being 5000.

When the same pod shown earlier runs against this new SCC (assuming, of course, the pod matches the new SCC), it will start because the group **5555**, supplied in the pod definition, is now allowed by the custom SCC.

24.18.4. fsGroup



NOTE

Read [SCCs, Defaults, and Allowed Ranges](#) before working with supplemental groups.

TIP

It is generally preferable to use group IDs ([supplemental](#) or **fsGroup**) to gain access to persistent storage versus using [user IDs](#).

fsGroup defines a pod's "file system group" ID, which is added to the container's supplemental groups. The **supplementalGroups** ID applies to shared storage, whereas the **fsGroup** ID is used for block storage.

Block storage, such as Ceph RBD, iSCSI, and various cloud storage, is typically dedicated to a single pod which has requested the block storage volume, either directly or using a PVC. Unlike shared storage, block storage is taken over by a pod, meaning that user and group IDs supplied in the pod definition (or image) are applied to the actual, physical block device. Typically, block storage is not shared.

A **fsGroup** definition is shown below in the following pod definition fragment:

```
kind: Pod
...
spec:
  containers:
  - name: ...
    securityContext: 1
      fsGroup: 5555 2
    ...
```

- 1 As with **supplementalGroups**, **fsGroup** must be defined globally to the pod, not per container.
- 2 5555 will become the group ID for the volume's group permissions and for all new files created in the volume.

As with **supplementalGroups**, all containers in the above pod (assuming the matching SCC or project allows the group **5555**) will be members of the group **5555**, and will have access to the block volume, regardless of the container's user ID. If the pod matches the **restricted** SCC, whose **fsGroup** strategy is **MustRunAs**, then the pod will fail to run. However, if the SCC has its **fsGroup** strategy set to

RunAsAny, then any **fsGroup** ID (including **5555**) will be accepted. Note that if the SCC has its **fsGroup** strategy set to **RunAsAny** and no **fsGroup** ID is specified, the "taking over" of the block storage does not occur and permissions may be denied to the pod.

fsGroups and Custom SCCs

To remedy the situation in the previous example, a custom SCC can be created such that:

- a minimum and maximum group ID are defined,
- ID range checking is enforced, and
- the group ID of **5555** is allowed.

It is better to create new SCCs versus modifying a predefined SCC, or changing the range of allowed IDs in the predefined projects.

Consider the following fragment of a new SCC definition:

```
# oc export scc new-scc
...
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs ①
  ranges: ②
    - max: 6000
      min: 5000 ③
  ...
```

- ① **MustRunAs** triggers group ID range checking, whereas **RunAsAny** does not require range checking.
- ② The range of allowed group IDs is 5000 through, and including, 5999. Multiple ranges are supported but not used. The allowed group ID range here is 5000 through 5999, with the default **fsGroup** being 5000.
- ③ The minimum value (or the entire range) can be omitted from the SCC, and thus range checking and generating a default value will defer to the project's **openshift.io/sa.scc.supplemental-groups** range. **fsGroup** and **supplementalGroups** use the same group field in the project; there is not a separate range for **fsGroup**.

When the pod shown above runs against this new SCC (assuming, of course, the pod matches the new SCC), it will start because the group **5555**, supplied in the pod definition, is allowed by the custom SCC. Additionally, the pod will "take over" the block device, so when the block storage is viewed by a process outside of the pod, it will actually have **5555** as its group ID.

A list of volumes supporting block ownership include:

- AWS Elastic Block Store
- OpenStack Cinder
- Ceph RBD

- GCE Persistent Disk
- iSCSI
- emptyDir
- gitRepo

**NOTE**

This list is potentially incomplete.

24.18.5. User IDs**NOTE**

Read [SCCs, Defaults, and Allowed Ranges](#) before working with supplemental groups.

TIP

It is generally preferable to use group IDs ([supplemental](#) or [fsGroup](#)) to gain access to persistent storage versus using user IDs.

User IDs can be defined in the container image or in the pod definition. In the pod definition, a single user ID can be defined globally to all containers, or specific to individual containers (or both). A user ID is supplied as shown in the pod definition fragment below:

```
spec:
  containers:
  - name: ...
    securityContext:
      runAsUser: 1000100001
```

ID 1000100001 in the above is container-specific and matches the owner ID on the export. If the NFS export's owner ID was **54321**, then that number would be used in the pod definition. Specifying **securityContext** outside of the container definition makes the ID global to all containers in the pod.

Similar to group IDs, user IDs may be validated according to policies set in the SCC and/or project. If the SCC's **runAsUser** strategy is set to **RunAsAny**, then any user ID defined in the pod definition or in the image is allowed.

**WARNING**

This means even a UID of **0** (root) is allowed.

If, instead, the **runAsUser** strategy is set to **MustRunAsRange**, then a supplied user ID will be validated against a range of allowed IDs. If the pod supplies no user ID, then the default ID is set to the minimum value of the range of allowable user IDs.

Returning to the earlier [NFS example](#), the container needs its UID set to **1000100001**, which is shown in the pod fragment above. Assuming the **default** project and the **restricted** SCC, the pod's requested user ID of 1000100001 will not be allowed, and therefore the pod will fail. The pod fails because:

- it requests **1000100001** as its user ID,
- all available SCCs use **MustRunAsRange** for their **runAsUser** strategy, so UID range checking is required, and
- **1000100001** is not included in the SCC or in the project's user ID range.

To remedy this situation, a new SCC can be created with the appropriate user ID range. A new project could also be created with the appropriate user ID range defined. There are also other, less-preferred options:

- The **restricted** SCC could be modified to include **1000100001** within its minimum and maximum user ID range. This is not recommended as you should avoid modifying the predefined SCCs if possible.
- The **restricted** SCC could be modified to use **RunAsAny** for the **runAsUser** value, thus eliminating ID range checking. This is *strongly* not recommended, as containers could run as root.
- The **default** project's UID range could be changed to allow a user ID of **1000100001**. This is not generally advisable because only a single range of user IDs can be specified, and thus other pods may not run if the range is altered.

User IDs and Custom SCCs

It is good practice to avoid modifying the predefined SCCs if possible. The preferred approach is to create a custom SCC that better fits an organization's security needs, or [create a new project](#) that supports the desired user IDs.

To remedy the situation in the previous example, a custom SCC can be created such that:

- a minimum and maximum user ID is defined,
- UID range checking is still enforced, and
- the UID of **1000100001** is allowed.

For example:

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false ❶
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc ❷
priority: 9 ❸
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsRange ❹
```

```
uidRangeMax: 1000100001 5
uidRangeMin: 1000100001
...
```

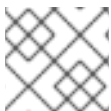
- 1 The **allowXX** bools are the same as for the **restricted** SCC.
- 2 The name of this new SCC is **nfs-scc**.
- 3 Numerically larger numbers have greater priority. Nil or omitted is the lowest priority. Higher priority SCCs sort before lower priority SCCs, and thus have a better chance of matching a new pod.
- 4 The **runAsUser** strategy is set to **MustRunAsRange**, which means UID range checking is enforced.
- 5 The UID range is 1000100001 through 1000100001 (a range of one value).

Now, with **runAsUser: 1000100001** shown in the previous pod definition fragment, the pod matches the new **nfs-scc** and is able to run with a UID of 1000100001.

24.18.6. SELinux Options

All predefined SCCs, except for the **privileged** SCC, set the **seLinuxContext** to **MustRunAs**. So the SCCs most likely to match a pod's requirements will force the pod to use an SELinux policy. The SELinux policy used by the pod can be defined in the pod itself, in the image, in the SCC, or in the project (which provides the default).

SELinux labels can be defined in a pod's **securityContext.seLinuxOptions** section, and supports **user**, **role**, **type**, and **level**:



NOTE

Level and MCS label are used interchangeably in this topic.

```
...
securityContext: 1
  seLinuxOptions:
    level: "s0:c123,c456" 2
...
```

- 1 **level** can be defined globally for the entire pod, or individually for each container.
- 2 SELinux level label.

Here are fragments from an SCC and from the **default** project:

```
$ oc export scc scc-name
...
seLinuxContext:
  type: MustRunAs 1

# oc export project default
...
```

```

metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c1,c0 2
  ...

```

1 **MustRunAs** causes volume relabeling.

2 If the label is not provided in the pod or in the SCC, then the default comes from the project.

All predefined SCCs, except for the **privileged** SCC, set the **seLinuxContext** to **MustRunAs**. This forces pods to use MCS labels, which can be defined in the pod definition, the image, or provided as a default.

The SCC determines whether or not to require an SELinux label and can provide a default label. If the **seLinuxContext** strategy is set to **MustRunAs** and the pod (or image) does not define a label, OpenShift Container Platform defaults to a label chosen from the SCC itself or from the project.

If **seLinuxContext** is set to **RunAsAny**, then no default labels are provided, and the container determines the final label. In the case of Docker, the container will use a unique MCS label, which will not likely match the labeling on existing storage mounts. Volumes which support SELinux management will be relabeled so that they are accessible by the specified label and, depending on how exclusionary the label is, only that label.

This means two things for unprivileged containers:

- The volume is given a type that is accessible by unprivileged containers. This type is usually **container_file_t** in Red Hat Enterprise Linux (RHEL) version 7.5 and later. This type treats volumes as container content. In previous RHEL versions, RHEL 7.4, 7.3, and so forth, the volume is given the **svirt_sandbox_file_t** type.
- If a **level** is specified, the volume is labeled with the given MCS label.

For a volume to be accessible by a pod, the pod must have both categories of the volume. So a pod with **s0:c1,c2** will be able to access a volume with **s0:c1,c2**. A volume with **s0** will be accessible by all pods.

If pods fail authorization, or if the storage mount is failing due to permissions errors, then there is a possibility that SELinux enforcement is interfering. One way to check for this is to run:

```
# ausearch -m avc --start recent
```

This examines the log file for AVC (Access Vector Cache) errors.

24.19. SELECTOR-LABEL VOLUME BINDING

24.19.1. Overview

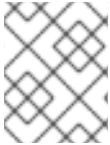
This guide provides the steps necessary to enable binding of persistent volume claims (PVCs) to persistent volumes (PVs) via **selector** and **label** attributes. By implementing selectors and labels, regular users are able to target [provisioned storage](#) by identifiers defined by a cluster administrator.

24.19.2. Motivation

In cases of statically provisioned storage, developers seeking persistent storage are required to know a

handful identifying attributes of a PV in order to deploy and bind a PVC. This creates several problematic situations. Regular users might have to contact a cluster administrator to either deploy the PVC or provide the PV values. PV attributes alone do not convey the intended use of the storage volumes, nor do they provide methods by which volumes can be grouped.

Selector and label attributes can be used to abstract away PV details from the user while providing cluster administrators a way of identifying volumes by a descriptive and customizable tag. Through the selector-label method of binding, users are only required to know which labels are defined by the administrator.



NOTE

The selector-label feature is currently only available for *statically* provisioned storage and is currently not implemented for storage provisioned dynamically.

24.19.3. Deployment

This section reviews how to define and deploy PVCs.

24.19.3.1. Prerequisites

1. A running OpenShift Container Platform 3.3+ cluster
2. A volume provided by a supported [storage provider](#)
3. A user with a cluster-admin role binding

24.19.3.2. Define the Persistent Volume and Claim

1. As the **cluster-admin** user, define the PV. For this example, we will be using a [GlusterFS](#) volume. See the appropriate [storage provider](#) for your provider's configuration.

Example 24.9. Persistent Volume with Labels

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: ❶
    volume-type: ssd
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

- ❶ A PVC whose selectors match *all* of a PV's labels will be bound, assuming a PV is available.

2. Define the PVC:

Example 24.10. Persistent Volume Claim with Selectors

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: ❶
    matchLabels: ❷
      volume-type: ssd
      aws-availability-zone: us-east-1

```

❶ Begin **selectors** section.❷ List all labels by which the user is requesting storage. Must match *all* labels of targeted PV.**24.19.3.3. Deploy the Persistent Volume and Claim**As the **cluster-admin** user, create the persistent volume:**Example 24.11. Create the Persistent Volume**

```

# oc create -f gluster-pv.yaml
persistentVolume "gluster-volume" created

# oc get pv
NAME                                LABELS                                CAPACITY    ACCESSMODES    STATUS
CLAIM      REASON    AGE
gluster-volume    map[]    2147483648    RWX
Available                                2s

```

Once the PV is created, any user whose selectors match *all* its labels can create their PVC.**Example 24.12. Create the Persistent Volume Claim**

```

# oc create -f gluster-pvc.yaml
persistentVolumeClaim "gluster-claim" created
# oc get pvc
NAME            LABELS            STATUS    VOLUME
gluster-claim    Bound            gluster-volume

```

24.20. ENABLING CONTROLLER-MANAGED ATTACHMENT AND DETACHMENT

24.20.1. Overview

As of OpenShift Container Platform 3.4, administrators can enable the controller running on the cluster's master to manage volume attach and detach operations on behalf of a set of nodes, as opposed to letting them manage their own volume attach and detach operations.

Enabling controller-managed attachment and detachment has the following benefits:

- If a node is lost, volumes that were attached to it can be detached by the controller and reattached elsewhere.
- Credentials for attaching and detaching do not need to be made present on every node, improving security.

As of OpenShift Container Platform 3.6, controller-managed attachment and detachment is the default setting.

24.20.2. Determining What Is Managing Attachment and Detachment

If a node has set the annotation `volumes.kubernetes.io/controller-managed-attach-detach` on itself, then its attach and detach operations are being managed by the controller. The controller will automatically inspect all nodes for this annotation and act according to whether it is present or not. Therefore, you may inspect the node for this annotation to determine if it has enabled controller-managed attach and detach.

To further ensure that the node is opting for controller-managed attachment and detachment, its logs can be searched for the following line:

```
Setting node annotation to enable volume controller attach/detach
```

If the above line is not found, the logs should instead contain:

```
Controller attach/detach is disabled for this node; Kubelet will attach and detach volumes
```

To check from the controller's end that it is managing a particular node's attach and detach operations, the logging level must first be set to at least **4**. Then, the following line should be found:

```
processVolumesInUse for node <node_hostname>
```

For information on how to view logs and configure logging levels, see [Configuring Logging Levels](#).

24.20.3. Configuring Nodes to Enable Controller-managed Attachment and Detachment

Enabling controller-managed attachment and detachment is done by configuring individual nodes to opt in and disable their own node-level attachment and detachment management. See [Node Configuration Files](#) for information on what node configuration file to edit and add the following:

```
kubeletArguments:
  enable-controller-attach-detach:
    - "true"
```

Once a node is configured, it must be restarted for the setting to take effect.

24.21. PERSISTENT VOLUME SNAPSHOTS

24.21.1. Overview



IMPORTANT

Persistent Volume Snapshots are a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

Many storage systems provide the ability to create "snapshots" of a persistent volume (PV) to protect against data loss. The external snapshot controller and provisioner provide means to use the feature in the OpenShift Container Platform cluster and handle volume snapshots through the OpenShift Container Platform API.

This document describes the current state of volume snapshot support in OpenShift Container Platform. Familiarity with [PVs](#), [persistent volume claims \(PVCs\)](#), and [dynamic provisioning](#) is recommended.

24.21.2. Features

- Create snapshot of a **PersistentVolume** bound to a **PersistentVolumeClaim**
- List existing **VolumeSnapshots**
- Delete existing **VolumeSnapshot**
- Create a new **PersistentVolume** from an existing **VolumeSnapshot**
- Supported **PersistentVolume** [types](#):
 - AWS Elastic Block Store (EBS)
 - Google Compute Engine (GCE) Persistent Disk (PD)

24.21.3. Installation and Setup

The external controller and provisioner are the external components that provide volume snapshotting. These external components run in the cluster. The controller is responsible for creating, deleting, and reporting events on volume snapshots. The provisioner creates new **PersistentVolumes** from the volume snapshots. See [Create Snapshot](#) and [Restore Snapshot](#) for more information.

24.21.3.1. Starting the External Controller and Provisioner

The external controller and provisioner services are distributed as container images and can be run in the OpenShift Container Platform cluster as usual. There are also RPM versions for the controller and provisioner.

To allow the containers managing the API objects, the necessary role-based access control (RBAC) rules need to be configured by the administrator:

1. Create a **ServiceAccount** and **ClusterRole**:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: snapshot-controller-runner
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: snapshot-controller-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    verbs: ["create", "list", "watch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshotdatas"]
    verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
```

2. Bind the rules via **ClusterRoleBinding**:

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: snapshot-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: snapshot-controller-role
subjects:
  - kind: ServiceAccount
    name: snapshot-controller-runner
    namespace: default
```

■

If the external controller and provisioner are deployed in Amazon Web Services (AWS), they must be able to authenticate using the access key. To provide the credential to the pod, the administrator creates a new secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: awskeys
type: Opaque
data:
  access-key-id: <base64 encoded AWS_ACCESS_KEY_ID>
  secret-access-key: <base64 encoded AWS_SECRET_ACCESS_KEY>
```

The AWS deployment of the external controller and provisioner containers (note that both pod containers use the secret to access the AWS cloud provider API):

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.access.redhat.com/openshift3/snapshot-controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: secret-access-key
        - name: snapshot-provisioner
          image: "registry.access.redhat.com/openshift3/snapshot-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
```

```

      valueFrom:
        secretKeyRef:
          name: awskeys
          key: access-key-id
-   name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: awskeys
          key: secret-access-key

```

For GCE, there is no need to use secrets to access the GCE cloud provider API. The administrator can proceed with the deployment:

```

kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.access.redhat.com/openshift3/snapshot-
controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
        - name: snapshot-provisioner
          image: "registry.access.redhat.com/openshift3/snapshot-
provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]

```

24.21.3.2. Managing Snapshot Users

Depending on the cluster configuration, it might be necessary to allow non-administrator users to manipulate the **VolumeSnapshot** objects on the API server. This can be done by creating a **ClusterRole** bound to a particular user or group.

For example, assume the user 'alice' needs to work with snapshots in the cluster. The cluster administrator completes the following steps:

1. Define a new **ClusterRole**:

```

apiVersion: v1
kind: ClusterRole
metadata:
  name: volumesnapshot-admin
rules:

```

```

- apiGroups:
  - "volumesnapshot.external-storage.k8s.io"
  attributeRestrictions: null
  resources:
  - volumesnapshots
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch

```

2. Bind the cluster role to the user 'alice' by creating a **ClusterRole** binding object:

```

apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: volumesnapshot-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: volumesnapshot-admin
subjects:
- kind: User
  name: alice

```



NOTE

This is only an example of API access configuration. The **VolumeSnapshot** objects behave similar to other OpenShift Container Platform API objects. See the [API access control documentation](#) for more information on managing the API RBAC.

24.21.4. Lifecycle of a Volume Snapshot and Volume Snapshot Data

24.21.4.1. Persistent Volume Claim and Persistent Volume

The **PersistentVolumeClaim** is bound to a **PersistentVolume**. The **PersistentVolume** type must be one of the snapshot supported persistent volume types.

24.21.4.1.1. Snapshot Promoter

To create a **StorageClass**:

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: snapshot-promoter
provisioner: volumesnapshot.external-storage.k8s.io/snapshot-promoter

```

This **StorageClass** is necessary to restore a **PersistentVolume** from a **VolumeSnapshot** that was previously created.

24.21.4.2. Create Snapshot

To take a snapshot of a PV, create a new **VolumeSnapshot** object:

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  persistentVolumeClaimName: ebs-pvc
```

persistentVolumeClaimName is the name of the **PersistentVolumeClaim** bound to a **PersistentVolume**. This particular PV is snapshotted.

A **VolumeSnapshotData** object is then automatically created based on the **VolumeSnapshot**. The relationship between **VolumeSnapshot** and **VolumeSnapshotData** is similar to the relationship between **PersistentVolumeClaim** and **PersistentVolume**.

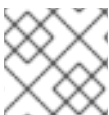
Depending on the PV type, the operation might go through several phases, which are reflected by the **VolumeSnapshot** status:

1. The new **VolumeSnapshot** object is created.
2. The controller starts the snapshot operation. The snapshotted **PersistentVolume** might need to be frozen and the applications paused.
3. The storage system finishes creating the snapshot (the snapshot is "cut") and the snapshotted **PersistentVolume** might return to normal operation. The snapshot itself is not yet ready. The last status condition is of **Pending** type with status value **True**. A new **VolumeSnapshotData** object is created to represent the actual snapshot.
4. The newly created snapshot is complete and ready to use. The last status condition is of **Ready** type with status value **True**.



IMPORTANT

It is the user's responsibility to ensure data consistency (stop the pod/application, flush caches, freeze the file system, and so on).



NOTE

In case of error, the **VolumeSnapshot** status is appended with an **Error** condition.

To display the **VolumeSnapshot** status:

```
$ oc get volumesnapshot -o yaml
```

The status is displayed.

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
```

```

kind: VolumeSnapshot
metadata:
  clusterName: ""
  creationTimestamp: 2017-09-19T13:58:28Z
  generation: 0
  labels:
    Timestamp: "1505829508178510973"
  name: snapshot-demo
  namespace: default
  resourceVersion: "780"
  selfLink: /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/default/volumesnapshots/snapshot-demo
  uid: 9cc5da57-9d42-11e7-9b25-90b11c132b3f
spec:
  persistentVolumeClaimName: ebs-pvc
  snapshotDataName: k8s-volume-snapshot-9cc8813e-9d42-11e7-8bed-
90b11c132b3f
status:
  conditions:
  - lastTransitionTime: null
    message: Snapshot created successfully
    reason: ""
    status: "True"
    type: Ready
  creationTimestamp: null

```

24.21.4.3. Restore Snapshot

To restore a PV from a **VolumeSnapshot**, create a PVC:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-pv-provisioning-demo
  annotations:
    snapshot.alpha.kubernetes.io/snapshot: snapshot-demo
spec:
  storageClassName: snapshot-promoter

```

annotations: snapshot.alpha.kubernetes.io/snapshot is the name of the **VolumeSnapshot** to be restored. **storageClassName: StorageClass** is created by the administrator for restoring **VolumeSnapshots**.

A new **PersistentVolume** is created and bound to the **PersistentVolumeClaim**. The process may take several minutes depending on the PV type.

24.21.4.4. Delete Snapshot

To delete a **snapshot -demo**:

```
$ oc delete volumesnapshot/snapshot-demo
```

The **VolumeSnapshotData** bound to the **VolumeSnapshot** is automatically deleted.

CHAPTER 25. PERSISTENT STORAGE EXAMPLES

25.1. OVERVIEW

The following sections provide detailed, comprehensive instructions on setting up and configuring common storage use cases. These examples cover both the administration of persistent volumes and their security, and how to claim against the volumes as a user of the system.

- [Sharing an NFS PV Across Two Pods](#)
- [Ceph-RBD Block Storage Volume](#)
- [Shared Storage Using a GlusterFS Volume](#)
- [Dynamic Provisioning Storage Using GlusterFS](#)
- [Mounting a PV to Privileged Pods](#)
- [Backing Docker Registry with GlusterFS Storage](#)
- [Binding Persistent Volumes by Labels](#)
- [Using StorageClasses for Dynamic Provisioning](#)
- [Using StorageClasses for Existing Legacy Storage](#)
- [Configuring Azure Blob Storage for Integrated Docker Registry](#)

25.2. SHARING AN NFS MOUNT ACROSS TWO PERSISTENT VOLUME CLAIMS

25.2.1. Overview

The following use case describes how a cluster administrator wanting to leverage shared storage for use by two separate containers would configure the solution. This example highlights the use of NFS, but can easily be adapted to other shared storage types, such as GlusterFS. In addition, this example will show configuration of pod security as it relates to shared storage.

[Persistent Storage Using NFS](#) provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and using NFS as persistent storage. This topic shows an end-to-end example of using an existing NFS cluster and OpenShift Container Platform persistent store, and assumes an existing NFS server and exports exist in your OpenShift Container Platform infrastructure.



NOTE

All **oc** commands are executed on the OpenShift Container Platform master host.

25.2.2. Creating the Persistent Volume

Before creating the PV object in OpenShift Container Platform, the persistent volume (PV) file is defined:

Example 25.1. Persistent Volume Object Definition Using NFS

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteMany ❸
  persistentVolumeReclaimPolicy: Retain ❹
  nfs: ❺
    path: /opt/nfs ❻
    server: nfs.f22 ❼
    readOnly: false

```

- ❶ The name of the PV, which is referenced in pod definitions or displayed in various **oc** volume commands.
- ❷ The amount of storage allocated to this volume.
- ❸ **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control.
- ❹ The volume reclaim policy **Retain** indicates that the volume will be preserved after the pods accessing it terminates.
- ❺ This defines the volume type being used, in this case the **NFS** plug-in.
- ❻ This is the NFS mount path.
- ❼ This is the NFS server. This can also be specified by IP address.

Save the PV definition to a file, for example **nfs-pv.yaml**, and create the persistent volume:

```

# oc create -f nfs-pv.yaml
persistentvolume "nfs-pv" created

```

Verify that the persistent volume was created:

```

# oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS    CLAIM
REASON      AGE
nfs-pv      <none>         1Gi       RWX           Available
37s

```

25.2.3. Creating the Persistent Volume Claim

A persistent volume claim (PVC) specifies the desired access mode and storage capacity. Currently, based on only these two attributes, a PVC is bound to a single PV. Once a PV is bound to a PVC, that PV is essentially tied to the PVC's project and cannot be bound to by another PVC. There is a one-to-

one mapping of PVs and PVCs. However, multiple pods in the same project can use the same PVC. This is the use case we are highlighting in this example.

Example 25.2. PVC Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc ❶
spec:
  accessModes:
    - ReadWriteMany ❷
  resources:
    requests:
      storage: 1Gi ❸
```

- ❶ The claim name is referenced by the pod under its **volumes** section.
- ❷ As mentioned above for PVs, the **accessModes** do not enforce access right, but rather act as labels to match a PV to a PVC.
- ❸ This claim will look for PVs offering **1Gi** or greater capacity.

Save the PVC definition to a file, for example **nfs-pvc.yaml**, and create the PVC:

```
# oc create -f nfs-pvc.yaml
persistentvolumeclaim "nfs-pvc" created
```

Verify that the PVC was created and bound to the expected PV:

```
# oc get pvc
NAME          LABELS    STATUS    VOLUME    CAPACITY    ACCESSMODES
AGE
nfs-pvc       <none>    Bound     nfs-pv     1Gi         RWX
24s
```

❶

- ❶ The claim, **nfs-pvc**, was bound to the **nfs-pv** PV.

25.2.4. Ensuring NFS Volume Access

Access is necessary to a node in the NFS server. On this node, examine the NFS export mount:

```
[root@nfs nfs]# ls -lZ /opt/nfs/
total 8
-rw-r--r--. 1 root 100003 system_u:object_r:usr_t:s0 10 Oct 12 23:27
test2b
```

❶ ❷

- 1 the owner has ID 0.
- 2 the group has ID 100003.

In order to access the NFS mount, the container must match the SELinux label, and either run with a UID of 0, or with 100003 in its supplemental groups range. Gain access to the volume by matching the NFS mount's groups, which will be defined in the pod definition below.

By default, SELinux does not allow writing from a pod to a remote NFS server. To enable writing to NFS volumes with SELinux enforcing on each node, run:

```
# setsebool -P virt_use_nfs on
```

25.2.5. Creating the Pod

A pod definition file or a template file can be used to define a pod. Below is a pod specification that creates a single container and mounts the NFS volume for read-write access:

Example 25.3. Pod Object Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-nfs-pod 1
  labels:
    name: hello-openshift-nfs-pod
spec:
  containers:
    - name: hello-openshift-nfs-pod
      image: openshift/hello-openshift 2
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol 3
          mountPath: /usr/share/nginx/html 4
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: nfs-pvc 6
```

- 1 The name of this pod as displayed by `oc get pod`.
- 2 The image run by this pod.
- 3 The name of the volume. This name must be the same in both the **containers** and **volumes** sections.
- 4 The mount path as seen in the container.

- 5 The group ID to be assigned to the container.
- 6 The PVC that was created in the previous step.

Save the pod definition to a file, for example *nfs.yaml*, and create the pod:

```
# oc create -f nfs.yaml
pod "hello-openshift-nfs-pod" created
```

Verify that the pod was created:

```
# oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hello-openshift-nfs-pod	1/1	Running	0	4s

More details are shown in the **oc describe pod** command:

```
[root@ose70 nfs]# oc describe pod hello-openshift-nfs-pod
Name:      hello-openshift-nfs-pod
Namespace: default 1
Image(s):   fedora/S3
Node:       ose70.rh7/192.168.234.148 2
Start Time: Mon, 21 Mar 2016 09:59:47 -0400
Labels:     name=hello-openshift-nfs-pod
Status:     Running
Reason:
Message:
IP:         10.1.0.4
Replication Controllers: <none>
Containers:
  hello-openshift-nfs-pod:
    Container ID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    Image:      fedora/S3
    Image ID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    QoS Tier:
      cpu:      BestEffort
      memory:   BestEffort
    State:      Running
      Started:  Mon, 21 Mar 2016 09:59:49 -0400
    Ready:      True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type      Status
  Ready     True
Volumes:
  nfsvol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName: nfs-pvc 3
```

```

    ReadOnly: false
  default-token-a06zb:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-a06zb
Events: 4
  FirstSeen LastSeen Count From          SubobjectPath
Reason  Message
-----
4m 4m 1 {scheduler }
Scheduled Successfully assigned hello-openshift-nfs-pod to ose70.rh7
4m 4m 1 {kubelet ose70.rh7} implicitly required container POD
Pulled Container image "openshift3/ose-pod:v3.1.0.4" already present on
machine
4m 4m 1 {kubelet ose70.rh7} implicitly required container POD
Created Created with docker id 866a37108041
4m 4m 1 {kubelet ose70.rh7} implicitly required container POD
Started Started with docker id 866a37108041
4m 4m 1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Pulled Container image "fedora/S3" already present on machine
4m 4m 1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Created Created with docker id a3292104d6c2
4m 4m 1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Started Started with docker id a3292104d6c2

```

- 1 The project (namespace) name.
- 2 The IP address of the OpenShift Container Platform node running the pod.
- 3 The PVC name used by the pod.
- 4 The list of events resulting in the pod being launched and the NFS volume being mounted. The container will not start correctly if the volume cannot mount.

There is more internal information, including the SCC used to authorize the pod, the pod's user and group IDs, the SELinux label, and more, shown in the **oc get pod <name> -o yaml** command:

```

[root@ose70 nfs]# oc get pod hello-openshift-nfs-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/scc: restricted 1
  creationTimestamp: 2016-03-21T13:59:47Z
  labels:
    name: hello-openshift-nfs-pod
    name: hello-openshift-nfs-pod
    namespace: default 2
    resourceVersion: "2814411"
    selflink: /api/v1/namespaces/default/pods/hello-openshift-nfs-pod
    uid: 2c22d2ea-ef6d-11e5-adc7-000c2900f1e3
spec:
  containers:
    - image: fedora/S3
      imagePullPolicy: IfNotPresent

```

```

name: hello-openshift-nfs-pod
ports:
- containerPort: 80
  name: web
  protocol: TCP
resources: {}
securityContext:
  privileged: false
terminationMessagePath: /dev/termination-log
volumeMounts:
- mountPath: /usr/share/S3/html
  name: nfsvol
- mountPath: /var/run/secrets/kubernetes.io/serviceaccount
  name: default-token-a06zb
  readOnly: true
dnsPolicy: ClusterFirst
host: ose70.rh7
imagePullSecrets:
- name: default-dockercfg-xvdew
nodeName: ose70.rh7
restartPolicy: Always
securityContext:
  supplementalGroups:
  - 100003 ③
serviceAccount: default
serviceAccountName: default
terminationGracePeriodSeconds: 30
volumes:
- name: nfsvol
  persistentVolumeClaim:
    claimName: nfs-pvc ④
- name: default-token-a06zb
  secret:
    secretName: default-token-a06zb
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-03-21T13:59:49Z
    status: "True"
    type: Ready
  containerStatuses:
  - containerID:
    docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    image: fedora/S3
    imageID:
    docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    lastState: {}
    name: hello-openshift-nfs-pod
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-03-21T13:59:49Z
  hostIP: 192.168.234.148

```

```

phase: Running
podIP: 10.1.0.4
startTime: 2016-03-21T13:59:47Z

```

- 1 The SCC used by the pod.
- 2 The project (namespace) name.
- 3 The supplemental group ID for the pod (all containers).
- 4 The PVC name used by the pod.

25.2.6. Creating an Additional Pod to Reference the Same PVC

This pod definition, created in the same namespace, uses a different container. However, we can use the same backing storage by specifying the claim name in the volumes section below:

Example 25.4. Pod Object Definition

```

apiVersion: v1
kind: Pod
metadata:
  name: busybox-nfs-pod 1
  labels:
    name: busybox-nfs-pod
spec:
  containers:
  - name: busybox-nfs-pod
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: nfsvol-2 3
      mountPath: /usr/share/busybox 4
      readOnly: false
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
  - name: nfsvol-2
    persistentVolumeClaim:
      claimName: nfs-pvc 6

```

- 1 The name of this pod as displayed by `oc get pod`.
- 2 The image run by this pod.
- 3 The name of the volume. This name must be the same in both the **containers** and **volumes** sections.
- 4 The mount path as seen in the container.
- 5 The group ID to be assigned to the container.
- 6 The PVC that was created earlier and is also being used by a different container.

Save the pod definition to a file, for example **nfs-2.yaml**, and create the pod:

```
# oc create -f nfs-2.yaml
pod "busybox-nfs-pod" created
```

Verify that the pod was created:

```
# oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
busybox-nfs-pod	1/1	Running	0	3s

More details are shown in the **oc describe pod** command:

```
[root@ose70 nfs]# oc describe pod busybox-nfs-pod
Name:      busybox-nfs-pod
Namespace: default
Image(s):  busybox
Node:      ose70.rh7/192.168.234.148
Start Time: Mon, 21 Mar 2016 10:19:46 -0400
Labels:    name=busybox-nfs-pod
Status:    Running
Reason:
Message:
IP:        10.1.0.5
Replication Controllers: <none>
Containers:
  busybox-nfs-pod:
    Container ID:
docker://346d432e5a4824ebf5a47fceb4247e0568ecc64eadcc160e9bab481aecfb0594
    Image: busybox
    Image ID:
docker://17583c7dd0dae6244203b8029733bdb7d17fccbb2b5d93e2b24cf48b8bfd06e2
    QoS Tier:
      cpu: BestEffort
      memory: BestEffort
    State: Running
      Started: Mon, 21 Mar 2016 10:19:48 -0400
      Ready: True
      Restart Count: 0
    Environment Variables:
Conditions:
  Type Status
  Ready True
Volumes:
  nfsvol-2:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName: nfs-pvc
    ReadOnly: false
  default-token-32d2z:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-32d2z
Events:
```

FirstSeen	LastSeen	Count	From	SubobjectPath	Reason	Message
4m	4m	1	{scheduler }		Scheduled	Successfully assigned busybox-nfs-pod to ose70.rh7
4m	4m	1	{kubelet ose70.rh7}		implicitly required container	POD Pulled Container image "openshift3/ose-pod:v3.1.0.4" already present on machine
4m	4m	1	{kubelet ose70.rh7}		implicitly required container	POD Created Created with docker id 249b7d7519b1
4m	4m	1	{kubelet ose70.rh7}		implicitly required container	POD Started Started with docker id 249b7d7519b1
4m	4m	1	{kubelet ose70.rh7}	spec.containers{busybox-nfs-pod}	Pulled	Container image "busybox" already present on machine
4m	4m	1	{kubelet ose70.rh7}	spec.containers{busybox-nfs-pod}	Created	Created with docker id 346d432e5a48
4m	4m	1	{kubelet ose70.rh7}	spec.containers{busybox-nfs-pod}	Started	Started with docker id 346d432e5a48

As you can see, both containers are using the same storage claim that is attached to the same NFS mount on the back end.

25.3. COMPLETE EXAMPLE USING CEPH RBD

25.3.1. Overview

This topic provides an end-to-end example of using an existing Ceph cluster as an OpenShift Container Platform persistent store. It is assumed that a working Ceph cluster is already set up. If not, consult the [Overview of Red Hat Ceph Storage](#).

[Persistent Storage Using Ceph Rados Block Device](#) provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and using Ceph RBD as persistent storage.



NOTE

All **oc** ... commands are executed on the OpenShift Container Platform master host.

25.3.2. Installing the ceph-common Package

The **ceph-common** library must be installed on **all schedulable** OpenShift Container Platform nodes:



NOTE

The OpenShift Container Platform all-in-one host is not often used to run pod workloads and, thus, is not included as a schedulable node.

```
# yum install -y ceph-common
```

25.3.3. Creating the Ceph Secret

The **ceph auth get-key** command is run on a Ceph **MON** node to display the key value for the **client.admin** user:

Example 25.5. Ceph Secret Definition


```

apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFB0FF2S1ZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== ❶

```

- ❶ This base64 key is generated on one of the Ceph MON nodes using the **ceph auth get-key client.admin | base64** command, then copying the output and pasting it as the secret key's value.

Save the secret definition to a file, for example **ceph-secret.yaml**, then create the secret:

```

$ oc create -f ceph-secret.yaml
secret "ceph-secret" created

```

Verify that the secret was created:

```

# oc get secret ceph-secret
NAME          TYPE      DATA      AGE
ceph-secret   Opaque    1          23d

```

25.3.4. Creating the Persistent Volume

Next, before creating the PV object in OpenShift Container Platform, define the persistent volume file:

Example 25.6. Persistent Volume Object Definition Using Ceph RBD

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv ❶
spec:
  capacity:
    storage: 2Gi ❷
  accessModes:
    - ReadWriteOnce ❸
  rbd: ❹
    monitors: ❺
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret ❻
    fsType: ext4 ❼
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle

```

- 1 The name of the PV, which is referenced in pod definitions or displayed in various **oc** volume commands.
- 2 The amount of storage allocated to this volume.
- 3 **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control. All block storage is defined to be single user (non-shared storage).
- 4 This defines the volume type being used. In this case, the **rbd** plug-in is defined.
- 5 This is an array of Ceph monitor IP addresses and ports.
- 6 This is the Ceph secret, defined above. It is used to create a secure connection from OpenShift Container Platform to the Ceph server.
- 7 This is the file system type mounted on the Ceph RBD block device.

Save the PV definition to a file, for example **ceph-pv.yaml**, and create the persistent volume:

```
# oc create -f ceph-pv.yaml
persistentvolume "ceph-pv" created
```

Verify that the persistent volume was created:

```
# oc get pv
NAME                                LABELS            CAPACITY          ACCESSMODES      STATUS
CLAIM          REASON          AGE
ceph-pv        <none>          2147483648        RWO              Available
2s
```

25.3.5. Creating the Persistent Volume Claim

A persistent volume claim (PVC) specifies the desired access mode and storage capacity. Currently, based on only these two attributes, a PVC is bound to a single PV. Once a PV is bound to a PVC, that PV is essentially tied to the PVC's project and cannot be bound to by another PVC. There is a one-to-one mapping of PVs and PVCs. However, multiple pods in the same project can use the same PVC.

Example 25.7. PVC Object Definition

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

- 1 As mentioned above for PVs, the **accessModes** do not enforce access right, but rather act as labels to match a PV to a PVC.

- 2 This claim will look for PVs offering **2Gi** or greater capacity.

Save the PVC definition to a file, for example **ceph-claim.yaml**, and create the PVC:

```
# oc create -f ceph-claim.yaml
persistentvolumeclaim "ceph-claim" created

#and verify the PVC was created and bound to the expected PV:
# oc get pvc
NAME          LABELS      STATUS    VOLUME    CAPACITY    ACCESSMODES    AGE
ceph-claim    <none>      Bound     ceph-pv    1Gi         RWX            21s
```

- 1 the claim was bound to the **ceph-pv** PV.

25.3.6. Creating the Pod

A pod definition file or a template file can be used to define a pod. Below is a pod specification that creates a single container and mounts the Ceph RBD volume for read-write access:

Example 25.8. Pod Object Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1
spec:
  containers:
  - name: ceph-busybox
    image: busybox
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1
      mountPath: /usr/share/busybox
      readOnly: false
  volumes:
  - name: ceph-vol1
    persistentVolumeClaim:
      claimName: ceph-claim
```

- 1 The name of this pod as displayed by **oc get pod**.
- 2 The image run by this pod. In this case, we are telling **busybox** to sleep.
- 3 5 The name of the volume. This name must be the same in both the **containers** and **volumes** sections.
- 4 The mount path as seen in the container.
- 6 The PVC that is bound to the Ceph RBD cluster.

Save the pod definition to a file, for example ***ceph-pod1.yaml***, and create the pod:

```
# oc create -f ceph-pod1.yaml
pod "ceph-pod1" created

#verify pod was created
# oc get pod
NAME          READY    STATUS    RESTARTS   AGE
ceph-pod1     1/1      Running   0           2m
```

1

1 After a minute or so, the pod will be in the **Running** state.

25.3.7. Defining Group and Owner IDs (Optional)

When using block storage, such as Ceph RBD, the physical block storage is **managed** by the pod. The group ID defined in the pod becomes the group ID of **both** the Ceph RBD mount inside the container, and the group ID of the actual storage itself. Thus, it is usually unnecessary to define a group ID in the pod specification. However, if a group ID is desired, it can be defined using **fsGroup**, as shown in the following pod definition fragment:

Example 25.9. Group ID Pod Definition

```
...
spec:
  containers:
    - name:
      ...
  securityContext: 1
    fsGroup: 7777 2
  ...
```

1 **securityContext** must be defined at the pod level, not under a specific container.

2 All containers in the pod will have the same **fsGroup** ID.

25.3.8. Setting ceph-user-secret as Default for Projects

If you would like to make the persistent storage available to every project you have to modify the default project template. You can read more on modifying the default project template. Read more on [modifying the default project template](#). Adding this to your default project template allows every user who has access to create a project access to the Ceph cluster.

Example 25.10. Default Project Example

```
...
apiVersion: v1
kind: Template
metadata:
```

```

    creationTimestamp: null
    name: project-request
  objects:
  - apiVersion: v1
    kind: Project
    metadata:
      annotations:
        openshift.io/description: ${PROJECT_DESCRIPTION}
        openshift.io/display-name: ${PROJECT_DISPLAYNAME}
        openshift.io/requester: ${PROJECT_REQUESTING_USER}
      creationTimestamp: null
      name: ${PROJECT_NAME}
    spec: {}
    status: {}
  - apiVersion: v1
    kind: Secret
    metadata:
      name: ceph-user-secret
    data:
      key: yoursupersecretbase64keygoeshere 1
    type:
      kubernetes.io/rbd
  ...

```

1 Place your super secret Ceph user key here in base64 format. See [Creating the Ceph Secret](#).

25.4. USING CEPH RBD FOR DYNAMIC PROVISIONING

25.4.1. Overview

This topic provides a complete example of using an existing Ceph cluster for OpenShift Container Platform persistent storage. It is assumed that a working Ceph cluster is already set up. If not, consult the [Overview of Red Hat Ceph Storage](#).

[Persistent Storage Using Ceph Rados Block Device](#) provides an explanation of persistent volumes (PVs), persistent volume claims (PVCs), and how to use Ceph Rados Block Device (RBD) as persistent storage.



NOTE

- Run all **oc** commands on the OpenShift Container Platform master host.
- The OpenShift Container Platform all-in-one host is not often used to run pod workloads and, thus, is not included as a schedulable node.

25.4.2. Creating a pool for dynamic volumes

1. Install the latest ceph-common package:

```
yum install -y ceph-common
```

**NOTE**

The **ceph-common** library must be installed on **all schedulable** OpenShift Container Platform nodes.

- From an administrator or **MON** node, create a new pool for dynamic volumes, for example:

```
$ ceph osd pool create kube 1024
$ ceph auth get-or-create client.kube mon 'allow r, allow command
"osd blacklist"' osd 'allow class-read object_prefix rbd_children,
allow rwx pool=kube' -o ceph.client.kube.keyring
```

**NOTE**

Using the default pool of RBD is an option, but not recommended.

25.4.3. Using an existing Ceph cluster for dynamic persistent storage

To use an existing Ceph cluster for dynamic persistent storage:

- Generate the client.admin base64-encoded key:

```
$ ceph auth get client.admin
```

Ceph secret definition example

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
  namespace: kube-system
data:
  key: QVFB0FF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
type: kubernetes.io/rbd 2
```

1 This base64 key is generated on one of the Ceph MON nodes using the **ceph auth get-key client.admin | base64** command, then copying the output and pasting it as the secret key's value.

2 This value is required for Ceph RBD to work with dynamic provisioning.

- Create the Ceph secret for the client.admin:

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

- Verify that the secret was created:

```
$ oc get secret ceph-secret
NAME          TYPE                      DATA   AGE
ceph-secret   kubernetes.io/rbd         1       5d
```

4. Create the storage class:

```
$ oc create -f ceph-storageclass.yaml
storageclass "dynamic" created
```

Ceph storage class example

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: dynamic
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.1.11:6789,192.168.1.12:6789,192.168.1.13:6789 1
  adminId: admin 2
  adminSecretName: ceph-secret 3
  adminSecretNamespace: kube-system 4
  pool: kube 5
  userId: kube 6
  userSecretName: ceph-user-secret 7
```

- 1 A comma-delimited list of IP addresses Ceph monitors. This value is required.
- 2 The Ceph client ID that is capable of creating images in the pool. The default is **admin**.
- 3 The secret name for **adminId**. This value is required. The secret that you provide must have **kubernetes.io/rbd**.
- 4 The namespace for **adminSecret**. The default is **default**.
- 5 The Ceph RBD pool. The default is **rbd**, but this value is not recommended.
- 6 The Ceph client ID used to map the Ceph RBD image. The default is the same as the secret name for **adminId**.
- 7 The name of the Ceph secret for **userId** to map the Ceph RBD image. It must exist in the same namespace as the PVCs. Unless you set the Ceph secret as the default in new projects, you must provide this parameter value.

5. Verify that the storage class was created:

```
$ oc get storageclasses
NAME                                TYPE
dynamic (default)                  kubernetes.io/rbd
```

6. Create the PVC object definition:

PVC object definition example

```
kind: PersistentVolumeClaim
apiVersion: v1
```

```

metadata:
  name: ceph-claim-dynamic
spec:
  accessModes: ❶
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi ❷

```

- ❶ The **accessModes** do not enforce access rights but instead act as labels to match a PV to a PVC.
- ❷ This claim looks for PVs that offer **2Gi** or greater capacity.

7. Create the PVC:

```

$ oc create -f ceph-pvc.yaml
persistentvolumeclaim "ceph-claim-dynamic" created

```

8. Verify that the PVC was created and bound to the expected PV:

```

$ oc get pvc
NAME          STATUS  VOLUME
CAPACITY ACCESSMODES  AGE
ceph-claim    Bound   pvc-f548d663-3cac-11e7-9937-0024e8650c7a 2Gi
RW0           1m

```

9. Create the pod object definition:

Pod object definition example

```

apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1 ❶
spec:
  containers:
    - name: ceph-busybox
      image: busybox ❷
      command: ["sleep", "60000"]
      volumeMounts:
        - name: ceph-vol1 ❸
          mountPath: /usr/share/busybox ❹
          readOnly: false
  volumes:
    - name: ceph-vol1
      persistentVolumeClaim:
        claimName: ceph-claim ❺

```

- ❶ The name of this pod as displayed by **oc get pod**.
- ❷ The image run by this pod. In this case, **busybox** is set to **sleep**.

- 3 The name of the volume. This name must be the same in both the **containers** and **volumes** sections.
- 4 The mount path in the container.
- 5 The PVC that is bound to the Ceph RBD cluster.

10. Create the pod:

```
$ oc create -f ceph-pod1.yaml
pod "ceph-pod1" created
```

11. Verify that the pod was created:

```
$ oc get pod
NAME          READY   STATUS    RESTARTS   AGE
ceph-pod1     1/1     Running   0           2m
```

After a minute or so, the pod status changes to **Running**.

25.4.4. Setting ceph-user-secret as the default for projects

To make persistent storage available to every project, you must modify the default project template. Adding this to your default project template allows every user who has access to create a project access to the Ceph cluster. See [modifying the default project template](#) for more information.

Default project example

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
```

```
key: QVFCbEV40VpmaGJtQ0JBQW55d2Z0NHZtcS96cE42SW1JVUQvekE9PQ== 1
type:
  kubernetes.io/rbd
...
```

- 1 Place your Ceph user key here in base64 format.

25.5. COMPLETE EXAMPLE USING GLUSTERFS

25.5.1. Overview

This topic provides an end-to-end example of how to use an existing converged mode, independent mode, or standalone Red Hat Gluster Storage cluster as persistent storage for OpenShift Container Platform. It is assumed that a working Red Hat Gluster Storage cluster is already set up. For help installing converged mode or independent mode, see [Persistent Storage Using Red Hat Gluster Storage](#). For standalone Red Hat Gluster Storage, consult the [Red Hat Gluster Storage Administration Guide](#).

For an end-to-end example of how to dynamically provision GlusterFS volumes, see [Complete Example Using GlusterFS for Dynamic Provisioning](#).



NOTE

All **oc** commands are executed on the OpenShift Container Platform master host.

25.5.2. Prerequisites

To access GlusterFS volumes, the **mount.glusterfs** command must be available on all schedulable nodes. For RPM-based systems, the **glusterfs-fuse** package must be installed:

```
# yum install glusterfs-fuse
```

This package comes installed on every RHEL system. However, it is recommended to update to the latest available version from Red Hat Gluster Storage if your servers use x86_64 architecture. To do this, the following RPM repository must be enabled:

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-
server-rpms
```

If **glusterfs-fuse** is already installed on the nodes, ensure that the latest version is installed:

```
# yum update glusterfs-fuse
```

By default, SELinux does not allow writing from a pod to a remote Red Hat Gluster Storage server. To enable writing to Red Hat Gluster Storage volumes with SELinux on, run the following on each node running GlusterFS:

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

- 1 The **-P** option makes the boolean persistent between reboots.

**NOTE**

The `virt_sandbox_use_fusefs` boolean is defined by the `docker-selinux` package. If you get an error saying it is not defined, ensure that this package is installed.

**NOTE**

If you use Atomic Host, the SELinux booleans are cleared when you upgrade Atomic Host. When you upgrade Atomic Host, you must set these boolean values again.

25.5.3. Static Provisioning

1. To enable static provisioning, first create a GlusterFS volume. See the [Red Hat Gluster Storage Administration Guide](#) for information on how to do this using the `gluster` command-line interface or the [heketi project site](#) for information on how to do this using `heketi-cli`. For this example, the volume will be named `myVol1`.
2. Define the following Service and Endpoints in `gluster-endpoints.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster 1
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster 2
subsets:
- addresses:
  - ip: 192.168.122.221 3
  ports:
  - port: 1 4
- addresses:
  - ip: 192.168.122.222 5
  ports:
  - port: 1 6
- addresses:
  - ip: 192.168.122.223 7
  ports:
  - port: 1 8
```

1 2 These names must match.

3 5 7 The `ip` values must be the actual IP addresses of a Red Hat Gluster Storage server, not hostnames.

4 6 8 The port number is ignored.

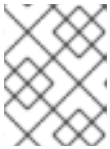
3. From the OpenShift Container Platform master host, create the Service and Endpoints:

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. Verify that the Service and Endpoints were created:

```
$ oc get services
NAME                                CLUSTER_IP          EXTERNAL_IP    PORT(S)
SELECTOR          AGE
glusterfs-cluster 172.30.205.34        <none>         1/TCP
<none>            44s

$ oc get endpoints
NAME                                ENDPOINTS
AGE
docker-registry 10.1.0.3:5000
4h
glusterfs-cluster
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1 11s
kubernetes      172.16.35.3:8443
4d
```



NOTE

Endpoints are unique per project. Each project accessing the GlusterFS volume needs its own Endpoints.

5. In order to access the volume, the container must run with either a user ID (UID) or group ID (GID) that has access to the file system on the volume. This information can be discovered in the following manner:

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0 myVol1 1 2
```

1 The UID is 592.

2 The GID is 590.

6. Define the following PersistentVolume (PV) in **gluster-pv.yaml**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
  annotations:
    pv.beta.kubernetes.io/gid: "590" 2
spec:
```

```

capacity:
  storage: 2Gi 3
accessModes: 4
- ReadWriteMany
glusterfs:
  endpoints: glusterfs-cluster 5
  path: myVol1 6
  readOnly: false
persistentVolumeReclaimPolicy: Retain

```

- 1 The name of the volume.
- 2 The GID on the root of the GlusterFS volume.
- 3 The amount of storage allocated to this volume.
- 4 **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control.
- 5 The Endpoints resource previously created.
- 6 The GlusterFS volume that will be accessed.

7. From the OpenShift Container Platform master host, create the PV:

```
$ oc create -f gluster-pv.yaml
```

8. Verify that the PV was created:

```

$ oc get pv
NAME                                LABELS      CAPACITY      ACCESSMODES
STATUS      CLAIM      REASON      AGE
gluster-default-volume  <none>      2147483648    RWX
Available                                         2s

```

9. Create a PersistentVolumeClaim (PVC) that will bind to the new PV in **gluster-claim.yaml**:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim 1
spec:
  accessModes:
  - ReadWriteMany 2
  resources:
    requests:
      storage: 1Gi 3

```

- 1 The claim name is referenced by the pod under its **volumes** section.
- 2 Must match the **accessModes** of the PV.
- 3 This claim will look for PVs offering **1Gi** or greater capacity.

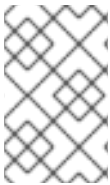
- From the OpenShift Container Platform master host, create the PVC:

```
$ oc create -f gluster-claim.yaml
```

- Verify that the PV and PVC are bound:

```
$ oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS      CLAIM
REASON      AGE
gluster-pv    <none>          1Gi       RWX           Available
gluster-claim 37s

$ oc get pvc
NAME          LABELS          STATUS    VOLUME      CAPACITY
ACCESSMODES  AGE
gluster-claim <none>          Bound     gluster-pv   1Gi       RWX
24s
```



NOTE

PVCs are unique per project. Each project accessing the GlusterFS volume needs its own PVC. PVs are not bound to a single project, so PVCs across multiple projects may refer to the same PV.

25.5.4. Using the Storage

At this point, you have a dynamically created GlusterFS volume bound to a PVC. You can now utilize this PVC in a pod.

- Create the pod object definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 1
```

- The name of the PVC created in the previous step.

- From the OpenShift Container Platform master host, create the pod:

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

- View the pod. Give it a few minutes, as it might need to download the image if it does not already exist:

```
# oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS
AGE	IP	NODE	
hello-openshift-pod		1/1	Running 0
9m	10.38.0.0	node1	

- oc exec** into the container and create an *index.html* file in the **mountPath** definition of the pod:

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

- Now **curl** the URL of the pod:

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

- Delete the pod, recreate it, and wait for it to come up:

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS
AGE	IP	NODE	
hello-openshift-pod		1/1	Running 0
9m	10.37.0.0	node1	

- Now **curl** the pod again and it should still have the same data as before. Note that its IP address may have changed:

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

- Check that the *index.html* file was written to GlusterFS storage by doing the following on any of the nodes:

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
```

```

/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e7
30a5462c352835055018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!

```

25.6. COMPLETE EXAMPLE USING GLUSTERFS FOR DYNAMIC PROVISIONING

25.6.1. Overview

This topic provides an end-to-end example of how to use an existing converged mode, independent mode, or standalone Red Hat Gluster Storage cluster as dynamic persistent storage for OpenShift Container Platform. It is assumed that a working Red Hat Gluster Storage cluster is already set up. For help installing converged mode or independent mode, see [Persistent Storage Using Red Hat Gluster Storage](#). For standalone Red Hat Gluster Storage, consult the [Red Hat Gluster Storage Administration Guide](#).



NOTE

All **oc** commands are executed on the OpenShift Container Platform master host.

25.6.2. Prerequisites

To access GlusterFS volumes, the **mount.glusterfs** command must be available on all schedulable nodes. For RPM-based systems, the **glusterfs-fuse** package must be installed:

```
# yum install glusterfs-fuse
```

This package comes installed on every RHEL system. However, it is recommended to update to the latest available version from Red Hat Gluster Storage if your servers use x86_64 architecture. To do this, the following RPM repository must be enabled:

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-
server-rpms
```

If **glusterfs-fuse** is already installed on the nodes, ensure that the latest version is installed:

■


```
# yum update glusterfs-fuse
```

By default, SELinux does not allow writing from a pod to a remote Red Hat Gluster Storage server. To enable writing to Red Hat Gluster Storage volumes with SELinux on, run the following on each node running GlusterFS:

```
$ sudo setsebool -P virt_sandbox_use_fusefs on ❶
$ sudo setsebool -P virt_use_fusefs on
```

❶ The **-P** option makes the boolean persistent between reboots.



NOTE

The **virt_sandbox_use_fusefs** boolean is defined by the **docker-selinux** package. If you get an error saying it is not defined, ensure that this package is installed.



NOTE

If you use Atomic Host, the SELinux booleans are cleared when you upgrade Atomic Host. When you upgrade Atomic Host, you must set these boolean values again.

25.6.3. Dynamic Provisioning

1. To enable dynamic provisioning, first create a **StorageClass** object definition. The definition below is based on the minimum requirements needed for this example to work with OpenShift Container Platform. See [Dynamic Provisioning and Creating Storage Classes](#) for additional parameters and specification definitions.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" ❶
  restauthenabled: "false" ❷
```

❶ The heketi server URL.

❷ Since authentication is not turned on in this example, set to **false**.

2. From the OpenShift Container Platform master host, create the StorageClass:

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. Create a PVC using the newly-created StorageClass. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```

name: gluster1
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 30Gi
  storageClassName: glusterfs

```

4. From the OpenShift Container Platform master host, create the PVC:

```

# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created

```

5. View the PVC to see that the volume was dynamically created and bound to the PVC:

```

# oc get pvc
NAME                                STATUS VOLUME
CAPACITY  ACCESSMODES  STORAGECLASS  AGE
gluster1  Bound       pvc-78852230-d8e2-11e6-a3fa-0800279cf26f
30Gi      RWX          gluster-dyn  42s

```

25.6.4. Using the Storage

At this point, you have a dynamically created GlusterFS volume bound to a PVC. You can now utilize this PVC in a pod.

1. Create the pod object definition:

```

apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
    - name: hello-openshift-pod
      image: openshift/hello-openshift
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: gluster-vol1
          mountPath: /usr/share/nginx/html
          readOnly: false
  volumes:
    - name: gluster-vol1
      persistentVolumeClaim:
        claimName: gluster1 1

```

- 1 The name of the PVC created in the previous step.

- From the OpenShift Container Platform master host, create the pod:

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

- View the pod. Give it a few minutes, as it might need to download the image if it does not already exist:

```
# oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS
AGE	IP	NODE	
hello-openshift-pod		1/1	Running 0
9m	10.38.0.0	node1	

- oc exec** into the container and create an *index.html* file in the **mountPath** definition of the pod:

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

- Now **curl** the URL of the pod:

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

- Delete the pod, recreate it, and wait for it to come up:

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS
AGE	IP	NODE	
hello-openshift-pod		1/1	Running 0
9m	10.37.0.0	node1	

- Now **curl** the pod again and it should still have the same data as before. Note that its IP address may have changed:

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

- Check that the *index.html* file was written to GlusterFS storage by doing the following on any of the nodes:

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
```

```

/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e7
30a5462c352835055018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!

```

25.7. MOUNTING VOLUMES ON PRIVILEGED PODS

25.7.1. Overview

Persistent volumes can be mounted to pods with the **privileged** security context constraint (SCC) attached.



NOTE

While this topic uses GlusterFS as a sample use-case for mounting volumes onto privileged pods, it can be adapted to use any [supported storage plug-in](#).

25.7.2. Prerequisites

- An existing Gluster volume.
- **glusterfs-fuse** installed on all hosts.
- Definitions for GlusterFS:
 - [Endpoints and services](#): **gluster-endpoints-service.yaml** and **gluster-endpoints.yaml**
 - [Persistent volumes](#): **gluster-pv.yaml**
 - [Persistent volume claims](#): **gluster-pvc.yaml**
 - [Privileged pods](#): **gluster-S3-pod.yaml**
- A user with the [cluster-admin](#) role binding. For this guide, that user is called **admin**.

25.7.3. Creating the Persistent Volume

Creating the **PersistentVolume** makes the storage accessible to users, regardless of projects.

1. As the admin, create the service, endpoint object, and persistent volume:

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
```

2. Verify that the objects were created:

```
$ oc get svc
NAME                                CLUSTER_IP          EXTERNAL_IP    PORT(S)    SELECTOR
AGE
gluster-cluster                    172.30.151.58       <none>         1/TCP      <none>
24s
```

```
$ oc get ep
NAME                                ENDPOINTS                                AGE
gluster-cluster                    192.168.59.102:1,192.168.59.103:1      2m
```

```
$ oc get pv
NAME                                LABELS    CAPACITY    ACCESSMODES    STATUS
CLAIM    REASON    AGE
gluster-default-volume              <none>    2Gi         RWX
Available                                2d
```

25.7.4. Creating a Regular User

Adding a [regular user](#) to the **privileged** SCC (or to a group given access to the SCC) allows them to run **privileged** pods:

1. As the admin, add a user to the SCC:

```
$ oc adm policy add-scc-to-user privileged <username>
```

2. Log in as the regular user:

```
$ oc login -u <username> -p <password>
```

3. Then, create a new project:

```
$ oc new-project <project_name>
```

25.7.5. Creating the Persistent Volume Claim

1. As a regular user, create the **PersistentVolumeClaim** to access the volume:

```
$ oc create -f gluster-pvc.yaml -n <project_name>
```

2. Define your pod to access the claim:

Example 25.11. Pod Definition

```
apiVersion: v1
```

```

id: gluster-S3-pvc
kind: Pod
metadata:
  name: gluster-nginx-priv
spec:
  containers:
    - name: gluster-nginx-priv
      image: fedora/nginx
      volumeMounts:
        - mountPath: /mnt/gluster ❶
          name: gluster-volume-claim
      securityContext:
        privileged: true
  volumes:
    - name: gluster-volume-claim
      persistentVolumeClaim:
        claimName: gluster-claim ❷

```

❶ Volume mount within the pod.

❷ The **gluster-claim** must reflect the name of the **PersistentVolume**.

3. Upon pod creation, the mount directory is created and the volume is attached to that mount point.

As regular user, create a pod from the definition:

```
$ oc create -f gluster-S3-pod.yaml
```

4. Verify that the pod created successfully:

```

$ oc get pods
NAME                READY   STATUS    RESTARTS   AGE
gluster-S3-pod      1/1     Running   0           36m

```

It can take several minutes for the pod to create.

25.7.6. Verifying the Setup

25.7.6.1. Checking the Pod SCC

1. Export the pod configuration:

```
$ oc export pod <pod_name>
```

2. Examine the output. Check that **openshift.io/scc** has the value of **privileged**:

Example 25.12. Export Snippet

```

metadata:
  annotations:
    openshift.io/scc: privileged

```

25.7.6.2. Verifying the Mount

1. Access the pod and check that the volume is mounted:

```
$ oc rsh <pod_name>
[root@gluster-S3-pvc /]# mount
```

2. Examine the output for the Gluster volume:

Example 25.13. Volume Mount

```
192.168.59.102:gv0 on /mnt/gluster type fuse.gluster
(rw,relatime,user_id=0,group_id=0,default_permissions,allow_other,
max_read=131072)
```

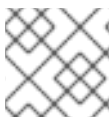
25.8. SWITCHING AN INTEGRATED OPENSIFT CONTAINER REGISTRY TO GLUSTERFS

25.8.1. Overview

This topic reviews how to attach a GlusterFS volume to an integrated OpenShift Container Registry. This can be done with any of converged mode, independent mode, or standalone Red Hat Gluster Storage. It is assumed that the registry has already been started and a volume has been created.

25.8.2. Prerequisites

- An existing [registry](#) deployed **without** configuring storage.
- An existing GlusterFS volume
- **glusterfs-fuse** installed on all schedulable nodes.
- A user with the [cluster-admin](#) role binding.
 - For this guide, that user is **admin**.



NOTE

All **oc** commands are executed on the master node as the **admin** user.

25.8.3. Manually Provision the GlusterFS PersistentVolumeClaim

1. To enable static provisioning, first create a GlusterFS volume. See the [Red Hat Gluster Storage Administration Guide](#) for information on how to do this using the **gluster** command-line interface or the [heketi project site](#) for information on how to do this using **heketi-cli**. For this example, the volume will be named **myVol1**.
2. Define the following Service and Endpoints in **gluster-endpoints.yaml**:

```
---
apiVersion: v1
```

```

kind: Service
metadata:
  name: glusterfs-cluster ❶
spec:
  ports:
  - port: 1
  ---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster ❷
subsets:
  - addresses:
    - ip: 192.168.122.221 ❸
    ports:
    - port: 1 ❹
  - addresses:
    - ip: 192.168.122.222 ❺
    ports:
    - port: 1 ❻
  - addresses:
    - ip: 192.168.122.223 ❼
    ports:
    - port: 1 ❽

```

❶ ❷ These names must match.

❸ ❺ ❼ The **ip** values must be the actual IP addresses of a Red Hat Gluster Storage server, not hostnames.

❹ ❻ ❽ The port number is ignored.

3. From the OpenShift Container Platform master host, create the Service and Endpoints:

```

$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created

```

4. Verify that the Service and Endpoints were created:

```

$ oc get services
NAME                                CLUSTER_IP          EXTERNAL_IP  PORT(S)
SELECTOR      AGE
glusterfs-cluster 172.30.205.34      <none>       1/TCP
<none>         44s

$ oc get endpoints
NAME                                ENDPOINTS
AGE
docker-registry 10.1.0.3:5000
4h
glusterfs-cluster

```



```
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1    11s
kubernetes      172.16.35.3:8443
4d
```

**NOTE**

Endpoints are unique per project. Each project accessing the GlusterFS volume needs its own Endpoints.

5. In order to access the volume, the container must run with either a user ID (UID) or group ID (GID) that has access to the file system on the volume. This information can be discovered in the following manner:

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 1 2
```

1 The UID is 592.

2 The GID is 590.

6. Define the following PersistentVolume (PV) in **gluster-pv.yaml**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
  annotations:
    pv.beta.kubernetes.io/gid: "590" 2
spec:
  capacity:
    storage: 2Gi 3
  accessModes: 4
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster 5
    path: myVol1 6
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1 The name of the volume.

2 The GID on the root of the GlusterFS volume.

3 The amount of storage allocated to this volume.

4 **accessModes** are used as labels to match a PV and a PVC. They currently do not define any form of access control.

5 The Endpoints resource previously created.

- 6 The GlusterFS volume that will be accessed.

7. From the OpenShift Container Platform master host, create the PV:

```
$ oc create -f gluster-pv.yaml
```

8. Verify that the PV was created:

```
$ oc get pv
NAME                                LABELS            CAPACITY            ACCESSMODES
STATUS          CLAIM          REASON          AGE
gluster-default-volume  <none>          2147483648        RWX
Available                                             2s
```

9. Create a PersistentVolumeClaim (PVC) that will bind to the new PV in **gluster-claim.yaml**:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim 1
spec:
  accessModes:
    - ReadWriteMany 2
  resources:
    requests:
      storage: 1Gi 3
```

- 1 The claim name is referenced by the pod under its **volumes** section.
- 2 Must match the **accessModes** of the PV.
- 3 This claim will look for PVs offering **1Gi** or greater capacity.

10. From the OpenShift Container Platform master host, create the PVC:

```
$ oc create -f gluster-claim.yaml
```

11. Verify that the PV and PVC are bound:

```
$ oc get pv
NAME                                LABELS            CAPACITY            ACCESSMODES    STATUS          CLAIM
REASON          AGE
gluster-pv      <none>          1Gi              RWX            Available
gluster-claim   37s

$ oc get pvc
NAME                                LABELS            STATUS          VOLUME          CAPACITY
ACCESSMODES    AGE
gluster-claim  <none>          Bound          gluster-pv      1Gi              RWX
24s
```

**NOTE**

PVCs are unique per project. Each project accessing the GlusterFS volume needs its own PVC. PVs are not bound to a single project, so PVCs across multiple projects may refer to the same PV.

25.8.4. Attach the PersistentVolumeClaim to the Registry

Before moving forward, ensure that the **docker-registry** service is running.

```
$ oc get svc
NAME                                CLUSTER_IP          EXTERNAL_IP    PORT(S)
SELECTOR                            AGE
docker-registry    172.30.167.194      <none>         5000/TCP
docker-registry=default  18m
```

**NOTE**

If either the **docker-registry** service or its associated pod is not running, refer back to the [registry](#) setup instructions for troubleshooting before continuing.

Then, attach the PVC:

```
$ oc volume deploymentconfigs/docker-registry --add --name=registry-
storage -t pvc \
    --claim-name=gluster-claim --overwrite
```

[Setting up the Registry](#) provides more information on using an OpenShift Container Registry.

25.9. BINDING PERSISTENT VOLUMES BY LABELS**25.9.1. Overview**

This topic provides an end-to-end example for binding [persistent volume claims \(PVCs\)](#) to [persistent volumes \(PVs\)](#), by defining labels in the PV and matching selectors in the PVC. This feature is available for all [storage options](#). It is assumed that a OpenShift Container Platform cluster contains persistent storage resources which are available for binding by PVCs.

A Note on Labels and Selectors

Labels are an OpenShift Container Platform feature that support user-defined tags (key-value pairs) as part of an object's specification. Their primary purpose is to enable the arbitrary grouping of objects by defining identical labels among them. These labels can then be targeted by selectors to match all objects with specified label values. It is this functionality we will take advantage of to enable our PVC to bind to our PV. For a more in-depth look at labels, see [Pods and Services](#).

**NOTE**

For this example, we will be using modified [GlusterFS](#) PV and PVC specifications. However, implementation of selectors and labels is generic across for all storage options. See the [relevant storage option](#) for your volume provider to learn more about its unique configuration.

25.9.1.1. Assumptions

It is assumed that you have:

- An existing OpenShift Container Platform cluster with at least one **master** and one **node**
- At least one supported [storage volume](#)
- A user with **cluster-admin** privileges

25.9.2. Defining Specifications



NOTE

These specifications are tailored to **GlusterFS**. Consult the [relevant storage option](#) for your volume provider to learn more about its unique configuration.

25.9.2.1. Persistent Volume with Labels

Example 25.14. glusterfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: ❶
    storage-tier: gold
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster ❷
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

❶ Use labels to identify common attributes or characteristics shared among volumes. In this case, we defined the Gluster volume to have a custom attribute (key) named **storage-tier** with a value of **gold** assigned. A claim will be able to select a PV with **storage-tier=gold** to match this PV.

❷ Endpoints define the Gluster trusted pool and are discussed below.

25.9.2.2. Persistent Volume Claim with Selectors

A claim with a **selector** stanza (see example below) attempts to match existing, unclaimed, and non-prebound PVs. The existence of a PVC selector ignores a PV's capacity. However, **accessModes** are still considered in the matching criteria.

It is important to note that a claim must match **all** of the key-value pairs included in its **selector** stanza. If no PV matches the claim, then the PVC will remain unbound (Pending). A PV can subsequently be created and the claim will automatically check for a label match.

Example 25.15. glusterfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: ❶
    matchLabels:
      storage-tier: gold
      aws-availability-zone: us-east-1
```

❶ The **selector** stanza defines all labels necessary in a PV in order to match this claim.

25.9.2.3. Volume Endpoints

To attach the PV to the Gluster volume, endpoints should be configured before creating our objects.

Example 25.16. glusterfs-ep.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
    - ip: 192.168.122.221
    ports:
      - port: 1
  - addresses:
    - ip: 192.168.122.222
    ports:
      - port: 1
```

25.9.2.4. Deploy the PV, PVC, and Endpoints

For this example, run the **oc** commands as a **cluster-admin** privileged user. In a production environment, cluster clients might be expected to define and create the PVC.

```
# oc create -f glusterfs-ep.yaml
endpoints "glusterfs-cluster" created
```

```
# oc create -f glusterfs-pv.yaml
persistentvolume "gluster-volume" created
# oc create -f glusterfs-pvc.yaml
persistentvolumeclaim "gluster-claim" created
```

Lastly, confirm that the PV and PVC bound successfully.

```
# oc get pv,pvc
NAME                                CAPACITY  ACCESSMODES  STATUS  CLAIM
REASON  AGE
gluster-volume  2Gi       RWX          Bound   gfs-
trial/gluster-claim              7s
NAME                                STATUS  VOLUME  CAPACITY  ACCESSMODES
AGE
gluster-claim  Bound   gluster-volume  2Gi       RWX
7s
```



NOTE

PVCs are local to a project, whereas PVs are a cluster-wide, global resource. Developers and non-administrator users may not have access to see all (or any) of the available PVs.

25.10. USING STORAGE CLASSES FOR DYNAMIC PROVISIONING

25.10.1. Overview

In these examples we will walk through a few scenarios of various configurations of [StorageClasses](#) and Dynamic Provisioning using Google Cloud Platform Compute Engine (GCE). These examples assume some familiarity with Kubernetes, GCE and Persistent Disks and OpenShift Container Platform is installed and [properly configured to use GCE](#).

- [Basic Dynamic Provisioning](#)
- [Defaulting Cluster Dynamic Provisioning Behavior](#)

25.10.2. Scenario 1: Basic Dynamic Provisioning with Two Types of *StorageClasses*

StorageClasses can be used to differentiate and delineate storage levels and usages. In this case, the **cluster-admin** or **storage-admin** sets up two distinct classes of storage in GCE.

- **slow**: Cheap, efficient, and optimized for sequential data operations (slower reading and writing)
- **fast**: Optimized for higher rates of random IOPS and sustained throughput (faster reading and writing)

By creating these *StorageClasses*, the **cluster-admin** or **storage-admin** allows users to create claims requesting a particular level or service of *StorageClass*.

Example 25.17. StorageClass Slow Object Definitions

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```

metadata:
  name: slow ❶
provisioner: kubernetes.io/gce-pd ❷
parameters:
  type: pd-standard ❸
  zone: us-east1-d ❹

```

- ❶ Name of the *StorageClass*.
- ❷ The provisioner plug-in to be used. This is a required field for *StorageClasses*.
- ❸ PD type. This example uses **pd-standard**, which has a slightly lower cost, rate of sustained IOPS, and throughput versus **pd-ssd**, which carries more sustained IOPS and throughput.
- ❹ The zone is required.

Example 25.18. StorageClass Fast Object Definition

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-east1-d

```

As a **cluster-admin** or **storage-admin**, save both definitions as YAML files. For example, **slow-gce.yaml** and **fast-gce.yaml**. Then create the *StorageClasses*.

```

# oc create -f slow-gce.yaml
storageclass "slow" created

# oc create -f fast-gce.yaml
storageclass "fast" created

# oc get storageclass
NAME          TYPE
fast          kubernetes.io/gce-pd
slow          kubernetes.io/gce-pd

```



IMPORTANT

cluster-admin or **storage-admin** users are responsible for relaying the correct *StorageClass* name to the correct users, groups, and projects.

As a regular user, create a new project:

```
# oc new-project rh-eng
```

Create the claim YAML definition, save it to a file (**pvc-fast.yaml**):

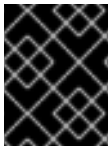
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-engineering
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: fast
```

Add the claim with the **oc create** command:

```
# oc create -f pvc-fast.yaml
persistentvolumeclaim "pvc-engineering" created
```

Check to see if your claim is bound:

```
# oc get pvc
NAME                                STATUS    VOLUME
CAPACITY  ACCESSMODES  AGE
pvc-engineering  Bound        pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi         RWX          2m
```

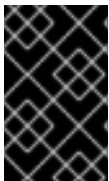


IMPORTANT

Since this claim was created and bound in the *rh-eng* project, it can be shared by any user in the same project.

As a **cluster-admin** or **storage-admin** user, view the recent dynamically provisioned Persistent Volume (PV).

```
# oc get pv
NAME                                CAPACITY  ACCESSMODES
RECLAIMPOLICY  STATUS    CLAIM                                REASON    AGE
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi      RWX
Delete        Bound        rh-eng/pvc-engineering              5m
```



IMPORTANT

Notice the **RECLAIMPOLICY** is *Delete* by default for all dynamically provisioned volumes. This means the volume only lasts as long as the claim still exists in the system. If you delete the claim, the volume is also deleted and all data on the volume is lost.

Finally, check the GCE console. The new disk has been created and is ready for use.

```
kubernetes-dynamic-pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  SSD
persistent disk  10 GB  us-east1-d
```


Pods can now reference the persistent volume claim and start using the volume.

25.10.3. Scenario 2: How to enable Default *StorageClass* behavior for a Cluster

In this example, a **cluster-admin** or **storage-admin** enables a *default* storage class for all other users and projects that do not implicitly specify a *StorageClass* in their claim. This is useful for a **cluster-admin** or **storage-admin** to provide easy management of a storage volume without having to set up or communicate specialized *StorageClasses* across the cluster.

This example builds upon [Section 25.10.2, “Scenario 1: Basic Dynamic Provisioning with Two Types of *StorageClasses*”](#). The **cluster-admin** or **storage-admin** will create another *StorageClass* for designation as the *defaultStorageClass*.

Example 25.19. Default *StorageClass* Object Definition

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: generic 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" 2
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zone: us-east1-d
```

- 1** Name of the *StorageClass*, which needs to be unique in the cluster.
- 2** Annotation that marks this *StorageClass* as the default class. You must use **"true"** quoted in this version of the API. Without this annotation, OpenShift Container Platform considers this not the *default StorageClass*.

As a **cluster-admin** or **storage-admin** save the definition to a YAML file (**generic-gce.yaml**), then create the *StorageClasses*:

```
# oc create -f generic-gce.yaml
storageclass "generic" created

# oc get storageclass
NAME          TYPE
generic       kubernetes.io/gce-pd
fast          kubernetes.io/gce-pd
slow          kubernetes.io/gce-pd
```

As a regular user, create a new claim definition without any *StorageClass* requirement and save it to a file (**generic-pvc.yaml**).

Example 25.20. *default* Storage Claim Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```

name: pvc-engineering2
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi

```

Execute it and check the claim is bound:

```

# oc create -f generic-pvc.yaml
persistentvolumeclaim "pvc-engineering2" created

# oc get pvc
NAME                                STATUS    VOLUME
CAPACITY  ACCESSMODES  AGE
pvc-engineering    Bound       pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi            RWX         41m
pvc-engineering2    Bound       pvc-a9f70544-8bfd-11e6-9962-42010af00004
5Gi            RWX         7s 1

```

1 **pvc-engineering2** is bound to a dynamically provisioned Volume by *default*.

As a **cluster-admin** or **storage-admin**, view the Persistent Volumes defined so far:

```

# oc get pv
NAME                                CAPACITY  ACCESSMODES
RECLAIMPOLICY  STATUS    CLAIM
pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi      RWX
Delete         Bound     rh-eng/pvc-engineering2
pvc-ba4612ce-8b4d-11e6-9962-42010af00004  5Gi      RWX
Delete         Bound     mytest/gce-dyn-claim1
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi     RWX
Delete         Bound     rh-eng/pvc-engineering

```

1 This PV was bound to our *default* dynamic volume from the *default StorageClass*.

2 This PV was bound to our first PVC from [Section 25.10.2, “Scenario 1: Basic Dynamic Provisioning with Two Types of StorageClasses”](#) with our *fast StorageClass*.

Create a manually provisioned disk using [GCE](#) (not dynamically provisioned). Then create a [Persistent Volume](#) that connects to the new GCE disk (**pv-manual-gce.yaml**).

Example 25.21. Manual PV Object Definition

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-manual-gce
spec:
  capacity:

```

```

    storage: 35Gi
    accessModes:
      - ReadWriteMany
    gcePersistentDisk:
      readOnly: false
      pdName: the-newly-created-gce-PD
      fsType: ext4

```

Execute the object definition file:

```
# oc create -f pv-manual-gce.yaml
```

Now view the PVs again. Notice that a **pv-manual-gce** volume is *Available*.

```

# oc get pv
NAME                                     CAPACITY  ACCESSMODES
RECLAIMPOLICY   STATUS      CLAIM                                REASON    AGE
pv-manual-gce   Retain      Available                                     4s
pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi       RWX                                     12m
Delete         Bound      rh-eng/pvc-engineering2
pvc-ba4612ce-8b4d-11e6-9962-42010af00004  5Gi       RW0                                     21h
Delete         Bound      mytest/gce-dyn-claim1
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi      RWX                                     53m
Delete         Bound      rh-eng/pvc-engineering

```

Now create another claim identical to the **generic-pvc.yaml** PVC definition but change the name and do not set a storage class name.

Example 25.22. Claim Object Definition

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-engineering3
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 15Gi

```

Because *default StorageClass* is enabled in this instance, the manually created PV does not satisfy the claim request. The user receives a new dynamically provisioned Persistent Volume.

```

# oc get pvc
NAME                                     STATUS      VOLUME
CAPACITY  ACCESSMODES  AGE
pvc-engineering  Bound      pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX         1h
pvc-engineering2  Bound      pvc-a9f70544-8bfd-11e6-9962-42010af00004

```

5Gi	RWX	19m	
pvc-engineering3	Bound		pvc-6fa8e73b-8c00-11e6-9962-42010af00004
15Gi	RWX	6s	



IMPORTANT

Since the *default StorageClass* is enabled on this system, for the manually created Persistent Volume to get bound by the above claim and not have a new dynamic provisioned volume be bound, the PV would need to have been created in the *default StorageClass*.

Since the *default StorageClass* is enabled on this system, you would need to create the PV in the *default StorageClass* for the manually created Persistent Volume to get bound to the above claim and not have a new dynamic provisioned volume bound to the claim.

To fix this, the **cluster-admin** or **storage-admin** user simply needs to create another GCE disk or delete the first manual PV and use a PV object definition that assigns a *StorageClass* name (**pv-manual-gce2.yaml**) if necessary:

Example 25.23. Manual PV Spec with *default StorageClass* name

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-manual-gce2
spec:
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
  gcePersistentDisk:
    readOnly: false
    pdName: the-newly-created-gce-PD
    fsType: ext4
  storageClassName: generic 1
```

1 The name for previously created *generic StorageClass*.

Execute the object definition file:

```
# oc create -f pv-manual-gce2.yaml
```

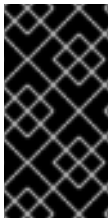
List the PVs:

```
# oc get pv
```

NAME	RECLAIMPOLICY	STATUS	CLAIM	CAPACITY	ACCESSMODES	REASON	AGE
pv-manual-gce	Retain	Available		35Gi	RWX		4s 1
pv-manual-gce2	Retain	Bound	rh-eng/pvc-engineering3	35Gi	RWX		4s 2

pvc-a9f70544-8bfd-11e6-9962-42010af00004	5Gi	RWX	
Delete	Bound	rh-eng/pvc-engineering2	12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004	5Gi	RWO	
Delete	Bound	mytest/gce-dyn-claim1	21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004	10Gi	RWX	
Delete	Bound	rh-eng/pvc-engineering	53m

- 1 The original manual PV, still unbound and Available. This is because it was not created in the *default StorageClass*.
- 2 The second PVC (other than the name) is bound to the Available manually created PV **pv-manual-gce2**.



IMPORTANT

Notice that all dynamically provisioned volumes by default have a *RECLAIMPOLICY* of *Delete*. Once the PVC dynamically bound to the PV is deleted, the GCE volume is deleted and all data is lost. However, the manually created PV has a default *RECLAIMPOLICY* of *Retain*.

25.11. USING STORAGE CLASSES FOR EXISTING LEGACY STORAGE

25.11.1. Overview

In this example, a legacy data volume exists and a **cluster-admin** or **storage-admin** needs to make it available for consumption in a particular project. Using *StorageClasses* decreases the likelihood of other users and projects gaining access to this volume from a claim because the claim would have to have an exact matching value for the *StorageClass* name. This example also disables dynamic provisioning. This example assumes:

- Some familiarity with OpenShift Container Platform, GCE, and Persistent Disks
- OpenShift Container Platform is [properly configured to use GCE](#).

25.11.1.1. Scenario 1: Link StorageClass to existing Persistent Volume with Legacy Data

As a **cluster-admin** or **storage-admin**, define and create the *StorageClass* for historical financial data.

Example 25.24. StorageClass finance-history Object Definitions

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: finance-history 1
provisioner: no-provisioning 2
parameters: 3
```

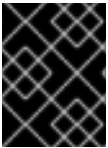
- 1 Name of the StorageClass.
- 2 This is a required field, but since there is to be no dynamic provisioning, a value must be put here as long as it is not an actual provisioner plug-in type.

- 3 Parameters can simply be left blank, since these are only used for the dynamic provisioner.

Save the definitions to a YAML file (**finance-history-storageclass.yaml**) and create the *StorageClass*.

```
# oc create -f finance-history-storageclass.yaml
storageclass "finance-history" created

# oc get storageclass
NAME                                TYPE
finance-history                    no-provisioning
```



IMPORTANT

cluster-admin or **storage-admin** users are responsible for relaying the correct *StorageClass* name to the correct users, groups, and projects.

The *StorageClass* exists. A **cluster-admin** or **storage-admin** can create the Persistent Volume (PV) for use with the *StorageClass*. Create a manually provisioned disk using [GCE](#) (not dynamically provisioned) and a [Persistent Volume](#) that connects to the new GCE disk (**gce-pv.yaml**).

Example 25.25. Finance History PV Object

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-finance-history
spec:
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
  gcePersistentDisk:
    readOnly: false
    pdName: the-existing-PD-volume-name-that-contains-the-valuable-data
    1 fsType: ext4
    storageClassName: finance-history 2
```

- 2 The *StorageClass* name, that must match exactly.

- 1 The name of the GCE disk that already exists and contains the legacy data.

As a **cluster-admin** or **storage-admin**, create and view the PV.

```
# oc create -f gce-pv.yaml
persistentvolume "pv-finance-history" created
```

```
# oc get pv
NAME                                CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM                               REASON    AGE
pv-finance-history 35Gi      RWX          Retain         Available
2d
```

Notice you have a **pv-finance-history** Available and ready for consumption.

As a user, create a Persistent Volume Claim (PVC) as a YAML file and specify the correct *StorageClass* name:

Example 25.26. Claim for finance-history Object Definition

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-finance-history
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
  storageClassName: finance-history ❶
```

- ❶ The *StorageClass* name, that must match exactly or the claim will go unbound until it is deleted or another *StorageClass* is created that matches the name.

Create and view the PVC and PV to see if it is bound.

```
# oc create -f pvc-finance-history.yaml
persistentvolumeclaim "pvc-finance-history" created

# oc get pvc
NAME                                STATUS  VOLUME                                CAPACITY
ACCESSMODES  AGE
pvc-finance-history  Bound  pv-finance-history 35Gi      RWX
9m

# oc get pv (cluster/storage-admin)
NAME                                CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM                               REASON    AGE
pv-finance-history 35Gi      RWX          Retain         Bound
default/pvc-finance-history                    5m
```



IMPORTANT

You can use *StorageClasses* in the same cluster for both legacy data (no dynamic provisioning) and with [dynamic provisioning](#).

25.12. CONFIGURING AZURE BLOB STORAGE FOR INTEGRATED DOCKER REGISTRY

25.12.1. Overview

This topic reviews how to configure [Microsoft Azure Blob Storage](#) for [OpenShift integrated Docker registry](#).

25.12.2. Before You Begin

- Create a storage container using Microsoft Azure Portal, Microsoft Azure CLI, or Microsoft Azure Storage Explorer. Keep a note of the **storage account name**, **storage account key** and **container name**.
- [Deploy the integrated Docker registry](#) if it is not deployed.

25.12.3. Overriding Registry Configuration

To create a new registry pod and replace the old pod automatically:

1. Create a new registry configuration file called **registryconfig.yaml** and add the following information:

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  azure: 1
    accountname: azureblobacc
    accountkey: azureblobacckey
    container: azureblobname
    realm: core.windows.net 2
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
    options:
      acceptschema2: false
      pullthrough: true
      enforcequota: false
      projectcachettl: 1m
      blobrepositorycachettl: 10m
storage:
  - name: openshift
```


- 1 Replace the values for **accountname**, **accountkey**, and **container** with **storage account name**, **storage account key**, and **storage container name** respectively.
- 2 If using Azure regional cloud, set to the desired realm. For example, **core.cloudapi.de** for the Germany regional cloud.

2. Create a new registry configuration:

```
$ oc create secret generic registry-config --from-
file=config.yaml=registryconfig.yaml
```

3. Add the secret:

```
$ oc volume dc/docker-registry --add --type=secret \
--secret-name=registry-config -m /etc/docker/registry/
```

4. Set the **REGISTRY_CONFIGURATION_PATH** environment variable:

```
$ oc set env dc/docker-registry \
REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yaml
```

5. If you already created a registry configuration:

a. Delete the secret:

```
$ oc delete secret registry-config
```

b. Create a new registry configuration:

```
$ oc create secret generic registry-config --from-
file=config.yaml=registryconfig.yaml
```

c. Update the configuration by starting a new rollout:

```
$ oc rollout latest docker-registry
```

CHAPTER 26. CONFIGURING EPHEMERAL STORAGE

26.1. OVERVIEW

OpenShift Container Platform can be configured to allow management of ephemeral storage of pod and container working data. While containers have been able to utilize writable layers, logs directories, and EmptyDir volumes, this storage has been subject to a number of limitations, [as discussed here](#).

Ephemeral storage management permits administrators to limit the resources consumed by individual pods and containers, and for pods and containers to specify requests and limits on their use of such ephemeral storage. This is a technology preview for OpenShift Container Platform 3.10 and is disabled by default.



NOTE

This technology preview does not change any of the mechanisms for making local storage available in OpenShift Container Platform; the existing mechanisms, root directory or runtime directory, still apply. This technology preview only provides a mechanism for managing the use of this resource.

26.2. ENABLING EPHEMERAL STORAGE

To enable ephemeral storage:

1. Edit or create the master configuration file on all masters, */etc/origin/master/master-config.yaml* by default, and add **LocalStorageCapacityIsolation=true** in the **apiServerArguments** and **controllerArguments** sections:

```
apiServerArguments:
  feature-gates:
    - LocalStorageCapacityIsolation=true
...

controllerArguments:
  feature-gates:
    - LocalStorageCapacityIsolation=true
...
```

2. Edit the ConfigMap for all nodes to enable the LocalStorageCapacityIsolation on the command line. You can identify the ConfigMaps that need to be edited as follows:

```
$ oc get cm -n openshift-node
NAME                DATA      AGE
node-config-compute  1          52m
node-config-infra    1          52m
node-config-master   1          52m
```

For each of these maps, **node-config-compute**, **node-config-infra**, and **node-config-master**, you need to add the feature gate:

```
oc edit cm node-config-master -n openshift-node
```

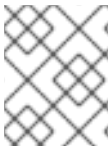
If there is already a **feature-gates:** declaration, add the following text to the list of feature gates:

```
,LocalStorageCapacityIsolation=true
```

If there is no **feature-gates:** declaration, add the following section:

```
feature-gates:
- LocalStorageCapacityIsolation=true
```

3. Repeat for **node-config-compute**, **node-config-infra**, and any other ConfigMaps.
4. Restart OpenShift Container Platform and delete the container running the apiserver.



NOTE

Omitting any of these steps may result in ephemeral storage management not being enabled.

CHAPTER 27. WORKING WITH HTTP PROXIES

27.1. OVERVIEW

Production environments can deny direct access to the Internet and instead have an HTTP or HTTPS proxy available. Configuring OpenShift Container Platform to use these proxies can be as simple as setting standard environment variables in configuration or JSON files. This can be done during an [cluster installation](#) or configured after installation.

The proxy configuration must be the same on each host in the cluster. Therefore, when setting up the proxy or modifying it, you must update the files on each OpenShift Container Platform host to the same values. Then, you must restart OpenShift Container Platform services on each host in the cluster.

The **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables are found in each host's */etc/origin/master/master.env* file and [node configuration map](#).

27.2. CONFIGURING NO_PROXY

The **NO_PROXY** environment variable lists all of the OpenShift Container Platform components and all IP addresses that are managed by OpenShift Container Platform.

On the OpenShift service accepting the CIDR, **NO_PROXY** accepts a comma-separated list of hosts, IP addresses, or IP ranges in CIDR format:

For master hosts

- Node host name
- Master IP or host name
- IP address of etcd hosts

For node hosts

- Master IP or host name

For the Docker service

- Registry service IP and host name
- Registry service URL **docker-registry.default.svc.cluster.local**
- Registry route host name (if created)



NOTE

When using Docker, Docker accepts a comma-separated list of hosts, domain extensions, or IP addresses, but does not accept IP ranges in CIDR format, which are only accepted by OpenShift services. The ``no_proxy`` variable should contain a comma-separated list of domain extensions that the proxy should *not* be used for.

For example, if **no_proxy** is set to **.school.edu**, the proxy will not be used to retrieve documents from the specific school.

NO_PROXY also includes the SDN network and service IP addresses as found in the ***master-config.yaml*** file.

/etc/origin/master/master-config.yaml

```
networkConfig:
  clusterNetworks:
  - cidr: 10.1.0.0/16
    hostSubnetLength: 9
  serviceNetworkCIDR: 172.30.0.0/16
```

OpenShift Container Platform does not accept ***** as a wildcard attached to a domain suffix. For example, the following would be accepted:

```
NO_PROXY=.example.com
```

However, the following would not be:

```
NO_PROXY=*.example.com
```

The only wildcard **NO_PROXY** accepts is a single ***** character, which matches all hosts, and effectively disables the proxy.

Each name in this list is matched as either a domain which contains the host name as a suffix, or the host name itself.



NOTE

When scaling up nodes, use a domain name rather than a list of hostnames.

For instance, **example.com** would match **example.com**, **example.com:80**, and **www.example.com**.

27.3. CONFIGURING HOSTS FOR PROXIES

1. Edit the proxy environment variables in the OpenShift Container Platform control files. Ensure all of the files in the cluster are correct.

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,10.1.0.0/16,172.30.0.0/16 1
```

- 1** Supports host names and CIDRs. Must include the SDN network and service IP ranges **10.1.0.0/16**, **172.30.0.0/16** by default.

2. Restart the master or node host:

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

27.4. CONFIGURING HOSTS FOR PROXIES USING ANSIBLE

During cluster installations, the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables can be configured using the **openshift_no_proxy**, **openshift_http_proxy**, and **openshift_https_proxy** parameters, which are configurable in the [inventory file](#).

Example Proxy Configuration with Ansible

```
# Global Proxy Configuration
# These options configure HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment
# variables for docker and master services.
openshift_http_proxy=http://<user>:<password>@<ip_addr>:<port>
openshift_https_proxy=https://<user>:<password>@<ip_addr>:<port>
openshift_no_proxy='.hosts.example.com,some-host.com'
#
# Most environments do not require a proxy between OpenShift masters,
# nodes, and
# etcd hosts. So automatically add those host names to the
# openshift_no_proxy list.
# If all of your hosts share a common domain you may wish to disable this
# and
# specify that domain above.
# openshift_generate_no_proxy_hosts=True
```

NOTE

There are [additional proxy settings](#) that can be [configured for builds](#) using Ansible parameters. For example:

The **openshift_builddefaults_git_http_proxy** and **openshift_builddefaults_git_https_proxy** parameters allow you to [use a proxy for Git cloning](#)

The **openshift_builddefaults_http_proxy** and **openshift_builddefaults_https_proxy** parameters can make environment variables available to the [Docker build strategy](#) and [Custom build strategy](#) processes.

27.5. PROXYING DOCKER PULL

OpenShift Container Platform node hosts need to perform push and pull operations to Docker registries. If you have a registry that does not need a proxy for nodes to access, include the **NO_PROXY** parameter with:

- the registry's host name,
- the registry service's IP address, and
- the service name.

This blacklists that registry, leaving the external HTTP proxy as the only option.

1. Retrieve the registry service's IP address **docker_registry_ip** by running:

```
$ oc describe svc/docker-registry -n default

Name:    docker-registry
```

```

Namespace: default
Labels:    docker-registry=default
Selector:  docker-registry=default
Type:      ClusterIP
IP:        172.30.163.183 1
Port:      5000-tcp 5000/TCP
Endpoints: 10.1.0.40:5000
Session Affinity: ClientIP
No events.

```

1 Registry service IP.

2. Edit the `/etc/sysconfig/docker` file and add the **NO_PROXY** variables in shell format, replacing `<docker_registry_ip>` with the IP address from the previous step.

```

HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,<docker_registry_ip>,docker-
registry.default.svc.cluster.local

```

3. Restart the Docker service:

```
# systemctl restart docker
```

27.6. USING MAVEN BEHIND A PROXY

Using Maven with proxies requires using the **HTTP_PROXY_NONPROXYHOSTS** variable.

See the [Red Hat JBoss Enterprise Application Platform for OpenShift documentation](#) for information about configuring your OpenShift Container Platform environment for Red Hat JBoss Enterprise Application Platform, including the step for setting up Maven behind a proxy.

27.7. CONFIGURING S2I BUILDS FOR PROXIES

S2I builds fetch dependencies from various locations. You can [use a `.s2i/environment` file](#) to specify simple shell variables and OpenShift Container Platform will react accordingly when seeing build images.

The following are the supported proxy environment variables with example values:

```

HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com

```

27.8. CONFIGURING DEFAULT TEMPLATES FOR PROXIES

The [example templates](#) available in OpenShift Container Platform by default do not include settings for HTTP proxies. For existing applications based on these templates, modify the **source** section of the application's build configuration and add proxy settings:

```

...
source:
  type: Git

```

```
git:
  uri: https://github.com/openshift/ruby-hello-world
  httpProxy: http://proxy.example.com
  httpsProxy: https://proxy.example.com
  noProxy: somedomain.com, otherdomain.com
...
```

This is similar to the process for [using proxies for Git cloning](#).

27.9. SETTING PROXY ENVIRONMENT VARIABLES IN PODS

You can set the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables in the **templates.spec.containers** stanza in a deployment configuration to pass proxy connection information. The same can be done for configuring a Pod's proxy at runtime:

```
...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://<user>:<password>@<ip_addr>:<port>"
...
```

You can also use the **oc set env** command to update an existing deployment configuration with a new environment variable:

```
$ oc set env dc/frontend HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>
```

If you have a [ConfigChange trigger](#) set up in your OpenShift Container Platform instance, the changes happen automatically. Otherwise, manually redeploy your application for the changes to take effect.

27.10. GIT REPOSITORY ACCESS

If your Git repository can only be accessed using a proxy, you can define the proxy to use in the **source** section of the **BuildConfig**. You can configure both a HTTP and HTTPS proxy to use. Both fields are optional. Domains for which no proxying should be performed can also be specified via the **NoProxy** field.



NOTE

Your source URI must use the HTTP or HTTPS protocol for this to work.

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```


CHAPTER 28. CONFIGURING GLOBAL BUILD DEFAULTS AND OVERRIDES

28.1. OVERVIEW

Developers can define settings in specific build configurations within their projects, such as [configuring a proxy for Git cloning](#). Rather than requiring developers to define certain settings in each build configuration, administrators can use admission control plug-ins to configure global build defaults and overrides that automatically use these settings in any build.

The settings from these plug-ins are used only during the build process but are not set in the build configurations or builds themselves. Configuring the settings through the plug-ins allows administrators to change the global configuration at any time, and any builds that are re-run from existing build configurations or builds are assigned the new settings.

- The **BuildDefaults** admission control plug-in allows administrators to set global defaults for settings such as the Git HTTP and HTTPS proxy, as well as default environment variables. These defaults do not overwrite values that have been configured for a specific build. However, if those values are not present on the build definition, they are set to the default value.
- The **BuildOverrides** admission control plug-in allows administrators to override a setting in a build, regardless of the value stored in the build. The plug-in currently supports [overriding the forcePull flag on a build strategy](#) to force refreshing the local image from the registry during a build. Refreshing ensures that users can build only with images that they are allowed to pull. The plug-in can also be configured to apply a set of image labels to every built image. For information on configuring the **BuildOverrides** admission control plug-in and the values you can override, see [Manually Setting Global Build Overrides](#).

The default node selectors and the **BuildDefaults** or **BuildOverride** admission plug-ins work together as follows:

- The default project node selector, defined in the **projectConfig.defaultNodeSelector** field in the master configuration file, is applied to the pods created in all projects without a specified **nodeSelector** value. These settings are applied to builds with **nodeSelector="null"** on clusters where the **BuildDefaults** or **BuildOverride** nodeSelector is not set.
- The cluster-wide default build node selector, **admissionConfig.pluginConfig.BuildDefaults.configuration.nodeSelector**, is applied only if the **nodeSelector="null"** parameter is set in the build configuration. **nodeSelector=null** is the default setting.
- With a default project or cluster-wide node selector, the default setting is added as an AND to the build node selector, which is set by the **BuildDefaults** or **BuildOverride** admission plug-ins. These settings mean that the build will be scheduled only to nodes that satisfy the **BuildOverrides** node selector AND the project default node selector.



NOTE

You can define a hard limit on how long build pods can run by using the [RunOnceDuration](#) plugin.

28.2. SETTING GLOBAL BUILD DEFAULTS

You can set global build defaults two ways:

- [using Ansible](#)
- [manually by modifying the *master-config.yaml* file](#)

28.2.1. Configuring Global Build Defaults with Ansible

During cluster installations, the **BuildDefaults** plug-in can be configured using [the following parameters](#), which are configurable in the inventory file:

- **openshift_builddefaults_http_proxy**
- **openshift_builddefaults_https_proxy**
- **openshift_builddefaults_no_proxy**
- **openshift_builddefaults_git_http_proxy**
- **openshift_builddefaults_git_https_proxy**
- **openshift_builddefaults_git_no_proxy**
- **openshift_builddefaults_image_labels**
- **openshift_builddefaults_nodeselectors**
- **openshift_builddefaults_annotations**
- **openshift_builddefaults_resources_requests_cpu**
- **openshift_builddefaults_resources_requests_memory**
- **openshift_builddefaults_resources_limits_cpu**
- **openshift_builddefaults_resources_limits_memory**

Example 28.1. Example Build Defaults Configuration with Ansible

```
# These options configure the BuildDefaults admission controller which
# injects
# configuration into Builds. Proxy related values will default to the
# global proxy
# config values. You only need to set these if they differ from the
# global proxy settings.
openshift_builddefaults_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_no_proxy=mycorp.com
openshift_builddefaults_git_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_no_proxy=mycorp.com
openshift_builddefaults_image_labels=
[{'name': 'imagelabelname1', 'value': 'imagelabelvalue1'}]
openshift_builddefaults_nodeselectors={'nodelabel1': 'nodelabelvalue1'}
openshift_builddefaults_annotations=
{'annotationkey1': 'annotationvalue1'}
```

```

openshift_builddefaults_resources_requests_cpu=100m
openshift_builddefaults_resources_requests_memory=256Mi
openshift_builddefaults_resources_limits_cpu=1000m
openshift_builddefaults_resources_limits_memory=512Mi

# Or you may optionally define your own build defaults configuration
serialized as json
#openshift_builddefaults_json='{ "BuildDefaults": { "configuration":
{ "apiVersion": "v1", "env":
[ { "name": "HTTP_PROXY", "value": "http://proxy.example.com.redhat.com:3128"
}, { "name": "NO_PROXY", "value": "ose3-
master.example.com" } ], "gitHTTPProxy": "http://proxy.example.com:3128", "gi
tNoProxy": "ose3-master.example.com", "kind": "BuildDefaultsConfig" } } }'
```

28.2.2. Manually Setting Global Build Defaults

To configure the **BuildDefaults** plug-in:

1. Add a configuration for it in the */etc/origin/master/master-config.yaml* file on the master nodes:

```

admissionConfig:
  pluginConfig:
    BuildDefaults:
      configuration:
        apiVersion: v1
        kind: BuildDefaultsConfig
        gitHTTPProxy: http://my.proxy:8080 ①
        gitHTTPSProxy: https://my.proxy:8443 ②
        gitNoProxy: somedomain.com, otherdomain.com ③
        env:
          - name: HTTP_PROXY ④
            value: http://my.proxy:8080
          - name: HTTPS_PROXY ⑤
            value: https://my.proxy:8443
          - name: BUILD_LOGLEVEL ⑥
            value: 4
          - name: CUSTOM_VAR ⑦
            value: custom_value
        imageLabels:
          - name: url ⑧
            value: https://containers.example.org
          - name: vendor
            value: ExampleCorp Ltd.
        nodeSelector: ⑨
          key1: value1
          key2: value2
        annotations: ⑩
          key1: value1
          key2: value2
        resources: ⑪
          requests:
            cpu: "100m"
```

```
memory: "256Mi"
limits:
  cpu: "100m"
  memory: "256Mi"
```

- 1 Sets the HTTP proxy to use when cloning source code from a Git repository.
- 2 Sets the HTTPS proxy to use when cloning source code from a Git repository.
- 3 Sets the list of domains for which proxying should not be performed.
- 4 Default environment variable that sets the HTTP proxy to use during the build. This can be used for downloading dependencies during the assemble and build phases.
- 5 Default environment variable that sets the HTTPS proxy to use during the build. This can be used for downloading dependencies during the assemble and build phases.
- 6 Default environment variable that sets the build log level during the build.
- 7 Additional default environment variable that will be added to every build.
- 8 Labels to be applied to every image built. Users can override these in their **BuildConfig**.
- 9 Build pods will only run on nodes with the **key1=value2** and **key2=value2** labels. Users can define a different set of **nodeSelectors** for their builds in which case these values will be ignored.
- 10 Build pods will have these annotations added to them.
- 11 Sets the default resources to the build pod if the **BuildConfig** does not have related resource defined.

2. Restart the master services for the changes to take effect:

```
# master-restart api
# master-restart controllers
```

28.3. SETTING GLOBAL BUILD OVERRIDES

You can set global build overrides two ways:

- [using Ansible](#)
- [manually by modifying the *master-config.yaml* file](#)

28.3.1. Configuring Global Build Overrides with Ansible

During cluster installations, the **BuildOverrides** plug-in can be configured using the following parameters, which are configurable in the inventory file:

- **openshift_buildoverrides_force_pull**
- **openshift_buildoverrides_image_labels**

- `openshift_buildoverrides_nodeselectors`
- `openshift_buildoverrides_annotations`
- `openshift_buildoverrides_tolerations`

Example 28.2. Example Build Overrides Configuration with Ansible

```
# These options configure the BuildOverrides admission controller which
# injects
# configuration into Builds.
openshift_buildoverrides_force_pull=true
openshift_buildoverrides_image_labels=
[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
openshift_buildoverrides_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_buildoverrides_annotations=
{'annotationkey1':'annotationvalue1'}
openshift_buildoverrides_tolerations=
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Equal'}]

# Or you may optionally define your own build overrides configuration
# serialized as json
#openshift_buildoverrides_json='{"BuildOverrides":{"configuration":
{"apiVersion":"v1","kind":"BuildOverridesConfig","forcePull":"true","tolerations":
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Equal'}]}}}'
```

28.3.2. Manually Setting Global Build Overrides

To configure the **BuildOverrides** plug-in:

1. Add a configuration for it in the `/etc/origin/master/master-config.yaml` file on masters:

```
admissionConfig:
  pluginConfig:
    BuildOverrides:
      configuration:
        apiVersion: v1
        kind: BuildOverridesConfig
        forcePull: true ①
        imageLabels:
          - name: distribution-scope ②
            value: private
        nodeSelector: ③
          key1: value1
          key2: value2
        annotations: ④
          key1: value1
          key2: value2
        tolerations: ⑤
          - key: mykey1
```

```
value: myvalue1
effect: NoSchedule
operator: Equal
- key: mykey2
  value: myvalue2
  effect: NoExecute
  operator: Equal
```

- 1 Force all builds to pull their builder image and any source images before starting the build.
- 2 Additional labels to be applied to every image built. Labels defined here take precedence over labels defined in **BuildConfig**.
- 3 Build pods will only run on nodes with the **key1=value2** and **key2=value2** labels. Users can define additional key/value labels to further constrain the set of nodes a build runs on, but the **node** must have at least these labels.
- 4 Build pods will have these annotations added to them.
- 5 Build pods will have any existing tolerations overridden by those listed here.

2. Restart the master services for the changes to take effect:

```
# master-restart api
# master-restart controllers
```

CHAPTER 29. CONFIGURING PIPELINE EXECUTION

29.1. OVERVIEW

The first time a user creates a build configuration using the [Pipeline](#) build strategy, OpenShift Container Platform looks for a template named **jenkins-ephemeral** in the **openshift** namespace and instantiates it within the user's project. The **jenkins-ephemeral** template that ships with OpenShift Container Platform creates, upon instantiation:

- a deployment configuration for Jenkins using the official OpenShift Container Platform Jenkins image
- a service and route for accessing the Jenkins deployment
- a new Jenkins service account
- rolebindings to grant the service account edit access to the project

Cluster administrators can control what is created by either modifying the content of the built-in template, or by editing the cluster configuration to direct the cluster to a different template location.

To modify the content of the default template:

```
$ oc edit template jenkins-ephemeral -n openshift
```

To use a different template, such as the **jenkins-persistent** template which uses persistent storage for Jenkins, add the following to your master configuration file:

```
jenkinsPipelineConfig:
  autoProvisionEnabled: true 1
  templateNamespace: openshift 2
  templateName: jenkins-persistent 3
  serviceName: jenkins-persistent-svc 4
  parameters: 5
    key1: value1
    key2: value2
```

- 1 Defaults to **true** if unspecified. If **false**, then no template will be instantiated.
- 2 Namespace containing the template to be instantiated.
- 3 Name of the template to be instantiated.
- 4 Name of the service to be created by the template upon instantiation.
- 5 Optional values to pass to the template during instantiation.

When a Pipeline build configuration is created, OpenShift Container Platform looks for a Service matching **serviceName**. This means **serviceName** must be chosen such that it is unique in the project. If no Service is found, OpenShift Container Platform instantiates the **jenkinsPipelineConfig** template. If this is not desirable (if you would like to use a Jenkins server external to OpenShift Container Platform, for example), there are a few things you can do, depending on who you are.

- If you are a cluster administrator, simply set **autoProvisionEnabled** to **false**. This will disable autoprovisioning across the cluster.
- If you are an unprivileged user, a Service must be created for OpenShift Container Platform to use. The service name must match the cluster configuration value of **serviceName** in the **jenkinsPipelineConfig**. The default value is **jenkins**. If you are disabling autoprovisioning because you are running a Jenkins server outside your project, it is recommended that you point this new service to your existing Jenkins server. See: [Integrating External Services](#)

The latter option could also be used to disable autoprovisioning in select projects only.

29.2. OPENSIFT JENKINS CLIENT PLUGIN

The OpenShift Jenkins Client Plugin is a Jenkins plugin which aims to provide a readable, concise, comprehensive, and fluent Jenkins Pipeline syntax for rich interactions with an OpenShift API Server. The plugin leverages the OpenShift command line tool (**oc**) which must be available on the nodes executing the script.

For more information about installing and configuring the plugin, use the links provided below that reference the official documentation.

- [Installing](#)
- [Configuring an OpenShift Cluster](#)
- [Setting up Credentials](#)
- [Setting up Jenkins Nodes](#)



NOTE

Are you a developer looking for information about using this plugin? If so, see [OpenShift Pipeline Overview](#).

29.3. OPENSIFT JENKINS SYNC PLUGIN

This Jenkins plugin keeps OpenShift BuildConfig and Build objects in sync with Jenkins Jobs and Builds.

The OpenShift jenkins Sync Plugin provides the following:

- Dynamic job/run creation in Jenkins.
- Dynamic creation of slave pod templates from ImageStreams, ImageStreamTags, or ConfigMaps.
- Injecting of environment variables.
- Pipeline visualization in the OpenShift web console.
- Integration with the Jenkins git plugin, which passes commit information from OpenShift builds to the Jenkins git plugin.

For more information about this plugin, see:

- [OpenShift Jenkins Sync Plug-in](#)

- [OpenShift Container Platform Sync Plug-in](#)

CHAPTER 30. CONFIGURING ROUTE TIMEOUTS

After installing OpenShift Container Platform and [deploying a router](#), you can configure the default timeouts for an existing route when you have services in need of a low timeout, as required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

Using the **oc annotate** command, add the timeout to the route:

```
# oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

For example, to set a route named **myroute** to a timeout of two seconds:

```
# oc annotate route myroute --overwrite
haproxy.router.openshift.io/timeout=2s
```

Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

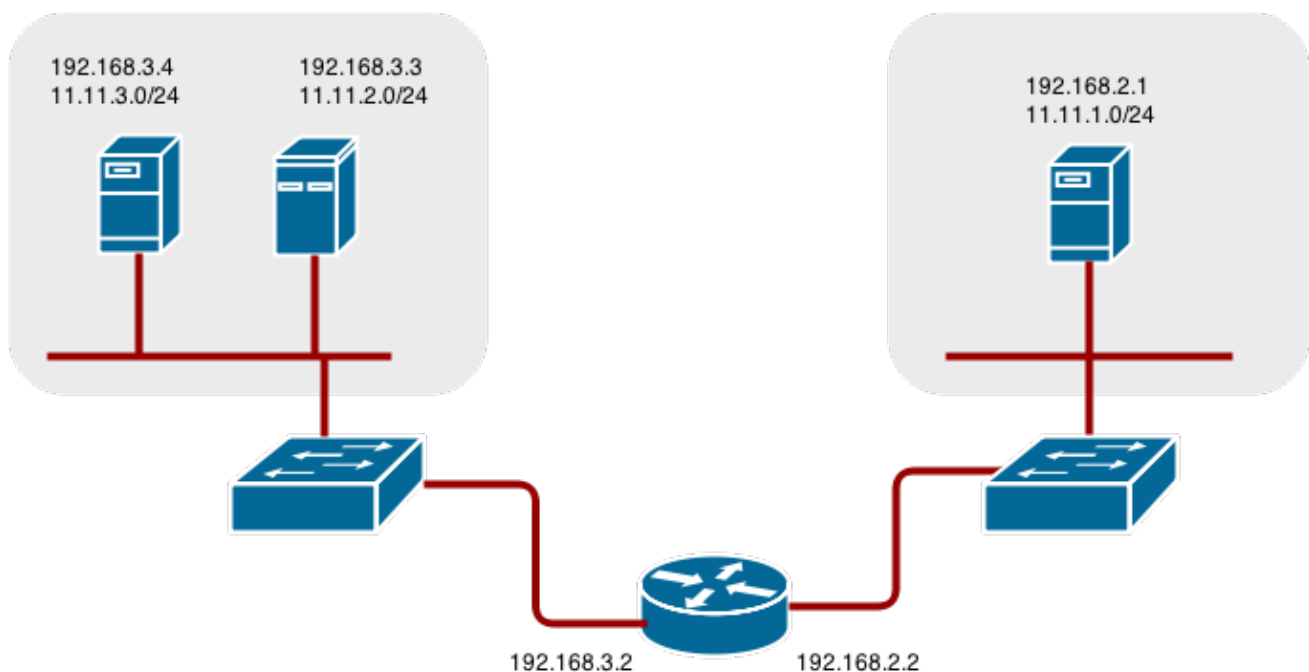
CHAPTER 31. CONFIGURING NATIVE CONTAINER ROUTING

31.1. NETWORK OVERVIEW

The following describes a general network setup:

- 11.11.0.0/16 is the container network.
- The 11.11.x.0/24 subnet is reserved for each node and assigned to the Docker Linux bridge.
- Each node has a route to the router for reaching anything in the 11.11.0.0/16 range, except the local subnet.
- The router has routes for each node, so it can be directed to the right node.
- Existing nodes do not need any changes when new nodes are added, unless the network topology is modified.
- IP forwarding is enabled on each node.

The following diagram shows the container networking setup described in this topic. It uses one Linux node with two network interface cards serving as a router, two switches, and three nodes connected to these switches.



31.2. CONFIGURE NATIVE CONTAINER ROUTING

You can set up container networking using existing switches and routers, and the kernel networking stack in Linux.

As a network administrator, you must modify, or create a script to modify, the router or routers when new nodes are added to the cluster.

You can adapt this process to use with any type of router.

31.3. SETTING UP A NODE FOR CONTAINER NETWORKING

1. Assign an unused 11.11.x.0/24 subnet IP address to the Linux bridge on the node:

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

2. Modify the Docker startup script to use the new bridge. By default, the startup script is the **/etc/sysconfig/docker** file:

```
# docker -d -b lbr0 --other-options
```

3. Add a route to the router for the 11.11.0.0/16 network:

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

4. Enable IP forwarding on the node:

```
# sysctl -w net.ipv4.ip_forward=1
```

31.4. SETTING UP A ROUTER FOR CONTAINER NETWORKING

The following procedure assumes a Linux box with multiple NICs is used as a router. Modify the steps as required to use the syntax for a particular router:

1. Enable IP forwarding on the router:

```
# sysctl -w net.ipv4.ip_forward=1
```

2. Add a route for each node added to the cluster:

```
# ip route add <node_subnet> via <node_ip_address> dev <interface
through which node is L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

CHAPTER 32. ROUTING FROM EDGE LOAD BALANCERS

32.1. OVERVIEW

Pods inside of an OpenShift Container Platform cluster are only reachable via their IP addresses on the cluster network. An edge load balancer can be used to accept traffic from outside networks and proxy the traffic to pods inside the OpenShift Container Platform cluster. In cases where the load balancer is not part of the cluster network, routing becomes a hurdle as the internal cluster network is not accessible to the edge load balancer.

To solve this problem where the OpenShift Container Platform cluster is using [OpenShift Container Platform SDN](#) as the cluster networking solution, there are two ways to achieve network access to the pods.

32.2. INCLUDING THE LOAD BALANCER IN THE SDN

If possible, run an OpenShift Container Platform node instance on the load balancer itself that uses OpenShift SDN as the networking plug-in. This way, the edge machine gets its own Open vSwitch bridge that the SDN automatically configures to provide access to the pods and nodes that reside in the cluster. The *routing table* is dynamically configured by the SDN as pods are created and deleted, and thus the routing software is able to reach the pods.

Mark the load balancer machine as an [unschedulable node](#) so that no pods end up on the load balancer itself:

```
$ oc adm manage-node <load_balancer_hostname> --schedulable=false
```

If the load balancer comes packaged as a container, then it is even easier to integrate with OpenShift Container Platform: Simply run the load balancer as a pod with the host port exposed. The pre-packaged [HAProxy router](#) in OpenShift Container Platform runs in precisely this fashion.

32.3. ESTABLISHING A TUNNEL USING A RAMP NODE

In some cases, the previous solution is not possible. For example, an **F5 BIG-IP®** host cannot run an OpenShift Container Platform node instance or the OpenShift Container Platform SDN because **F5®** uses a custom, incompatible Linux kernel and distribution.

Instead, to enable **F5 BIG-IP®** to reach pods, you can choose an existing node within the cluster network as a *ramp node* and establish a tunnel between the **F5 BIG-IP®** host and the designated ramp node. Because it is otherwise an ordinary OpenShift Container Platform node, the ramp node has the necessary configuration to route traffic to any pod on any node in the cluster network. The ramp node thus assumes the role of a gateway through which the **F5 BIG-IP®** host has access to the entire cluster network.

Following is an example of establishing an **ipip** tunnel between an **F5 BIG-IP®** host and a designated ramp node.

On the F5 BIG-IP® host:

1. Set the following variables:

```
# F5_IP=10.3.89.66 1
# RAMP_IP=10.3.89.89 2
```

```
# TUNNEL_IP1=10.3.91.216 3
# CLUSTER_NETWORK=10.128.0.0/14 4
```

- 1** **2** The **F5_IP** and **RAMP_IP** variables refer to the **F5 BIG-IP®** host's and the ramp node's IP addresses, respectively, on a shared, internal network.
- 3** An arbitrary, non-conflicting IP address for the **F5®** host's end of the **ipip** tunnel.
- 4** The overlay network CIDR range that the OpenShift SDN uses to assign addresses to pods.

2. Delete any old route, self, tunnel and SNAT pool:

```
# tmsh delete net route $CLUSTER_NETWORK || true
# tmsh delete net self SDN || true
# tmsh delete net tunnels tunnel SDN || true
# tmsh delete ltm snatpool SDN_snatpool || true
```

3. Create the new tunnel, self, route and SNAT pool and use the SNAT pool in the virtual servers:

```
# tmsh create net tunnels tunnel SDN \
  \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsh create net self SDN \{ address \
  ${TUNNEL_IP1}/24 allow-service all vlan SDN \}
# tmsh create net route $CLUSTER_NETWORK interface SDN
# tmsh create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1 }
# tmsh modify ltm virtual ose-vserver source-address-translation {
  type snat pool SDN_snatpool }
# tmsh modify ltm virtual https-ose-vserver source-address-
translation { type snat pool SDN_snatpool }
```

On the ramp node:



NOTE

The following creates a configuration that is not persistent, meaning that when the ramp node or the **openvswitch** service is restarted, the settings disappear.

1. Set the following variables:

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217 1
# CLUSTER_NETWORK=10.128.0.0/14 2
```

- 1** A second, arbitrary IP address for the ramp node's end of the **ipip** tunnel.
- 2** The overlay network CIDR range that the OpenShift SDN uses to assign addresses to pods.

2. Delete any old tunnel:

```
# ip tunnel del tun1 || true
```

3. Create the **ipip** tunnel on the ramp node, using a suitable L2-connected interface (e.g., **eth0**):

```
# ip tunnel add tun1 mode ipip \
    remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

4. SNAT the tunnel IP with an unused IP from the SDN subnet:

```
# source /run/openshift-sdn/config.env
# tap1=$(ip -o -4 addr list tun0 | awk '{print $4}' | cut -d/ -f1 |
head -n 1)
# subaddr=$(echo ${OPENSIFT_SDN_TAP1_ADDR:-"$tap1"} | cut -d "." -f
1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

5. Assign this **RAMP_SDN_IP** as an additional address to **tun0** (the local SDN's gateway):

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

6. Modify the OVS rules for SNAT:

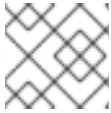
```
# ipflowopts="cookie=0x999,ip"
# arpflowopts="cookie=0x999, table=0, arp"
#
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN_IP
},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_IP1
},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, priority=200, in_port=2,
arp_spa=${RAMP_SDN_IP}, arp_tpa=${CLUSTER_NETWORK},
actions=goto_table:30"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "arp, table=5, priority=300, arp_tpa=${RAMP_SDN_IP},
actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "ip, table=5, priority=300, nw_dst=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0
"${ipflowopts},nw_dst=${TUNNEL_IP1},actions=output:2"
```

7. Optionally, if you do not plan on configuring the ramp node to be highly available, mark the ramp node as unschedulable. Skip this step if you do plan to follow the next section and plan on creating a highly available ramp node.

```
$ oc adm manage-node <ramp_node_hostname> --schedulable=false
```



NOTE

The [F5 router plug-in](#) integrates with F5 BIG-IP®.

32.3.1. Configuring a Highly-Available Ramp Node

You can use OpenShift Container Platform's **ipfailover** feature, which uses **keepalived** internally, to make the ramp node highly available from **F5 BIG-IP®**'s point of view. To do so, first bring up two nodes, for example called **ramp-node-1** and **ramp-node-2**, on the same L2 subnet.

Then, choose some unassigned IP address from within the same subnet to use for your virtual IP, or *VIP*. This will be set as the **RAMP_IP** variable with which you will configure your tunnel on **F5 BIG-IP®**.

For example, suppose you are using the **10.20.30.0/24** subnet for your ramp nodes, and you have assigned **10.20.30.2** to **ramp-node-1** and **10.20.30.3** to **ramp-node-2**. For your VIP, choose some unassigned address from the same **10.20.30.0/24** subnet, for example **10.20.30.4**. Then, to configure **ipfailover**, mark both nodes with a label, such as **f5ramptime**:

```
$ oc label node ramp-node-1 f5ramptime=true
$ oc label node ramp-node-2 f5ramptime=true
```

Similar to instructions from the [ipfailover documentation](#), you must now create a service account and add it to the **privileged** SCC. First, create the **f5ipfailover** service account:

```
$ oc create serviceaccount f5ipfailover -n default
```

Next, you can add the **f5ipfailover** service to the **privileged** SCC. To add the **f5ipfailover** in the **default** namespace to the **privileged** SCC, run:

```
$ oc adm policy add-scc-to-user privileged
system:serviceaccount:default:f5ipfailover
```

Finally, configure **ipfailover** using your chosen VIP (the **RAMP_IP** variable) and the **f5ipfailover** service account, assigning the VIP to your two nodes using the **f5ramptime** label you set earlier:

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 ❶
# oc adm ipfailover <name-tag> \
  --virtual-ips=$RAMP_IP \
  --interface=$IFNAME \
  --watch-port=0 \
  --replicas=2 \
  --service-account=f5ipfailover \
  --selector='f5ramptime=true'
```

❶ The interface where **RAMP_IP** should be configured.

With the above setup, the VIP (the **RAMP_IP** variable) is automatically re-assigned when the ramp node host that currently has it assigned fails.

CHAPTER 33. AGGREGATING CONTAINER LOGS

33.1. OVERVIEW

As an OpenShift Container Platform cluster administrator, you can deploy the EFK stack to aggregate logs for a range of OpenShift Container Platform services. Application developers can view the logs of the projects for which they have view access. The EFK stack aggregates logs from hosts and applications, whether coming from multiple containers or even deleted pods.

The EFK stack is a modified version of the [ELK stack](#) and is comprised of:

- [Elasticsearch \(ES\)](#): An object store where all logs are stored.
- [Fluentd](#): Gathers logs from nodes and feeds them to Elasticsearch.
- [Kibana](#): A web UI for Elasticsearch.

Once deployed in a cluster, the stack aggregates logs from all nodes and projects into Elasticsearch, and provides a Kibana UI to view any logs. Cluster administrators can view all logs, but application developers can only view logs for projects they have permission to view. The stack components communicate securely.



NOTE

[Managing Docker Container Logs](#) discusses the use of **json-file** logging driver options to manage container logs and prevent filling node disks.

33.2. PRE-DEPLOYMENT CONFIGURATION

1. An Ansible playbook is available to deploy and upgrade aggregated logging. You should familiarize yourself with the [Installing Clusters](#) guide. This provides information for preparing to use Ansible and includes information about configuration. Parameters are added to the Ansible inventory file to configure various areas of the EFK stack.
2. Review the [sizing guidelines](#) to determine how best to configure your deployment.
3. Ensure that you have deployed a router for the cluster.
4. Ensure that you have [the necessary storage](#) for Elasticsearch. Note that each Elasticsearch replica requires its own storage volume. See [Elasticsearch](#) for more information.
5. Choose a project. Once deployed, the EFK stack collects logs for every project within your OpenShift Container Platform cluster. The examples in this section use the default project **logging**. The Ansible playbook creates the project for you if it does not already exist. You will only need to create a project if you want to specify a node-selector on it. Otherwise, the **openshift-logging** role will create a project.

```
$ oc adm new-project logging --node-selector=""
$ oc project logging
```

**NOTE**

Specifying an empty [node selector](#) on the project is recommended, as Fluentd should be deployed throughout the cluster and any selector would restrict where it is deployed. To control component placement, specify node selectors per component to be applied to their deployment configurations.

33.3. SPECIFYING LOGGING ANSIBLE VARIABLES

Parameters for the EFK deployment may be specified to the [inventory host file](#) to override the [defaults](#). Read the [Elasticsearch](#) and the [Fluentd](#) sections before choosing parameters:

**NOTE**

By default the Elasticsearch service uses port 9300 for TCP communication between nodes in a cluster.

Parameter	Description
openshift_logging_use_ops	If set to true , configures a second Elasticsearch cluster and Kibana for operations logs. Fluentd splits logs between the main cluster and a cluster reserved for operations logs, which consists of the logs from the projects default , openshift , and openshift-infra , as well as Docker, OpenShift, and system logs from the journal. This means a second Elasticsearch cluster and Kibana are deployed. The deployments are distinguishable by the -ops suffix included in their names and have parallel deployment options listed below and described in Creating the Curator Configuration .
openshift_logging_master_url	The URL for the Kubernetes master, this does not need to be public facing but should be accessible from within the cluster. For example, https://<PRIVATE-MASTER-URL>:8443 .
openshift_logging_master_public_url	The public facing URL for the Kubernetes master. This is used for Authentication redirection by the Kibana proxy. For example, https://<CONSOLE-PUBLIC-URL-MASTER>:8443 .
openshift_logging	The namespace where Aggregated Logging is deployed.
openshift_logging_install_logging	Set to true to install logging. Set to false to uninstall logging.
openshift_logging_purge_logging	The common uninstall keeps PVC to prevent unwanted data loss during reinstalls. To ensure that the Ansible playbook completely and irreversibly removes all logging persistent data including PVC, set openshift_logging_install_logging to 'false' to trigger uninstallation and openshift_logging_purge_logging to 'true'. The default is set to 'false'.
openshift_logging_install_eventrouter	Coupled with openshift_logging_install_logging . When both are set to 'true', eventrouter will be installed. When both are 'false', eventrouter will be uninstalled.

Parameter	Description
<code>openshift_logging_eventrouter_image_prefix</code>	The prefix for the eventrouter logging image.
<code>openshift_logging_eventrouter_image_version</code>	The image version for the logging eventrouter .
<code>openshift_logging_eventrouter_sink</code>	Select a sink for eventrouter , supported stdout and glog . The default is set to stdout .
<code>openshift_logging_eventrouter_nodeselect</code>	A map of labels, such as <code>"node": "infra", "region": "west"</code> , to select the nodes where the pod will land.
<code>openshift_logging_eventrouter_replicas</code>	The default is set to '1'.
<code>openshift_logging_eventrouter_cpu_limit</code>	The minimum amount of CPU to allocate to eventrouter . The default is set to '100m'.
<code>openshift_logging_eventrouter_memory_limit</code>	The memory limit for eventrouter pods. The default is set to '128Mi'.
<code>openshift_logging_eventrouter_namespace</code>	<p>The project where eventrouter is deployed. The default is set to 'default'.</p> <div>  <div> <p>IMPORTANT</p> <p>Do not set the project to anything other than default or openshift-*. If you specify a different project, event information from the other project can leak into indices that are not restricted to operations users. To use a non-default project, create the project as usual using oc new-project.</p> </div> </div>
<code>openshift_logging_image_pull_secret</code>	Specify the name of an existing pull secret to be used for pulling component images from an authenticated registry.
<code>openshift_logging_curator_default_days</code>	The default minimum age (in days) Curator uses for deleting log records.
<code>openshift_logging_curator_run_hour</code>	The hour of the day Curator will run.
<code>openshift_logging_curator_run_minute</code>	The minute of the hour Curator will run.

Parameter	Description
openshift_logging_curator_run_timezone	The timezone Curator uses for figuring out its run time. Provide the timezone as a string in the tzselect(8) or timedatectl(1) "Region/Locality" format, for example America/New_York or UTC .
openshift_logging_curator_script_log_level	The script log level for Curator.
openshift_logging_curator_log_level	The log level for the Curator process.
openshift_logging_curator_cpu_limit	The amount of CPU to allocate to Curator.
openshift_logging_curator_memory_limit	The amount of memory to allocate to Curator.
openshift_logging_curator_nodeselector	A node selector that specifies which nodes are eligible targets for deploying Curator instances.
openshift_logging_curator_ops_cpu_limit	Equivalent to openshift_logging_curator_cpu_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_curator_ops_memory_limit	Equivalent to openshift_logging_curator_memory_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_hostname	The external host name for web clients to reach Kibana.
openshift_logging_kibana_cpu_limit	The amount of CPU to allocate to Kibana.
openshift_logging_kibana_memory_limit	The amount of memory to allocate to Kibana.
openshift_logging_kibana_proxy_debug	When true , set the Kibana Proxy log level to DEBUG.
openshift_logging_kibana_proxy_cpu_limit	The amount of CPU to allocate to Kibana proxy.
openshift_logging_kibana_proxy_memory_limit	The amount of memory to allocate to Kibana proxy.
openshift_logging_kibana_replica_count	The number of replicas to which Kibana should be scaled up.

Parameter	Description
openshift_logging_kibana_nodeselector	A node selector that specifies which nodes are eligible targets for deploying Kibana instances.
openshift_logging_kibana_env_vars	A map of environment variables to add to the Kibana deployment configuration. For example, <code>{"ELASTICSEARCH_REQUESTTIMEOUT":"30000"}</code> .
openshift_logging_kibana_key	The public facing key to use when creating the Kibana route.
openshift_logging_kibana_cert	The cert that matches the key when creating the Kibana route.
openshift_logging_kibana_ca	Optional. The CA to goes with the key and cert used when creating the Kibana route.
openshift_logging_kibana_ops_hostname	Equivalent to openshift_logging_kibana_hostname for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_cpu_limit	Equivalent to openshift_logging_kibana_cpu_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_memory_limit	Equivalent to openshift_logging_kibana_memory_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_proxy_debug	Equivalent to openshift_logging_kibana_proxy_debug for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_proxy_cpu_limit	Equivalent to openshift_logging_kibana_proxy_cpu_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_proxy_memory_limit	Equivalent to openshift_logging_kibana_proxy_memory_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_kibana_ops_replica_count	Equivalent to openshift_logging_kibana_replica_count for Ops cluster when openshift_logging_use_ops is set to true .

Parameter	Description
openshift_logging_es_allow_external	Set to true to expose Elasticsearch as a reencrypt route. Set to false by default.
openshift_logging_es_hostname	<p>The external-facing hostname to use for the route and the TLS server certificate. The default is set to es.</p> <p>For example, if openshift_master_default_subdomain is set to =example.test, then the default value of openshift_logging_es_hostname will be es.example.test.</p>
openshift_logging_es_cert	The location of the certificate Elasticsearch uses for the external TLS server cert. The default is a generated cert.
openshift_logging_es_key	The location of the key Elasticsearch uses for the external TLS server cert. The default is a generated key.
openshift_logging_es_ca_ext	The location of the CA cert Elasticsearch uses for the external TLS server cert. The default is the internal CA.
openshift_logging_es_ops_allow_external	Set to true to expose Elasticsearch as a reencrypt route. Set to false by default.
openshift_logging_es_ops_hostname	<p>The external-facing hostname to use for the route and the TLS server certificate. The default is set to es-ops.</p> <p>For example, if openshift_master_default_subdomain is set to =example.test, then the default value of openshift_logging_es_ops_hostname will be es-ops.example.test.</p>
openshift_logging_es_ops_cert	The location of the certificate Elasticsearch uses for the external TLS server cert. The default is a generated cert.
openshift_logging_es_ops_key	The location of the key Elasticsearch uses for the external TLS server cert. The default is a generated key.
openshift_logging_es_ops_ca_ext	The location of the CA cert Elasticsearch uses for the external TLS server cert. The default is the internal CA.
openshift_logging_fluentd_nodeselector	<p>A node selector that specifies which nodes are eligible targets for deploying Fluentd instances. Any node where Fluentd should run (typically, all) must have this label before Fluentd is able to run and collect logs.</p> <p>When scaling up the Aggregated Logging cluster after installation, the openshift_logging role labels nodes provided by openshift_logging_fluentd_hosts with this node selector.</p> <p>As part of the installation, it is recommended that you add the Fluentd node selector label to the list of persisted node labels.</p>

Parameter	Description
openshift_logging_fluentd_cpu_limit	The CPU limit for Fluentd pods.
openshift_logging_fluentd_memory_limit	The memory limit for Fluentd pods.
openshift_logging_fluentd_journal_read_from_head	Set to true if Fluentd should read from the head of Journal when first starting up, using this may cause a delay in ES receiving current log records.
openshift_logging_fluentd_hosts	List of nodes that should be labeled for Fluentd to be deployed. The default is to label all nodes with ['--all']. The null value is openshift_logging_fluentd_hosts={} . To spin up Fluentd pods update the daemonset's nodeSelector to a valid label. For example, ['host1.example.com', 'host2.example.com'].
openshift_logging_fluentd_audit_container_engine	When openshift_logging_fluentd_audit_container_engine is set to true , the audit log of the container engine is collected and stored in ES. Enabling this variable allows the EFK to watch the specified audit log file or the default /var/log/audit.log file, collects audit information for the container engine for the platform, then puts it into Kibana.
openshift_logging_fluentd_audit_file	Location of audit log file. The default is /var/log/audit/audit.log . Enabling this variable allows the EFK to watch the specified audit log file or the default /var/log/audit.log file, collects audit information for the container engine for the platform, then puts it into Kibana.
openshift_logging_fluentd_audit_pos_file	Location of the Fluentd in_tail position file for the audit log file. The default is /var/log/audit/audit.log.pos . Enabling this variable allows the EFK to watch the specified audit log file or the default /var/log/audit.log file, collects audit information for the container engine for the platform, then puts it into Kibana.
openshift_logging_es_host	The name of the ES service where Fluentd should send logs.
openshift_logging_es_port	The port for the ES service where Fluentd should send logs.
openshift_logging_es_ca	The location of the CA Fluentd uses to communicate with openshift_logging_es_host .
openshift_logging_es_client_cert	The location of the client certificate Fluentd uses for openshift_logging_es_host .
openshift_logging_es_client_key	The location of the client key Fluentd uses for openshift_logging_es_host .

Parameter	Description
openshift_logging_es_cluster_size	Elasticsearch replicas to deploy. Redundancy requires at least three or more.
openshift_logging_es_cpu_limit	The amount of CPU limit for the ES cluster.
openshift_logging_es_memory_limit	Amount of RAM to reserve per Elasticsearch instance. It must be at least 512M. Possible suffixes are G,g,M,m.
openshift_logging_es_number_of_replicas	The number of replica shards per primary shard for every new index. Defaults to '0'. A minimum of 1 is advisable for production clusters.
openshift_logging_es_number_of_shards	The number of primary shards for every new index created in ES. Defaults to '1'.
openshift_logging_es_pv_selector	A key/value map added to a PVC in order to select specific PVs.
openshift_logging_es_pvc_dynamic	To dynamically provision the backing storage, set the parameter value to true . When set to true , the storageClass spec is omitted from the PVC definition. If you set a openshift_logging_es_pvc_storage_class_name parameter value, its value overrides the value of the the openshift_logging_es_pvc_dynamic parameter.
openshift_logging_es_pvc_storage_class_name	To use a non-default storage class, specify the storage class name, such as glusterprovisioner or cephrbdprovisioner . After you specify the storage class name, dynamic volume provisioning is active regardless of the openshift_logging_es_pvc_dynamic value.
openshift_logging_es_pvc_size	Size of the persistent volume claim to create per Elasticsearch instance. For example, 100G. If omitted, no PVCs are created and ephemeral volumes are used instead.
openshift_logging_es_pvc_prefix	<p>Prefix for the names of persistent volume claims to be used as storage for Elasticsearch instances. A number is appended per instance, such as logging-es-1. If they do not already exist, they are created with size es-pvc-size.</p> <p>When openshift_logging_es_pvc_prefix is set, and:</p> <ul style="list-style-type: none"> • openshift_logging_es_pvc_dynamic=true, the value for openshift_logging_es_pvc_size is optional. • openshift_logging_es_pvc_dynamic=false, the value for openshift_logging_es_pvc_size must be set.

Parameter	Description
openshift_logging_es_recover_after_time	The amount of time ES will wait before it tries to recover.
openshift_logging_es_storage_group	Number of a supplemental group ID for access to Elasticsearch storage volumes. Backing volumes should allow access by this group ID.
openshift_logging_es_nodeselector	A node selector specified as a map that determines which nodes are eligible targets for deploying Elasticsearch instances. This can be used to place these instances on nodes reserved or optimized for running them. For example, the selector could be {"node-type": "infrastructure"} . At least one active node must have this label before Elasticsearch will deploy.
openshift_logging_es_ops_host	Equivalent to openshift_logging_es_host for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_port	Equivalent to openshift_logging_es_port for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_ca	Equivalent to openshift_logging_es_ca for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_client_cert	Equivalent to openshift_logging_es_client_cert for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_client_key	Equivalent to openshift_logging_es_client_key for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_cluster_size	Equivalent to openshift_logging_es_cluster_size for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_cpu_limit	Equivalent to openshift_logging_es_cpu_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_memory_limit	Equivalent to openshift_logging_es_memory_limit for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_pv_selector	Equivalent to openshift_logging_es_pv_selector for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_pvc_dynamic	Equivalent to openshift_logging_es_pvc_dynamic for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_pvc_size	Equivalent to openshift_logging_es_pvc_size for Ops cluster when openshift_logging_use_ops is set to true .
openshift_logging_es_ops_pvc_prefix	Equivalent to openshift_logging_es_pvc_prefix for Ops cluster when openshift_logging_use_ops is set to true .

Parameter	Description
<code>openshift_logging_es_ops_recover_after_time</code>	Equivalent to <code>openshift_logging_es_recovery_after_time</code> for Ops cluster when <code>openshift_logging_use_ops</code> is set to <code>true</code> .
<code>openshift_logging_es_ops_storage_group</code>	Equivalent to <code>openshift_logging_es_storage_group</code> for Ops cluster when <code>openshift_logging_use_ops</code> is set to <code>true</code> .
<code>openshift_logging_es_ops_nodeselector</code>	A node selector that specifies which nodes are eligible targets for deploying Elasticsearch instances. This can be used to place these instances on nodes reserved or optimized for running them. For example, the selector could be <code>node-type=infrastructure</code> . At least one active node must have this label before Elasticsearch will deploy.
<code>openshift_logging_elasticsearch_kibana_index_mode</code>	<p>The default value, <code>unique</code>, allows users to each have their own Kibana index. In this mode, their saved queries, visualizations, and dashboards are not shared.</p> <p>You may also set the value <code>shared_ops</code>. In this mode, all operations users share a Kibana index which allows each operations user to see the same queries, visualizations, and dashboards. To determine if you are an operations user:</p> <pre>#oc auth can-i view pod/logs -n default yes</pre> <p>If you do not have appropriate access, contact your cluster administrator.</p>
<code>openshift_logging_kibana_ops_nodeselector</code>	A node selector that specifies which nodes are eligible targets for deploying Kibana instances.
<code>openshift_logging_curator_ops_nodeselector</code>	A node selector that specifies which nodes are eligible targets for deploying Curator instances.

Custom Certificates

You can specify custom certificates using the following inventory variables instead of relying on those generated during the deployment process. These certificates are used to encrypt and secure communication between a user's browser and Kibana. The security-related files will be generated if they are not supplied.

File Name	Description
openshift_logging_kibana_cert	A browser-facing certificate for the Kibana server.
openshift_logging_kibana_key	A key to be used with the browser-facing Kibana certificate.
openshift_logging_kibana_ca	The absolute path on the control node to the CA file to use for the browser facing Kibana certs.
openshift_logging_kibana_ops_cert	A browser-facing certificate for the Ops Kibana server.
openshift_logging_kibana_ops_key	A key to be used with the browser-facing Ops Kibana certificate.
openshift_logging_kibana_ops_ca	The absolute path on the control node to the CA file to use for the browser facing ops Kibana certs.

33.4. DEPLOYING THE EFK STACK

The EFK stack is deployed using an Ansible playbook to the EFK components. Run the playbook from the default OpenShift Ansible location using the default [inventory](#) file.

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    logging/config.yml
```

Running the playbook deploys all resources needed to support the stack; such as Secrets, ServiceAccounts, and DeploymentConfigs. The playbook waits to deploy the component pods until the stack is running. If the wait steps fail, the deployment could still be successful; it may be retrieving the component images from the registry which can take up to a few minutes. You can watch the process with:

```
$ oc get pods -w
```

They will eventually enter **Running** status. For additional details about the status of the pods during deployment by retrieving associated events:

```
$ oc describe pods/<pod_name>
```

Check the logs if the pods do not run successfully:

```
$ oc logs -f <pod_name>
```

33.5. UNDERSTANDING AND ADJUSTING THE DEPLOYMENT

This section describes adjustments that you can make to deployed components.

33.5.1. Ops Cluster



NOTE

The logs for the **default**, **openshift**, and **openshift-infra** projects are automatically aggregated and grouped into the **.operations** item in the Kibana interface.

The project where you have deployed the EFK stack (**logging**, as documented here) is *not* aggregated into **.operations** and is found under its ID.

If you set **openshift_logging_use_ops** to **true** in your inventory file, Fluentd is configured to split logs between the main Elasticsearch cluster and another cluster reserved for operations logs, which are defined as node system logs and the projects **default**, **openshift**, and **openshift-infra**. Therefore, a separate Elasticsearch cluster, a separate Kibana, and a separate Curator are deployed to index, access, and manage operations logs. These deployments are set apart with names that include **-ops**. Keep these separate deployments in mind if you enable this option. Most of the following discussion also applies to the operations cluster if present, just with the names changed to include **-ops**.

33.5.2. Elasticsearch

A highly-available environment requires at least three replicas of Elasticsearch; each on a different host. Elasticsearch replicas require their own storage, but an OpenShift Container Platform deployment configuration shares storage volumes between all its pods. So, when scaled up, the EFK deployer ensures each replica of Elasticsearch has its own deployment configuration.

It is possible to scale your cluster up after creation by modifying the **openshift_logging_es_cluster_size** in the inventory file and re-running the logging playbook. Additional clustering parameters can be modified and are described in [Specifying Logging Ansible Variables](#).

Refer to [Elastic's documentation](#) for considerations involved in choosing storage and network location as directed below.

Viewing all Elasticsearch Deployments

To view all current Elasticsearch deployments:

```
$ oc get dc --selector logging-infra=elasticsearch
```

Node Selector

Because Elasticsearch can use a lot of resources, all members of a cluster should have low latency network connections to each other and to any remote storage. Ensure this by directing the instances to dedicated nodes, or a dedicated region within your cluster, using a [node selector](#).

To configure a node selector, specify the **openshift_logging_es_nodeselector** configuration option in the inventory file. This applies to all Elasticsearch deployments; if you need to individualize the node selectors, you must manually edit each deployment configuration after deployment. The node selector is specified as a python compatible dict. For example, **{"node-type": "infra", "region": "east"}**.

Persistent Elasticsearch Storage

By default, the **openshift_logging** Ansible role creates an ephemeral deployment in which all of a

pod's data is lost upon restart. For production usage, specify a persistent storage volume for each Elasticsearch deployment configuration. You can create the necessary [persistent volume claims](#) before deploying or have them created for you. The PVCs must be named to match the `openshift_logging_es_pvc_prefix` setting, which defaults to `logging-es`; each PVC name will have a sequence number added to it: `logging-es-0`, `logging-es-1`, `logging-es-2`, and so on. If a PVC needed for the deployment exists already, it is used; if not, and `openshift_logging_es_pvc_size` has been specified, it is created with a request for that size.



WARNING

Using NFS storage as a volume or a persistent volume (or via NAS such as Gluster) is not supported for Elasticsearch storage, as Lucene relies on file system behavior that NFS does not supply. Data corruption and other problems can occur. If NFS storage is a requirement, you can allocate a large file on a volume to serve as a storage device and mount it locally on one host. For example, if your NFS storage volume is mounted at `/nfs/storage`:

```
$ truncate -s 1T /nfs/storage/elasticsearch-1
$ mkfs.xfs /nfs/storage/elasticsearch-1
$ mount -o loop /nfs/storage/elasticsearch-1 /usr/local/es-storage
$ chown 1000:1000 /usr/local/es-storage
```

Then, use `/usr/local/es-storage` as a host-mount as described below. Use a different backing file as storage for each Elasticsearch replica.

This loopback must be maintained manually outside of OpenShift Container Platform, on the node. You must not maintain it from inside a container.

It is possible to use a local disk volume (if available) on each node host as storage for an Elasticsearch replica. Doing so requires some preparation as follows.

1. The relevant service account must be given the privilege to mount and edit a local volume:

```
$ oc adm policy add-scc-to-user privileged \
    system:serviceaccount:logging:aggregated-logging-
    elasticsearch 1
```

- 1** Use the project you created earlier (for example, `logging`) when running the logging playbook.

2. Each Elasticsearch replica definition must be patched to claim that privilege, for example (change to `--selector component=es-ops` for Ops cluster):

```
$ for dc in $(oc get deploymentconfig --selector component=es -o
name); do
    oc scale $dc --replicas=0
    oc patch $dc \
        -p '{"spec":{"template":{"spec":{"containers":
```

```
[{"name":"elasticsearch","securityContext":{"privileged":
true}}]}]}]}'
done
```

3. The Elasticsearch replicas must be located on the correct nodes to use the local storage, and should not move around even if those nodes are taken down for a period of time. This requires giving each Elasticsearch replica a node selector that is unique to a node where an administrator has allocated storage for it. To configure a node selector, edit each Elasticsearch deployment configuration and add or edit the **nodeSelector** section to specify a unique label that you have applied for each desired node:

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1" 1
```

- 1** This label should uniquely identify a replica with a single node that bears that label, in this case **logging-es-node=1**. Use the **oc label** command to apply labels to nodes as needed.

To automate applying the node selector you can instead use the **oc patch** command:

```
$ oc patch dc/logging-es-<suffix> \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-
node":"1"}}}}}'
```

4. Once these steps are taken, a local host mount can be applied to each replica as in this example (where we assume storage is mounted at the same path on each node) (change to **--selector component=es-ops** for Ops cluster):

```
$ for dc in $(oc get deploymentconfig --selector component=es -o
name); do
  oc set volume $dc \
    --add --overwrite --name=elasticsearch-storage \
    --type=hostPath --path=/usr/local/es-storage
  oc rollout latest $dc
  oc scale $dc --replicas=1
done
```

Changing the Scale of Elasticsearch

If you need to scale up the number of Elasticsearch instances your cluster uses, it is not as simple as scaling up an Elasticsearch deployment configuration. This is due to the nature of persistent volumes and how Elasticsearch is configured to store its data and recover the cluster. Instead, scaling up requires creating a deployment configuration for each Elasticsearch cluster node.

The simplest way to change the scale of Elasticsearch is to modify the inventory host file and re-run the logging playbook as described previously. Assuming you have supplied persistent storage for the deployment, this should not be disruptive.



NOTE

Resizing an Elasticsearch cluster using the logging playbook is only possible when the new **openshift_logging_es_cluster_size** value is higher than the current number of Elasticsearch nodes (scaled up) in the cluster.

If you do not wish to reinstall, for instance because you have made customizations that you would like to preserve, then it is possible to add new Elasticsearch deployment configurations to the cluster using a template supplied by the deployer. This requires a more complicated procedure however.

Expose Elasticsearch as a Route

By default, Elasticsearch deployed with OpenShift aggregated logging is not accessible from outside the logging cluster. You can enable a route for external access to Elasticsearch for those tools that want to access its data.

You have access to Elasticsearch using your OpenShift token, and you can provide the external Elasticsearch and Elasticsearch Ops hostnames when creating the server certificate (similar to Kibana).

1. To access Elasticsearch as a reencrypt route, define the following variables:

```
openshift_logging_es_allow_external=True
openshift_logging_es_hostname=elasticsearch.example.com
```

2. Run the following Ansible playbook:

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    logging/config.yml
```

3. To log in to Elasticsearch remotely, the request must contain three HTTP headers:

```
Authorization: Bearer $token
X-Proxy-Remote-User: $username
X-Forwarded-For: $ip_address
```

4. You must have access to the project in order to be able to access to the logs. For example:

```
$ oc login <user1>
$ oc new-project <user1project>
$ oc new-app <httpd-example>
```

5. You need to get the token of this ServiceAccount to be used in the request:

```
$ token=$(oc whoami -t)
```

6. Using the token previously configured, you should be able access Elasticsearch through the exposed route:

```
$ curl -k -H "Authorization: Bearer $token" -H "X-Proxy-Remote-User:
$(oc whoami)" -H "X-Forwarded-For: 127.0.0.1"
https://es.example.test/project.my-project.*/_search?q=level:err |
python -mjson.tool
```

33.5.3. Fluentd

Fluentd is deployed as a DaemonSet that deploys replicas according to a node label selector, which you can specify with the inventory parameter **openshift_logging_fluentd_nodeselector** and the default is **logging-infra-fluentd**. As part of the OpenShift cluster installation, it is recommended that you add the Fluentd node selector to the list of persisted [node labels](#).

Fluentd uses **journald** as the system log source. These are log messages from the operating system, the container runtime, and OpenShift.

The available container runtimes provide minimal information to identify the source of log messages. Log collection and normalization of logs can occur after a pod is deleted and additional metadata cannot be retrieved from the API server, such as labels or annotations.

If a pod with a given name and namespace is deleted before the log collector finishes processing logs, there might not be a way to distinguish the log messages from a similarly named pod and namespace. This can cause logs to be indexed and annotated to an index that is not owned by the user who deployed the pod.



IMPORTANT

The available container runtimes provide minimal information to identify the source of log messages and do not guarantee unique individual log messages or that these messages can be traced to their source.

Clean installations of OpenShift Container Platform 3.9 use **json-file** as the default log driver, but environments upgraded from OpenShift Container Platform 3.7 will maintain their existing **journald** log driver configuration. It is recommended to use the **json-file** log driver. See [Changing the Aggregated Logging Driver](#) for instructions to change your existing log driver configuration to **json-file**.

Configuring Fluentd to Send Logs to an External Log Aggregator

You can configure Fluentd to send a copy of its logs to an external log aggregator, and not the default Elasticsearch, using the **secure-forward** plug-in. From there, you can further process log records after the locally hosted Fluentd has processed them.

The logging deployment provides a **secure-forward.conf** section in the Fluentd configmap for configuring the external aggregator:

```
<store>
@type secure_forward
self_hostname pod-${HOSTNAME}
shared_key thisisasharedkey
secure yes
enable_strict_verification yes
ca_cert_path /etc/fluent/keys/your_ca_cert
ca_private_key_path /etc/fluent/keys/your_private_key
ca_private_key_passphrase passphrase
<server>
  host ose1.example.com
  port 24284
</server>
<server>
  host ose2.example.com
  port 24284
```



```

    standby
  </server>
</server>
  host ose3.example.com
  port 24284
  standby
</server>
</store>

```

This can be updated using the **oc edit** command:

```
$ oc edit configmap/logging-fluentd
```

Certificates to be used in **secure-forward.conf** can be added to the existing secret that is mounted on the Fluentd pods. The **your_ca_cert** and **your_private_key** values must match what is specified in **secure-forward.conf** in **configmap/logging-fluentd**:

```

$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_ca_cert','value':'$(base64
/path/to/your_ca_cert.pem)'}]"
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_private_key','value':'$(base64
/path/to/your_private_key.pem)'}]"

```



NOTE

Replace **your_private_key** with a generic name. This is a link to the JSON path, not a path on your host system.

When configuring the external aggregator, it must be able to accept messages securely from Fluentd.

If the external aggregator is another Fluentd server, it must have the **fluent-plugin-secure-forward** plug-in installed and make use of the input plug-in it provides:

```

<source>
  @type secure_forward

  self_hostname ${HOSTNAME}
  bind 0.0.0.0
  port 24284

  shared_key thisisasharedkey

  secure yes
  cert_path      /path/for/certificate/cert.pem
  private_key_path /path/for/certificate/key.pem
  private_key_passphrase secret_foo_bar_baz
</source>

```

You can find further explanation of how to set up the **fluent-plugin-secure-forward** plug-in in the [fluent-plugin-secure-forward repository](#).

Reducing the Number of Connections from Fluentd to the API Server



IMPORTANT

mux is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

mux is a Secure Forward listener service.

Parameter	Description
openshift_logging_use_mux	The default is set to False . If set to True , a service called mux is deployed. This service acts as a Fluentd secure_forward aggregator for the node agent Fluentd daemonsets running in the cluster. Use openshift_logging_use_mux to reduce the number of connections to the OpenShift API server, and configure each node in Fluentd to send raw logs to mux and turn off the Kubernetes metadata plug-in. This requires the use of openshift_logging_mux_client_mode .
openshift_logging_mux_client_mode	Values for openshift_logging_mux_client_mode are minimal and maximal , and there is no default. openshift_logging_mux_client_mode causes the Fluentd node agent to send logs to mux rather than directly to Elasticsearch. The value maximal means that Fluentd does as much processing as possible at the node before sending the records to mux . The maximal value is recommended for using mux . The value minimal means that Fluentd does no processing at all, and sends the raw logs to mux for processing. It is not recommended to use the minimal value.
openshift_logging_mux_allow_external	The default is set to False . If set to True , the mux service is deployed, and it is configured to allow Fluentd clients running outside of the cluster to send logs using secure_forward . This allows OpenShift logging to be used as a central logging service for clients other than OpenShift, or other OpenShift clusters.
openshift_logging_mux_hostname	The default is mux plus openshift_master_default_subdomain . This is the hostname external_clients will use to connect to mux , and is used in the TLS server cert subject.
openshift_logging_mux_port	24284
openshift_logging_mux_cpu_limit	500M
openshift_logging_mux_memory_limit	1Gi

Parameter	Description
openshift_logging_mux_default_namespaces	The default is mux-undefined . The first value in the list is the namespace to use for undefined projects, followed by any additional namespaces to create by default. Usually, you do not need to set this value.
openshift_logging_mux_namespaces	The default value is empty, allowing for additional namespaces to create for external mux clients to associate with their logs. You will need to set this value.

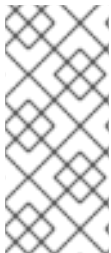
Throttling logs in Fluentd

For projects that are especially verbose, an administrator can throttle down the rate at which the logs are read in by Fluentd before being processed.



WARNING

Throttling can contribute to log aggregation falling behind for the configured projects; log entries can be lost if a pod is deleted before Fluentd catches up.



NOTE

Throttling does not work when using the systemd journal as the log source. The throttling implementation depends on being able to throttle the reading of the individual log files for each project. When reading from the journal, there is only a single log source, no log files, so no file-based throttling is available. There is not a method of restricting the log entries that are read into the Fluentd process.

To tell Fluentd which projects it should be restricting, edit the throttle configuration in its ConfigMap after deployment:

```
$ oc edit configmap/logging-fluentd
```

The format of the **throttle-config.yaml** key is a YAML file that contains project names and the desired rate at which logs are read in on each node. The default is 1000 lines at a time per node. For example:

- Projects

```
project-name:
  read_lines_limit: 50

second-project-name:
  read_lines_limit: 100
```

- Logging

```
logging:
```

```
    read_lines_limit: 500

test-project:
  read_lines_limit: 10

.operations:
  read_lines_limit: 100
```

When you make changes to any part of the EFK stack, specifically Elasticsearch or Fluentd, you should first scale Elasticsearch down to zero and scale Fluentd so it does not match any other nodes. Then, make the changes and scale Elasticsearch and Fluentd back.

To scale Elasticsearch to zero:

```
$ oc scale --replicas=0 dc/<ELASTICSEARCH_DC>
```

Change nodeSelector in the daemonset configuration to match zero:

Get the fluentd node selector:

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
nodeSelector:
  logging-infra-fluentd: "true"
```

Use the `oc patch` command to modify the daemonset nodeSelector:

```
$ oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":{"nonexistlabel":"true"}}}}}'
```

Get the fluentd node selector:

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
nodeSelector:
  "nonexistlabel: "true"
```

Scale Elasticsearch back up from zero:

```
$ oc scale --replicas=# dc/<ELASTICSEARCH_DC>
```

Change nodeSelector in the daemonset configuration back to logging-infra-fluentd: "true".

Use the **oc patch** command to modify the daemonset nodeSelector:

```
oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-infra-fluentd":"true"}}}}}'
```

33.5.4. Kibana

To access the Kibana console from the OpenShift Container Platform web console, add the **loggingPublicURL** parameter in the [master webconsole-config configmap file](#), with the URL of the Kibana console (the **kibana-hostname** parameter). The value must be an HTTPS URL:

```
...
clusterInfo:
  ...
  loggingPublicURL: "https://kibana.example.com"
  ...
```

Setting the **loggingPublicURL** parameter creates a **View Archive** button on the OpenShift Container Platform web console under the **Browse** → **Pods** → **<pod_name>** → **Logs** tab. This links to the Kibana console.

You can scale the Kibana deployment as usual for redundancy:

```
$ oc scale dc/logging-kibana --replicas=2
```



NOTE

To ensure the scale persists across multiple executions of the logging playbook, make sure to update the **openshift_logging_kibana_replica_count** in the inventory file.

You can see the user interface by visiting the site specified by the **openshift_logging_kibana_hostname** variable.

See the [Kibana documentation](#) for more information on Kibana.

Kibana Visualize

Kibana Visualize enables you to create visualizations and dashboards for monitoring container and pod logs allows administrator users (**cluster-admin** or **cluster-reader**) to view logs by deployment, namespace, pod, and container.

Kibana Visualize exists inside the Elasticsearch and ES-OPS pod, and must be run inside those pods. To load dashboards and other Kibana UI objects, you must first log into Kibana as the user you want to add the dashboards to, then log out. This will create the necessary per-user configuration that the next step relies on. Then, run:

```
$ oc exec <$espod> -- es_load_kibana_ui_objects <user-name>
```

Where **\$espod** is the name of any one of your Elasticsearch pods.

33.5.5. Curator

Curator allows administrators to configure scheduled Elasticsearch maintenance operations to be performed automatically on a per-project basis. It is scheduled to perform actions daily based on its configuration. Only one Curator pod is recommended per Elasticsearch cluster. Curator is configured via a YAML configuration file with the following structure:

```
$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE

$PROJECT_NAME:
```

```
$ACTION:
$UNIT: $VALUE
...
```

The available parameters are:

Variable Name	Description
PROJECT_NAME	The actual name of a project, such as myapp-devel . For OpenShift Container Platform operations logs, use the name .operations as the project name.
ACTION	The action to take, currently only delete is allowed.
UNIT	One of days , weeks , or months .
VALUE	An integer for the number of units.
.defaults	Use .defaults as the \$PROJECT_NAME to set the defaults for projects that are not specified.
runhour	(Number) the hour of the day in 24-hour format at which to run the Curator jobs. For use with .defaults .
runminute	(Number) the minute of the hour at which to run the Curator jobs. For use with .defaults .
timezone	(String) the tring in tzselect(8) or timedatectl(1) format. The default timezone is UTC.
.regex	The list of regular expressions that match project names.
pattern	The valid and properly escaped regular expression pattern enclosed by single quotation marks.

For example, to configure Curator to:

- delete indices in the **myapp-dev** project older than **1 day**
- delete indices in the **myapp-qa** project older than **1 week**
- delete **operations** logs older than **8 weeks**
- delete all other projects indices after they are **31 days** old
- run the Curator jobs at midnight every day

Use:

```
config.yaml: |
  myapp-dev:
```

```

    delete:
      days: 1

myapp-qe:
  delete:
    weeks: 1

.operations:
  delete:
    weeks: 8

.defaults:
  delete:
    days: 31
  runhour: 0
  runminute: 0
  timezone: America/New_York

.regex:
- pattern: '^project\..+\-dev\..*$'
  delete:
    days: 15
- pattern: '^project\..+\-test\..*$'
  delete:
    days: 30

```

IMPORTANT

When you use **month** as the **\$UNIT** for an operation, Curator starts counting at the first day of the current month, not the current day of the current month. For example, if today is April 15, and you want to delete indices that are 2 months older than today (`delete: months: 2`), Curator does not delete indices that are dated older than February 15; it deletes indices older than February 1. That is, it goes back to the first day of the current month, then goes back two whole months from that date. If you want to be exact with Curator, it is best to use days (for example, **`delete: days: 30`**).

33.5.5.1. Creating the Curator Configuration

The **openshift_logging** Ansible role provides a ConfigMap from which Curator reads its configuration. You may edit or replace this ConfigMap to reconfigure Curator. Currently the **logging-curator** ConfigMap is used to configure both your ops and non-ops Curator instances. Any **.operations** configurations are in the same location as your application logs configurations.

1. To edit the provided ConfigMap to configure your Curator instances:

```
$ oc edit configmap/logging-curator
```

2. To replace the provided ConfigMap instead:

```

$ create /path/to/mycuratorconfig.yaml
$ oc create configmap logging-curator -o yaml \
  --from-file=config.yaml=/path/to/mycuratorconfig.yaml | \
  oc replace -f -

```

3. After you make your changes, redeploy Curator:

```
$ oc rollout latest dc/logging-curator
$ oc rollout latest dc/logging-curator-ops
```

33.6. CLEANUP

Remove everything generated during the deployment.

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml \
    -e openshift_logging_install_logging=False
```

33.7. TROUBLESHOOTING KIBANA

Using the Kibana console with OpenShift Container Platform can cause problems that are easily solved, but are not accompanied with useful error messages. Check the following troubleshooting sections if you are experiencing any problems when deploying Kibana on OpenShift Container Platform:

Login Loop

The OAuth2 proxy on the Kibana console must share a secret with the master host's OAuth2 server. If the secret is not identical on both servers, it can cause a login loop where you are continuously redirected back to the Kibana login page.

To fix this issue, delete the current OAuthClient, and use **openshift-ansible** to re-run the **openshift_logging** role:

```
$ oc delete oauthclient/kibana-proxy
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```

Cryptic Error When Viewing the Console

When attempting to visit the Kibana console, you may receive a browser error instead:

```
{"error":"invalid_request","error_description":"The request is missing a
required parameter,
includes an invalid parameter value, includes a parameter more than once,
or is otherwise malformed."}
```

This can be caused by a mismatch between the OAuth2 client and server. The return address for the client must be in a whitelist so the server can securely redirect back after logging in.

Fix this issue by replacing the OAuthClient entry:

```
$ oc delete oauthclient/kibana-proxy
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```


If the problem persists, check that you are accessing Kibana at a URL listed in the OAuth client. This issue can be caused by accessing the URL at a forwarded port, such as 1443 instead of the standard 443 HTTPS port. You can adjust the server whitelist by editing the OAuth client:

```
$ oc edit oauthclient/kibana-proxy
```

503 Error When Viewing the Console

If you receive a proxy error when viewing the Kibana console, it could be caused by one of two issues.

First, Kibana may not be recognizing pods. If Elasticsearch is slow in starting up, Kibana may timeout trying to reach it. Check whether the relevant service has any endpoints:

```
$ oc describe service logging-kibana
Name:                logging-kibana
[...]
Endpoints:           <none>
```

If any Kibana pods are live, endpoints are listed. If they are not, check the state of the Kibana pods and deployment. You may need to scale the deployment down and back up again.

The second possible issue may be caused if the route for accessing the Kibana service is masked. This can happen if you perform a test deployment in one project, then deploy in a different project without completely removing the first deployment. When multiple routes are sent to the same destination, the default router will only route to the first created. Check the problematic route to see if it is defined in multiple places:

```
$ oc get route --all-namespaces --selector logging-infra=support
```

F-5 Load Balancer and X-Forwarded-For Enabled

If you are attempting to use a F-5 load balancer in front of Kibana with **X-Forwarded-For** enabled, this can cause an issue in which the Elasticsearch **Searchguard** plug-in is unable to correctly accept connections from Kibana.

Example Kibana Error Message

```
Kibana: Unknown error while connecting to Elasticsearch

Error: Unknown error while connecting to Elasticsearch
Error: UnknownHostException[No trusted proxies]
```

To configure Searchguard to ignore the extra header:

1. Scale down all Fluentd pods.
2. Scale down Elasticsearch after the Fluentd pods have terminated.
3. Add **searchguard.http.xforwardedfor.header: DUMMY** to the Elasticsearch configuration section.

```
$ oc edit configmap/logging-elasticsearch 1
```

1 This approach requires that Elasticsearch's configurations are within a ConfigMap.

4. Scale Elasticsearch back up.
5. Scale up all Fluentd pods.

33.8. SENDING LOGS TO AN EXTERNAL ELASTICSEARCH INSTANCE

Fluentd sends logs to the value of the **ES_HOST**, **ES_PORT**, **OPS_HOST**, and **OPS_PORT** environment variables of the Elasticsearch deployment configuration. The application logs are directed to the **ES_HOST** destination, and operations logs to **OPS_HOST**.



NOTE

Sending logs directly to an AWS Elasticsearch instance is not supported. Use [Fluentd Secure Forward](#) to direct logs to an instance of Fluentd that you control and that is configured with the **fluent-plugin-aws-elasticsearch-service** plug-in.

To direct logs to a specific Elasticsearch instance, edit the deployment configuration and replace the value of the above variables with the desired instance:

```
$ oc edit dc/<deployment_configuration>
```

For an external Elasticsearch instance to contain both application and operations logs, you can set **ES_HOST** and **OPS_HOST** to the same destination, while ensuring that **ES_PORT** and **OPS_PORT** also have the same value.

If your externally hosted Elasticsearch instance does not use TLS, update the **_CLIENT_CERT**, **_CLIENT_KEY**, and **_CA** variables to be empty. If it does use TLS, but not mutual TLS, update the **_CLIENT_CERT** and **_CLIENT_KEY** variables to be empty and patch or recreate the **logging-fluentd** secret with the appropriate **_CA** value for communicating with your Elasticsearch instance. If it uses Mutual TLS as the provided Elasticsearch instance does, patch or recreate the **logging-fluentd** secret with your client key, client cert, and CA.



NOTE

If you are not using the provided Kibana and Elasticsearch images, you will not have the same multi-tenant capabilities and your data will not be restricted by user access to a particular project.

33.9. SENDING LOGS TO AN EXTERNAL SYSLOG SERVER

Use the **fluent-plugin-remote-syslog** plug-in on the host to send logs to an external syslog server.

Set environment variables in the **logging-fluentd** or **logging-mux** deployment configurations:

- name: REMOTE_SYSLOG_HOST 1
value: host1
- name: REMOTE_SYSLOG_HOST_BACKUP
value: host2
- name: REMOTE_SYSLOG_PORT_BACKUP
value: 5555

- 1 The desired remote syslog host. Required for each host.

This will build two destinations. The syslog server on **host1** will be receiving messages on the default port of **514**, while **host2** will be receiving the same messages on port **5555**.

Alternatively, you can configure your own custom *fluent.conf* in the **logging-fluentd** or **logging-mux** ConfigMaps.

Fluentd Environment Variables

Parameter	Description
USE_REMOTE_SYSLOG	Defaults to false . Set to true to enable use of the fluent-plugin-remote-syslog gem
REMOTE_SYSLOG_HOST	(Required) Hostname or IP address of the remote syslog server.
REMOTE_SYSLOG_PORT	Port number to connect on. Defaults to 514 .
REMOTE_SYSLOG_SEVERITY	Set the syslog severity level. Defaults to debug .
REMOTE_SYSLOG_FACILITY	Set the syslog facility. Defaults to local0 .
REMOTE_SYSLOG_USE_RECORD	Defaults to false . Set to true to use the record's severity and facility fields to set on the syslog message.
REMOTE_SYSLOG_REMOVE_TAG_PREFIX	Removes the prefix from the tag, defaults to ' ' (empty).
REMOTE_SYSLOG_TAG_KEY	If specified, uses this field as the key to look on the record, to set the tag on the syslog message.
REMOTE_SYSLOG_PAYLOAD_KEY	If specified, uses this field as the key to look on the record, to set the payload on the syslog message.



WARNING

This implementation is insecure, and should only be used in environments where you can guarantee no snooping on the connection.

Fluentd Logging Ansible Variables

Parameter	Description
<code>openshift_logging_fluentd_remote_syslog</code>	The default is set to false . Set to true to enable use of the fluent-plugin-remote-syslog gem.
<code>openshift_logging_fluentd_remote_syslog_host</code>	Hostname or IP address of the remote syslog server, this is mandatory.
<code>openshift_logging_fluentd_remote_syslog_port</code>	Port number to connect on, defaults to 514 .
<code>openshift_logging_fluentd_remote_syslog_severity</code>	Set the syslog severity level, defaults to debug .
<code>openshift_logging_fluentd_remote_syslog_facility</code>	Set the syslog facility, defaults to local0 .
<code>openshift_logging_fluentd_remote_syslog_use_record</code>	The default is set to false . Set to true to use the record's severity and facility fields to set on the syslog message.
<code>openshift_logging_fluentd_remote_syslog_remove_tag_prefix</code>	Removes the prefix from the tag, defaults to ' ' (empty).
<code>openshift_logging_fluentd_remote_syslog_tag_key</code>	If string is specified, uses this field as the key to look on the record, to set the tag on the syslog message.
<code>openshift_logging_fluentd_remote_syslog_payload_key</code>	If string is specified, uses this field as the key to look on the record, to set the payload on the syslog message.

Mux Logging Ansible Variables

Parameter	Description
<code>openshift_logging_mux_remote_syslog</code>	The default is set to false . Set to true to enable use of the fluent-plugin-remote-syslog gem.
<code>openshift_logging_mux_remote_syslog_host</code>	Hostname or IP address of the remote syslog server, this is mandatory.
<code>openshift_logging_mux_remote_syslog_port</code>	Port number to connect on, defaults to 514 .

Parameter	Description
<code>openshift_logging_mux_remote_syslog_severity</code>	Set the syslog severity level, defaults to debug .
<code>openshift_logging_mux_remote_syslog_facility</code>	Set the syslog facility, defaults to local0 .
<code>openshift_logging_mux_remote_syslog_use_record</code>	The default is set to false . Set to true to use the record's severity and facility fields to set on the syslog message.
<code>openshift_logging_mux_remote_syslog_remove_tag_prefix</code>	Removes the prefix from the tag, defaults to ' ' (empty).
<code>openshift_logging_mux_remote_syslog_tag_key</code>	If string is specified, uses this field as the key to look on the record, to set the tag on the syslog message.
<code>openshift_logging_mux_remote_syslog_payload_key</code>	If string is specified, uses this field as the key to look on the record, to set the payload on the syslog message.

33.10. PERFORMING ADMINISTRATIVE ELASTICSEARCH OPERATIONS

As of logging version 3.2.0, an administrator certificate, key, and CA that can be used to communicate with and perform administrative operations on Elasticsearch are provided within the **logging-elasticsearch** secret.



NOTE

To confirm whether or not your EFK installation provides these, run:

```
$ oc describe secret logging-elasticsearch
```

1. Connect to an Elasticsearch pod that is in the cluster on which you are attempting to perform maintenance.
2. To find a pod in a cluster use either:

```
$ oc get pods -l component=es -o name | head -1
$ oc get pods -l component=es-ops -o name | head -1
```

3. Connect to a pod:

```
$ oc rsh <your_Elasticsearch_pod>
```

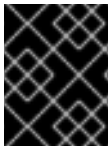
4. Once connected to an Elasticsearch container, you can use the certificates mounted from the secret to communicate with Elasticsearch per its [Indices APIs documentation](#).
Fluentd sends its logs to Elasticsearch using the index format **project.{project_name}.
{project_uuid}.YYYY.MM.DD** where YYYY.MM.DD is the date of the log record.

For example, to delete all logs for the **logging** project with uuid **3b3594fa-2ccd-11e6-acb7-0eb6b35eae3** from June 15, 2016, we can run:

```
$ curl --key /etc/elasticsearch/secret/admin-key \
  --cert /etc/elasticsearch/secret/admin-cert \
  --cacert /etc/elasticsearch/secret/admin-ca -XDELETE \
  "https://localhost:9200/project.logging.3b3594fa-2ccd-11e6-acb7-0eb6b35eae3.2016.06.15"
```

33.11. CHANGING THE AGGREGATED LOGGING DRIVER

For aggregated logging, it is recommended to use the **json-file** log driver.



IMPORTANT

When using the **json-file** driver, ensure that you are using Docker version **docker-1.12.6-55.gitc4618fb.el7_4 now** or later.

Fluentd determines the driver Docker is using by checking the **/etc/docker/daemon.json** and **/etc/sysconfig/docker** files.

You can determine which driver Docker is using with the **docker info** command:

```
# docker info | grep Logging

Logging Driver: journald
```

To change to **json-file**:

1. Modify either the **/etc/sysconfig/docker** or **/etc/docker/daemon.json** files.
For example:

```
# cat /etc/sysconfig/docker
OPTIONS=' --selinux-enabled --log-driver=json-file --log-opt max-size=1M --log-opt max-file=3 --signature-verification=False'

cat /etc/docker/daemon.json
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "1M",
    "max-file": "1"
  }
}
```

2. Restart the Docker service:

```
systemctl restart docker
```

3. Restart Fluentd.



WARNING

Restarting Fluentd on more than a dozen nodes at once will create a large load on the Kubernetes scheduler. Exercise caution when using the following the directions to restart Fluentd.

There are two methods for restarting Fluentd. You can restart the Fluentd on one node or a set of nodes, or on all nodes.

a. The following steps demonstrate how to restart Fluentd on one node or a set of nodes.

i. List the nodes where Fluentd is running:

```
$ oc get nodes -l logging-infra-fluentd=true
```

ii. For each node, remove the label and turn off Fluentd:

```
$ oc label node $node logging-infra-fluentd-
```

iii. Verify Fluentd is off:

```
$ oc get pods -l component=fluentd
```

iv. For each node, restart Fluentd:

```
$ oc label node $node logging-infra-fluentd=true
```

b. The following steps demonstrate how to restart the Fluentd all nodes.

i. Turn off Fluentd on all nodes:

```
$ oc label node -l logging-infra-fluentd=true --overwrite  
logging-infra-fluentd=false
```

ii. Verify Fluentd is off:

```
$ oc get pods -l component=fluentd
```

iii. Restart Fluentd on all nodes:

```
$ oc label node -l logging-infra-fluentd=false --overwrite  
logging-infra-fluentd=true
```

iv. Verify Fluentd is on:

```
$ oc get pods -l component=fluentd
```

33.12. MANUAL ELASTICSEARCH ROLLOUTS

As of OpenShift Container Platform 3.7 the Aggregated Logging stack updated the Elasticsearch Deployment Config object so that it no longer has a Config Change Trigger, meaning any changes to the **dc** will not result in an automatic rollout. This was to prevent unintended restarts happening in the ES cluster, which could create excessive shard rebalancing as cluster members restart.

This section presents two restart procedures: [rolling-restart](#) and [full-restart](#). Where a rolling restart applies appropriate changes to the Elasticsearch cluster without down time (provided three masters are configured) and a full restart safely applies major changes without risk to existing data.

33.12.1. Performing an Elasticsearch Rolling Cluster Restart

A rolling restart is recommended, when any of the following changes are made:

- nodes on which Elasticsearch pods run require a reboot
- logging-elasticsearch configmap
- logging-es-* deployment configuration
- new image deployment, or upgrade

This will be the recommended restart policy going forward.



NOTE

Any action you do for an ES cluster will need to be repeated for the ops cluster if **openshift_logging_use_ops** was configured to be **True**.

1. Prevent shard balancing when purposely bringing down nodes:

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
  curl -s
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPUT 'https://localhost:9200/_cluster/settings' \
  -d '{ "transient": { "cluster.routing.allocation.enable" :
    "none" } }'
```

2. Once complete, for each **dc** you have for an ES cluster, run **oc rollout latest** to deploy the latest version of the **dc** object:

```
$ oc rollout latest <dc_name>
```

You will see a new pod deployed. Once the pod has two ready containers, you can move on to the next **dc**.

3. Once all **dc**'s for the cluster have been rolled out, re-enable shard balancing:


```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
  curl -s
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPUT 'https://localhost:9200/_cluster/settings' \
  -d '{ "transient": { "cluster.routing.allocation.enable" :
    "all" } }'
```

33.12.2. Performing an Elasticsearch Full Cluster Restart

A full restart is recommended when changing major versions of Elasticsearch or other changes which might put data integrity at risk during the change process.



NOTE

Any action you do for an ES cluster will need to be repeated for the ops cluster if **openshift_logging_use_ops** was configured to be **True**.



NOTE

When making changes to the **logging-es-ops** service use components "es-ops-blocked" and "es-ops" instead in the patch

1. Disable all external communications to the ES cluster while it is down. Edit your non-cluster logging service (for example, **logging-es**, **logging-es-ops**) to no longer match the ES pods running:

```
$ oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es-
  blocked"},"provider":"openshift"}}}'
```

2. Perform a shard synced flush to ensure there are no pending operations waiting to be written to disk prior to shutting down:

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
  curl -s
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPOST 'https://localhost:9200/_flush/synced'
```

3. Prevent shard balancing when purposely bringing down nodes:

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
  curl -s
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPUT 'https://localhost:9200/_cluster/settings' \
  -d '{ "transient": { "cluster.routing.allocation.enable" :
    "none" } }'
```

4. Once complete, for each **dc** you have for an ES cluster, scale down all replicas:

```
$ oc scale dc <dc_name> --replicas=0
```

5. Once scale down is complete, for each **dc** you have for an ES cluster, run **oc rollout latest** to deploy the latest version of the **dc** object:

```
$ oc rollout latest <dc_name>
```

You will see a new pod deployed. Once the pod has two ready containers, you can move on to the next **dc**.

6. Once deployment is complete, for each **dc** you have for an ES cluster, scale up replicas:

```
$ oc scale dc <dc_name> --replicas=1
```

7. Once the scale up is complete, enable all external communications to the ES cluster. Edit your non-cluster logging service (for example, **logging-es**, **logging-es-ops**) to match the ES pods running again:

```
$ oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es","provider":"openshift"}}}'
```

CHAPTER 34. AGGREGATE LOGGING SIZING GUIDELINES

34.1. OVERVIEW

The [Elasticsearch, Fluentd, and Kibana](#) (EFK) stack aggregates logs from nodes and applications running inside your OpenShift Container Platform installation. Once deployed it uses [Fluentd](#) to aggregate event logs from all nodes, projects, and pods into [Elasticsearch \(ES\)](#). It also provides a centralized [Kibana](#) web UI where users and administrators can create rich visualizations and dashboards with the aggregated data.

Fluentd [bulk uploads](#) logs to an index, in JSON format, then Elasticsearch routes your search requests to the appropriate shards.

34.2. INSTALLATION

The general procedure for installing an aggregate logging stack in OpenShift Container Platform is described in [Aggregating Container Logs](#). There are some important things to keep in mind while going through the installation guide:

In order for the logging pods to spread evenly across your cluster, an empty [node selector](#) should be used.

```
$ oc adm new-project logging --node-selector=""
```

In conjunction with node labeling, which is done later, this controls pod placement across the logging project. You can now create the logging project.

```
$ oc project logging
```

Elasticsearch (ES) should be deployed with a cluster size of at least three for resiliency to node failures. This is specified by setting the `openshift_logging_es_cluster_size` parameter in the inventory host file.

Refer to [Ansible Variables](#) for a full list of parameters.

If you do not have an existing Kibana installation, you can use `kibana.example.com` as a value to `openshift_logging_kibana_hostname`.

Installation can take some time depending on whether the images were already retrieved from the registry or not, and on the size of your cluster.

Inside the `logging` namespace, you can check your deployment with `oc get all`.

```
$ oc get all
```

NAME	REVISION	REPLICAS
TRIGGERED BY		
logging-curator	1	1
logging-es-6cvk237t	1	1
logging-es-e5x4t4ai	1	1
logging-es-xmwvnrsv	1	1
logging-kibana	1	1
NAME	DESIRED	CURRENT
AGE		

```

logging-curator-1          1          1          3d
logging-es-6cvk237t-1      1          1          3d
logging-es-e5x4t4ai-1      1          1          3d
logging-es-xmwvnrsv-1      1          1          3d
logging-kibana-1           1          1          3d
NAME                        HOST/PORT    PATH
SERVICE                    TERMINATION  LABELS
logging-kibana              kibana.example.com
logging-kibana              reencrypt   component=support, logging-
infra=support, provider=openshift
logging-kibana-ops         kibana-ops.example.com
logging-kibana-ops         reencrypt   component=support, logging-
infra=support, provider=openshift
NAME                        CLUSTER-IP    EXTERNAL-IP
PORT(S)                    AGE
logging-es                 172.24.155.177 <none>
9200/TCP                   3d
logging-es-cluster         None           <none>
9300/TCP                   3d
logging-es-ops             172.27.197.57 <none>
9200/TCP                   3d
logging-es-ops-cluster     None           <none>
9300/TCP                   3d
logging-kibana             172.27.224.55 <none>
443/TCP                    3d
logging-kibana-ops         172.25.117.77 <none>
443/TCP                    3d
NAME                        READY          STATUS
RESTARTS                    AGE
logging-curator-1-6s7wy     1/1           Running        0
3d
logging-deployer-un6ut      0/1           Completed      0
3d
logging-es-6cvk237t-1-cnvw3 1/1           Running        0
3d
logging-es-e5x4t4ai-1-v933h 1/1           Running        0
3d
logging-es-xmwvnrsv-1-adr5x 1/1           Running        0
3d
logging-fluentd-156xn       1/1           Running        0
3d
logging-fluentd-40biz       1/1           Running        0
3d
logging-fluentd-8k847       1/1           Running        0
3d

```

You should end up with a similar setup to the following.

```
$ oc get pods -o wide
```

```

NAME                        READY    STATUS    RESTARTS  AGE
NODE
logging-curator-1-6s7wy     1/1     Running   0          3d
ip-172-31-24-239.us-west-2.compute.internal
logging-deployer-un6ut      0/1     Completed 0          3d
ip-172-31-6-152.us-west-2.compute.internal

```

```

logging-es-6cvk237t-1-cnvw3    1/1      Running    0          3d
ip-172-31-24-238.us-west-2.compute.internal
logging-es-e5x4t4ai-1-v933h    1/1      Running    0          3d
ip-172-31-24-235.us-west-2.compute.internal
logging-es-xmwvnrsv-1-adr5x     1/1      Running    0          3d
ip-172-31-24-233.us-west-2.compute.internal
logging-fluentd-156xn           1/1      Running    0          3d
ip-172-31-24-241.us-west-2.compute.internal
logging-fluentd-40biz           1/1      Running    0          3d
ip-172-31-24-236.us-west-2.compute.internal
logging-fluentd-8k847           1/1      Running    0          3d
ip-172-31-24-237.us-west-2.compute.internal
logging-fluentd-9a3qx           1/1      Running    0          3d
ip-172-31-24-231.us-west-2.compute.internal
logging-fluentd-abvgj           1/1      Running    0          3d
ip-172-31-24-228.us-west-2.compute.internal
logging-fluentd-bh74n           1/1      Running    0          3d
ip-172-31-24-238.us-west-2.compute.internal
...
...

```

By default the amount of RAM allocated to each ES instance is 16GB.

openshift_logging_es_memory_limit is the parameter used in the **openshift-ansible** host inventory file. Keep in mind that **half** of this value will be passed to the individual elasticsearch pods java processes [heap size](#).

[Learn more about installing EFK.](#)

34.2.1. Large Clusters

At 100 nodes or more, it is recommended to first pre-pull the logging images from **docker pull registry.access.redhat.com/openshift3/logging-fluentd:v3.10**. After deploying the logging infrastructure pods (Elasticsearch, Kibana, and Curator), node labeling should be done in steps of 20 nodes at a time. For example:

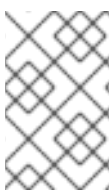
Using a simple loop:

```
$ while read node; do oc label nodes $node logging-infra-fluentd=true;
done < 20_fluentd.lst
```

The following also works:

```
$ oc label nodes 10.10.0.{100..119} logging-infra-fluentd=true
```

Labeling nodes in groups paces the DaemonSets used by OpenShift logging, helping to avoid contention on shared resources such as the image registry.



NOTE

Check for the occurrence of any "CrashLoopBackOff | ImagePullFailed | Error" issues. **oc logs <pod>**, **oc describe pod <pod>** and **oc get event** are helpful diagnostic commands.

34.3. SYSTEMD-JOURNALD AND RSYSLOG

Rate-limiting

In Red Hat Enterprise Linux (RHEL) 7 the **systemd-journald.socket** unit creates **/dev/log** during the boot process, and then passes input to **systemd-journald.service**. Every **syslog()** call goes to the journal.

Rsyslog uses the **imjournal** module as a default input mode for journal files. Refer to [Interaction of rsyslog and journal](#) for detailed information about this topic.

A simple test harness was developed, which uses [logger](#) across the cluster nodes to make entries of different sizes at different rates in the system log. During testing simulations under a default Red Hat Enterprise Linux (RHEL) 7 installation with **systemd-219-19.el7.x86_64** at certain logging rates (approximately 40 log lines per second), we encountered the default rate limit of **rsyslogd**. After adjusting these limits, entries stopped being written to journald due to local journal file corruption. [This issue is resolved in later versions of systemd](#).

Scaling up

As you scale up your project, the default logging environment might need some adjustments. After updating to **systemd-219-22.el7.x86_64**, we added:

```
$IMUXSockRateLimitInterval 0
$IMJournalRateLimitInterval 0
```

to **/etc/rsyslog.conf** and:

```
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
```

to **/etc/systemd/journald.conf**.

Now, restart the services.

```
$ systemctl restart systemd-journald.service
$ systemctl restart rsyslog.service
```

These settings account for the bursty nature of uploading in bulk.

After removing the rate limit, you may see increased CPU utilization on the system logging daemons as it processes any messages that would have previously been throttled.

Rsyslog is configured (see **ratelimit.interval**, **ratelimit.burst**) to rate-limit entries read from the journal at 10,000 messages in 300 seconds. A good rule of thumb is to ensure that the rsyslog rate-limits account for the systemd-journald rate-limits.

34.4. SCALING UP EFK LOGGING

If you do not indicate the desired scale at first deployment, the least disruptive way of adjusting your cluster is by re-running the Ansible logging playbook after updating the inventory file with an updated **openshift_logging_es_cluster_size** value. parameter. Refer to the [Performing Administrative Elasticsearch Operations](#) section for more in-depth information.

34.5. STORAGE CONSIDERATIONS

An Elasticsearch index is a collection of shards and its corresponding replica shards. This is how ES implements high availability internally, therefore there is little need to use hardware based mirroring RAID variants. RAID 0 can still be used to increase overall disk performance.

Every search request needs to hit a copy of every shard in the index. Each ES instance requires its own individual storage, but an OpenShift Container Platform deployment can only provide volumes shared by all of its pods, which again means that Elasticsearch shouldn't be implemented with a single node.

A [persistent volume](#) should be added to each Elasticsearch deployment configuration so that we have one volume per [replica shard](#). On OpenShift Container Platform this is often achieved through [Persistent Volume Claims](#)

- 1 volume per shard
- 1 volume per replica shard

The PVCs must be named based on the `openshift_logging_es_pvc_prefix` setting. Refer to [Persistent Elasticsearch Storage](#) for more details.

Below are capacity planning guidelines for OpenShift Container Platform aggregate logging. **Example scenario**

Assumptions:

1. Which application: Apache
2. Bytes per line: 256
3. Lines per second load on application: 1
4. Raw text data → JSON

Baseline (256 characters per minute → 15KB/min)

Logging Infra Pods	Storage Throughput
3 es 1 kibana 1 curator 1 fluentd	6 pods total: 90000 x 86400 = 7,7 GB/day
3 es 1 kibana 1 curator 11 fluentd	16 pods total: 225000 x 86400 = 24,0 GB/day
3 es 1 kibana 1 curator 20 fluentd	25 pods total: 225000 x 86400 = 32,4 GB/day

Calculating total logging throughput and disk space required for your logging environment requires knowledge of your application. For example, if one of your applications on average logs 10 lines-per-second, each 256 bytes-per-line, calculate per-application throughput and disk space as follows:

```
(bytes-per-line * (lines-per-second) = 2560 bytes per app per second
(2560) * (number-of-pods-per-node,100) = 256,000 bytes per second per node
256k * (number-of-nodes) = total logging throughput per cluster
```

Fluentd ships any logs from **systemd journal** and `/var/lib/docker/containers/` to Elasticsearch. [Learn more](#).

Local SSD drives are recommended in order to achieve the best performance. In Red Hat Enterprise Linux (RHEL) 7, the [deadline](#) IO scheduler is the default for all block devices except SATA disks. For SATA disks, the default IO scheduler is **cfq**.

Sizing storage for ES is greatly dependent on how you optimize your indices. Therefore, consider how much data you need in advance and that you are aggregating application log data. Some Elasticsearch users have found that it is necessary to [keep absolute storage consumption around 50% and below 70% at all times](#). This helps to avoid Elasticsearch becoming unresponsive during large merge operations.

CHAPTER 35. ENABLING CLUSTER METRICS

35.1. OVERVIEW

The [kubelet](#) exposes metrics that can be collected and stored in back-ends by [Heapster](#).

As an OpenShift Container Platform administrator, you can view a cluster's metrics from all containers and components in one user interface. These metrics are also used by [horizontal pod autoscalers](#) in order to determine when and how to scale.

This topic describes using [Hawkular Metrics](#) as a metrics engine which stores the data persistently in a [Cassandra](#) database. When this is configured, CPU, memory and network-based metrics are viewable from the OpenShift Container Platform web console and are available for use by [horizontal pod autoscalers](#).

Heapster retrieves a list of all nodes from the master server, then contacts each node individually through the `/stats` endpoint. From there, Heapster scrapes the metrics for CPU, memory and network usage, then exports them into Hawkular Metrics.

The storage volume metrics available on the kubelet are not available through the `/stats` endpoint, but are available through the `/metrics` endpoint. See OpenShift Container Platform via Prometheus for detailed information.

Browsing individual pods in the web console displays separate sparkline charts for memory and CPU. The time range displayed is selectable, and these charts automatically update every 30 seconds. If there are multiple containers on the pod, then you can select a specific container to display its metrics.

If [resource limits](#) are defined for your project, then you can also see a donut chart for each pod. The donut chart displays usage against the resource limit. For example: **145 Available of 200 MiB**, with the donut chart showing **55 MiB Used**.

35.2. BEFORE YOU BEGIN

An Ansible playbook is available to deploy and upgrade cluster metrics. You should familiarize yourself with the [Installing Clusters](#) guide. This provides information for preparing to use Ansible and includes information about configuration. Parameters are added to the Ansible inventory file to configure various areas of cluster metrics.

The following describe the various areas and the parameters that can be added to the Ansible inventory file in order to modify the defaults:

- [Metrics Project](#)
- [Metrics Data Storage](#)

35.3. METRICS PROJECT

The components for cluster metrics must be deployed to the **openshift-infra** project in order for autoscaling to work. [Horizontal pod autoscalers](#) specifically use this project to discover the Heapster service and use it to retrieve metrics. The metrics project can be changed by adding **openshift_metrics_project** to the inventory file.

35.4. METRICS DATA STORAGE

You can store the metrics data to either [persistent storage](#) or to a temporary [pod volume](#).

35.4.1. Persistent Storage

Running OpenShift Container Platform cluster metrics with persistent storage means that your metrics are stored to a [persistent volume](#) and are able to survive a pod being restarted or recreated. This is ideal if you require your metrics data to be guarded from data loss. For production environments it is highly recommended to configure persistent storage for your metrics pods.

The size requirement of the Cassandra storage is dependent on the number of pods. It is the administrator's responsibility to ensure that the size requirements are sufficient for their setup and to monitor usage to ensure that the disk does not become full. The size of the persisted volume claim is specified with the `openshift_metrics_cassandra_pvc_size` [Ansible variable](#) which is set to 10 GB by default.

If you would like to use [dynamically provisioned persistent volumes](#) set the `openshift_metrics_cassandra_storage_type` [variable](#) to `dynamic` in the inventory file.

35.4.2. Capacity Planning for Cluster Metrics

After running the `openshift_metrics` Ansible role, the output of `oc get pods` should resemble the following:

```
# oc get pods -n openshift-infra
NAME                                READY   STATUS    RESTARTS   AGE
hawkular-cassandra-1-l5y4g          1/1     Running   0          17h
hawkular-metrics-1t9so              1/1     Running   0          17h
heapster-febru                      1/1     Running   0          17h
```

OpenShift Container Platform metrics are stored using the Cassandra database, which is deployed with settings of `openshift_metrics_cassandra_limits_memory: 2G`; this value could be adjusted further based upon the available memory as determined by the Cassandra start script. This value should cover most OpenShift Container Platform metrics installations, but using environment variables you can modify the `MAX_HEAP_SIZE` along with heap new generation size, `HEAP_NEWSIZE`, in the Cassandra Dockerfile prior to deploying cluster metrics.

By default, metrics data is stored for seven days. After seven days, Cassandra begins to purge the oldest metrics data. Metrics data for deleted pods and projects is not automatically purged; it is only removed once the data is more than seven days old.

Example 35.1. Data Accumulated by 10 Nodes and 1000 Pods

In a test scenario including 10 nodes and 1000 pods, a 24 hour period accumulated 2.5 GB of metrics data. Therefore, the capacity planning formula for metrics data in this scenario is:

$$(((2.5 \times 10^9) \div 1000) \div 24) \div 10^6 = \sim 0.125 \text{ MB/hour per pod.}$$

Example 35.2. Data Accumulated by 120 Nodes and 10000 Pods

In a test scenario including 120 nodes and 10000 pods, a 24 hour period accumulated 25 GB of metrics data. Therefore, the capacity planning formula for metrics data in this scenario is:

$$(((11.410 \times 10^9) \div 1000) \div 24) \div 10^6 = 0.475 \text{ MB/hour}$$

	1000 pods	10000 pods
Cassandra storage data accumulated over 24 hours (default metrics parameters)	2.5 GB	11.4 GB

If the default value of 7 days for **openshift_metrics_duration** and 30 seconds for **openshift_metrics_resolution** are preserved, then weekly storage requirements for the Cassandra pod would be:

	1000 pods	10000 pods
Cassandra storage data accumulated over seven days (default metrics parameters)	20 GB	90 GB

In the previous table, an additional 10 percent was added to the expected storage space as a buffer for unexpected monitored pod usage.



WARNING

If the Cassandra persisted volume runs out of sufficient space, then data loss occurs.

For cluster metrics to work with persistent storage, ensure that the persistent volume has the **ReadWriteOnce** access mode. If this mode is not active, then the persistent volume claim cannot locate the persistent volume, and Cassandra fails to start.

To use persistent storage with the metric components, ensure that a [persistent volume](#) of sufficient size is available. The creation of [persistent volume claims](#) is handled by the OpenShift Ansible **openshift_metrics** role.

OpenShift Container Platform metrics also supports dynamically-provisioned persistent volumes. To use this feature with OpenShift Container Platform metrics, it is necessary to set the value of **openshift_metrics_cassandra_storage_type** to **dynamic**. You can use EBS, GCE, and Cinder storage back-ends to [dynamically provision persistent volumes](#).

For information on configuring the performance and scaling the cluster metrics pods, see the [Scaling Cluster Metrics](#) topic.

Table 35.1. Cassandra Database storage requirements based on number of nodes/pods in the cluster

Number of Nodes	Number of Pods	Cassandra Storage growth speed	Cassandra storage growth per day	Cassandra storage growth per week
210	10500	500 MB per hour	15 GB	75 GB
990	11000	1 GB per hour	30 GB	210 GB

In the above calculation, approximately 20 percent of the expected size was added as overhead to ensure that the storage requirements do not exceed calculated value.

If the **METRICS_DURATION** and **METRICS_RESOLUTION** values are kept at the default (7 days and 15 seconds respectively), it is safe to plan Cassandra storage size requirements for week, as in the values above.



WARNING

Because OpenShift Container Platform metrics uses the Cassandra database as a datastore for metrics data, if **USE_PERSISTANT_STORAGE=true** is set during the metrics set up process, **PV** will be on top in the network storage, with NFS as the default. However, using network storage in combination with Cassandra is not recommended, as per the [Cassandra documentation](#).

Known Issues and Limitations

Testing found that the **heapster** metrics component is capable of handling up to 25,000 pods. If the amount of pods exceed that number, Heapster begins to fall behind in metrics processing, resulting in the possibility of metrics graphs no longer appearing. Work is ongoing to increase the number of pods that Heapster can gather metrics on, as well as upstream development of alternate metrics-gathering solutions.

35.4.3. Non-Persistent Storage

Running OpenShift Container Platform cluster metrics with non-persistent storage means that any stored metrics are deleted when the pod is deleted. While it is much easier to run cluster metrics with non-persistent data, running with non-persistent data does come with the risk of permanent data loss. However, metrics can still survive a container being restarted.

In order to use non-persistent storage, you must set the **openshift_metrics_cassandra_storage_type** variable to **emptydir** in the inventory file.



NOTE

When using non-persistent storage, metrics data is written to **/var/lib/origin/openshift.local.volumes/pods** on the node where the Cassandra pod runs. Ensure **/var** has enough free space to accommodate metrics storage.

35.5. METRICS ANSIBLE ROLE

The OpenShift Container Platform Ansible **openshift_metrics** role configures and deploys all of the metrics components using the variables from the [Configuring Ansible](#) inventory file.

35.5.1. Specifying Metrics Ansible Variables

The **openshift_metrics** role included with OpenShift Ansible defines the tasks to deploy cluster metrics. The following is a list of role variables that can be added to your inventory file if it is necessary to override them.

Table 35.2. Ansible Variables

Variable	Description
openshift_metrics_install_metrics	Deploy metrics if true . Otherwise, undeploy.
openshift_metrics_start_cluster	Start the metrics cluster after deploying the components.
openshift_metrics_startup_timeout	The time, in seconds, to wait until Hawkular Metrics and Heapster start up before attempting a restart.
openshift_metrics_duration	The number of days to store metrics before they are purged.
openshift_metrics_resolution	The frequency that metrics are gathered. Defined as a number and time identifier: seconds (s), minutes (m), hours (h).
openshift_metrics_cassandra_pvc_prefix	The persistent volume claim prefix created for Cassandra. A serial number is appended to the prefix starting from 1.
openshift_metrics_cassandra_pvc_size	The persistent volume claim size for each of the Cassandra nodes.
openshift_metrics_cassandra_pvc_storage_class_name	If you want to explicitly set the storage class, you must not set openshift_metrics_cassandra_storage_type to emptydir or dynamic .
openshift_metrics_cassandra_storage_type	Use emptydir for ephemeral storage (for testing); pv for persistent volumes, which need to be created before the installation; or dynamic for dynamic persistent volumes.
openshift_metrics_cassandra_replicas	The number of Cassandra nodes for the metrics stack. This value dictates the number of Cassandra replication controllers.

Variable	Description
<code>openshift_metrics_cassandra_limits_memory</code>	The memory limit for the Cassandra pod. For example, a value of 2Gi would limit Cassandra to 2 GB of memory. This value could be further adjusted by the start script based on available memory of the node on which it is scheduled.
<code>openshift_metrics_cassandra_limits_cpu</code>	The CPU limit for the Cassandra pod. For example, a value of 4000m (4000 millicores) would limit Cassandra to 4 CPUs.
<code>openshift_metrics_cassandra_requests_memory</code>	The amount of memory to request for Cassandra pod. For example, a value of 2Gi would request 2 GB of memory.
<code>openshift_metrics_cassandra_requests_cpu</code>	The CPU request for the Cassandra pod. For example, a value of 4000m (4000 millicores) would request 4 CPUs.
<code>openshift_metrics_cassandra_storage_group</code>	The supplemental storage group to use for Cassandra.
<code>openshift_metrics_cassandra_nodeselector</code>	Set to the desired, existing node selector to ensure that pods are placed onto nodes with specific labels. For example, <code>{"node-role.kubernetes.io/infra": "true"}</code> .
<code>openshift_metrics_hawkular_ca</code>	An optional certificate authority (CA) file used to sign the Hawkular certificate.
<code>openshift_metrics_hawkular_cert</code>	The certificate file used for re-encrypting the route to Hawkular metrics. The certificate must contain the host name used by the route. If unspecified, the default router certificate is used.
<code>openshift_metrics_hawkular_key</code>	The key file used with the Hawkular certificate.
<code>openshift_metrics_hawkular_limits_memory</code>	The amount of memory to limit the Hawkular pod. For example, a value of 2Gi would limit the Hawkular pod to 2 GB of memory. This value could be further adjusted by the start script based on available memory of the node on which it is scheduled.
<code>openshift_metrics_hawkular_limits_cpu</code>	The CPU limit for the Hawkular pod. For example, a value of 4000m (4000 millicores) would limit the Hawkular pod to 4 CPUs.
<code>openshift_metrics_hawkular_replicas</code>	The number of replicas for Hawkular metrics.

Variable	Description
<code>openshift_metrics_hawkular_requests_memory</code>	The amount of memory to request for the Hawkular pod. For example, a value of 2Gi would request 2 GB of memory.
<code>openshift_metrics_hawkular_requests_cpu</code>	The CPU request for the Hawkular pod. For example, a value of 4000m (4000 millicores) would request 4 CPUs.
<code>openshift_metrics_hawkular_nodeselector</code>	Set to the desired, existing node selector to ensure that pods are placed onto nodes with specific labels. For example, <code>{"node-role.kubernetes.io/infra": "true"}</code> .
<code>openshift_metrics_heapster_allowed_users</code>	A comma-separated list of CN to accept. By default, this is set to allow the OpenShift service proxy to connect. Add system:master-proxy to the list when overriding in order to allow horizontal pod autoscaling to function properly.
<code>openshift_metrics_heapster_limits_memory</code>	The amount of memory to limit the Heapster pod. For example, a value of 2Gi would limit the Heapster pod to 2 GB of memory.
<code>openshift_metrics_heapster_limits_cpu</code>	The CPU limit for the Heapster pod. For example, a value of 4000m (4000 millicores) would limit the Heapster pod to 4 CPUs.
<code>openshift_metrics_heapster_requests_memory</code>	The amount of memory to request for Heapster pod. For example, a value of 2Gi would request 2 GB of memory.
<code>openshift_metrics_heapster_requests_cpu</code>	The CPU request for the Heapster pod. For example, a value of 4000m (4000 millicores) would request 4 CPUs.
<code>openshift_metrics_heapster_standalone</code>	Deploy only Heapster, without the Hawkular Metrics and Cassandra components.
<code>openshift_metrics_heapster_nodeselector</code>	Set to the desired, existing node selector to ensure that pods are placed onto nodes with specific labels. For example, <code>{"node-role.kubernetes.io/infra": "true"}</code> .
<code>openshift_metrics_hawkular_hostname</code>	Set when executing the <code>openshift_metrics</code> Ansible role, since it uses the host name for the Hawkular Metrics route . This value should correspond to a fully qualified domain name.

See [Compute Resources](#) for further discussion on how to specify requests and limits.

If you are using [persistent storage](#) with Cassandra, it is the administrator's responsibility to set a sufficient disk size for the cluster using the **openshift_metrics_cassandra_pvc_size** variable. It is also the administrator's responsibility to monitor disk usage to make sure that it does not become full.



WARNING

Data loss results if the Cassandra persisted volume runs out of sufficient space.

All of the other variables are optional and allow for greater customization. For instance, if you have a custom install in which the Kubernetes master is not available under **https://kubernetes.default.svc:443** you can specify the value to use instead with the **openshift_metrics_master_url** parameter.

35.5.2. Using Secrets

The OpenShift Container Platform Ansible **openshift_metrics** role auto-generates self-signed certificates for use between its components and generates a [re-encrypting route](#) to expose the Hawkular Metrics service. This route is what allows the web console to access the Hawkular Metrics service.

In order for the browser running the web console to trust the connection through this route, it must trust the route's certificate. This can be accomplished by [providing your own certificates](#) signed by a trusted Certificate Authority. The **openshift_metrics** role allows you to specify your own certificates, which it then uses when creating the route.

The router's default certificate are used if you do not provide your own.

35.5.2.1. Providing Your Own Certificates

To provide your own certificate, which is used by the [re-encrypting route](#), you can set the **openshift_metrics_hawkular_cert**, **openshift_metrics_hawkular_key**, and **openshift_metrics_hawkular_ca** variables in your inventory file.

The **hawkular-metrics.pem** value needs to contain the certificate in its **.pem** format. You may also need to provide the certificate for the Certificate Authority which signed this **.pem** file via the **hawkular-metrics-ca.cert** secret.

For more information, see the [re-encryption route documentation](#).

35.6. DEPLOYING THE METRIC COMPONENTS

Because deploying and configuring all the metric components is handled with OpenShift Container Platform Ansible, you can deploy everything in one step.

The following examples show you how to deploy metrics with and without persistent storage using the default parameters.

**IMPORTANT**

The host that you run the Ansible playbook on must have at least 75MiB of free memory per host in the inventory.

**IMPORTANT**

In accordance with upstream Kubernetes rules, metrics can be collected only on the default interface of **eth0**.

Example 35.3. Deploying with Persistent Storage

The following command sets the Hawkular Metrics route to use **hawkular-metrics.example.com** and is deployed using persistent storage.

You must have a persistent volume of sufficient size available.

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
  -e openshift_metrics_install_metrics=True \
  -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com \
  -e openshift_metrics_cassandra_storage_type=pv
```

Example 35.4. Deploying without Persistent Storage

The following command sets the Hawkular Metrics route to use **hawkular-metrics.example.com** and deploy without persistent storage.

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
  -e openshift_metrics_install_metrics=True \
  -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com
```

**WARNING**

Because this is being deployed without persistent storage, metric data loss can occur.

35.6.1. Metrics Diagnostics

There are some diagnostics for metrics to assist in evaluating the state of the metrics stack. To execute diagnostics for metrics:

```
$ oc adm diagnostics MetricsApiProxy
```

35.7. SETTING THE METRICS PUBLIC URL

The OpenShift Container Platform web console uses the data coming from the Hawkular Metrics service to display its graphs. The URL for accessing the Hawkular Metrics service must be configured with the **metricsPublicURL** option in the [master webconsole-config configmap file](#). This URL corresponds to the route created with the **openshift_metrics_hawkular_hostname** inventory variable used during the [deployment](#) of the metrics components.



NOTE

You must be able to resolve the **openshift_metrics_hawkular_hostname** from the browser accessing the console.

For example, if your **openshift_metrics_hawkular_hostname** corresponds to **hawkular-metrics.example.com**, then you must make the following change in the webconsole-config configmap file:

```
clusterInfo:
  ...
  metricsPublicURL: "https://hawkular-
    metrics.example.com/hawkular/metrics"
```

Once you have updated and saved the webconsole-config configmap file, you must restart your OpenShift Container Platform instance.

When your OpenShift Container Platform server is back up and running, metrics are displayed on the pod overview pages.

CAUTION

If you are using self-signed certificates, remember that the Hawkular Metrics service is hosted under a different host name and uses different certificates than the console. You may need to explicitly open a browser tab to the value specified in **metricsPublicURL** and accept that certificate.

To avoid this issue, use certificates which are configured to be acceptable by your browser.

35.8. ACCESSING HAWKULAR METRICS DIRECTLY

To access and manage metrics more directly, use the [Hawkular Metrics API](#).



NOTE

When accessing Hawkular Metrics from the API, you are only able to perform reads. Writing metrics is disabled by default. If you want individual users to also be able to write metrics, you must set the **openshift_metrics_hawkular_user_write_access** [variable](#) to **true**.

However, it is recommended to use the default configuration and only have metrics enter the system via Heapster. If write access is enabled, any user can write metrics to the system, which can affect performance and cause Cassandra disk usage to unpredictably increase.

The [Hawkular Metrics documentation](#) covers how to use the API, but there are a few differences when dealing with the version of Hawkular Metrics configured for use on OpenShift Container Platform:

35.8.1. OpenShift Container Platform Projects and Hawkular Tenants

Hawkular Metrics is a multi-tenanted application. It is configured so that a project in OpenShift Container Platform corresponds to a tenant in Hawkular Metrics.

As such, when accessing metrics for a project named **MyProject** you must set the [Hawkular-Tenant](#) header to **MyProject**.

There is also a special tenant named **_system** which contains system level metrics. This requires either a **cluster-reader** or **cluster-admin** level privileges to access.

35.8.2. Authorization

The Hawkular Metrics service authenticates the user against OpenShift Container Platform to determine if the user has access to the project it is trying to access.

Hawkular Metrics accepts a bearer token from the client and verifies that token with the OpenShift Container Platform server using a **SubjectAccessReview**. If the user has proper read privileges for the project, they are allowed to read the metrics for that project. For the **_system** tenant, the user requesting to read from this tenant must have **cluster-reader** permission.

When accessing the Hawkular Metrics API, you must pass a bearer token in the **Authorization** header.

35.9. SCALING OPENSIFT CONTAINER PLATFORM CLUSTER METRICS PODS

Information about scaling cluster metrics capabilities is available in the [Scaling and Performance Guide](#).

35.10. CLEANUP

You can remove everything deployed by the OpenShift Container Platform Ansible **openshift_metrics** role by performing the following steps:

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
    -e openshift_metrics_install_metrics=False
```

35.11. PROMETHEUS ON OPENSIFT CONTAINER PLATFORM

Prometheus is a stand-alone, open source systems monitoring and alerting toolkit. You can use Prometheus to visualize metrics and alerts for OpenShift Container Platform system resources.



IMPORTANT

Prometheus on OpenShift Container Platform is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see <https://access.redhat.com/support/offerings/techpreview/>.

35.11.1. Setting Prometheus Role Variables

The Prometheus role creates:

- The **openshift-metrics** namespace.
- Prometheus **clusterrolebinding** and service account.
- Prometheus pod with Prometheus behind OAuth proxy, Alertmanager, and Alert Buffer as a stateful set.
- Prometheus and **prometheus-alerts** ConfigMaps.
- Prometheus and Prometheus Alerts services and direct routes.

Prometheus deployment is enabled by default, uninstall it by setting **openshift_prometheus_state** to **absent**. For example:

```
# openshift_prometheus_state=absent
```

Set the following role variables to install and configure Prometheus.

Table 35.3. Prometheus Variables

Variable	Description
openshift_prometheus_state	The default value is present , which results in the installation or update of Prometheus. To uninstall Prometheus, set to absent .
openshift_prometheus_namespace	Project namespace where the components are deployed. Default set to openshift-metrics . For example, openshift_prometheus_namespace=\${USE_R_PROJECT} .
openshift_prometheus_node_selector	Selector for the nodes on which Prometheus is deployed. Default set to node-role.kubernetes.io/infra=true .

Variable	Description
openshift_prometheus_storage_kind	Set to create PV for Prometheus. For example, openshift_prometheus_storage_kind=nfs .
openshift_prometheus_alertmanager_storage_kind	Set to create PV for Alertmanager. For example, openshift_prometheus_alertmanager_storage_kind=nfs .
openshift_prometheus_alertbuffer_storage_kind	Set to create PV for Alert Buffer. For example, openshift_prometheus_alertbuffer_storage_kind=nfs .
openshift_prometheus_storage_type	Set to create PVC for Prometheus. For example, openshift_prometheus_storage_type=pvc .
openshift_prometheus_alertmanager_storage_type	Set to create PVC for Alertmanager. For example, openshift_prometheus_alertmanager_storage_type=pvc .
openshift_prometheus_alertbuffer_storage_type	Set to create PVC for Alert Buffer. For example, openshift_prometheus_alertbuffer_storage_type=pvc .
openshift_prometheus_additional_rules_file	Additional Prometheus rules file. Set to null by default.

35.11.2. Deploying Prometheus Using Ansible Installer



IMPORTANT

The host that you run the Ansible playbook on must have at least 75MiB of free memory per host in the inventory.

The Ansible Installer is the default method of deploying Prometheus.

Run the playbook:

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-prometheus/config.yml
```

**NOTE**

Make sure you have nodes labeled with **node-role.kubernetes.io/infra=true**, which is the default value for **openshift_prometheus_node_selector**. If you want to use other node selectors, please see [Deploy Using Node-Selector](#).

35.11.2.1. Additional Methods for Deploying Prometheus**Deploy Using Node-Selector**

Label the node on which you want to deploy Prometheus:

```
# oc adm label node/$NODE ${KEY}=${VALUE}
```

Deploy Prometheus with Ansible and container resources:

```
# Set node selector for prometheus
openshift_prometheus_node_selector="{${KEY}":"${VALUE}"}
```

Run the playbook:

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml
```

Deploy Using a Non-default Namespace

Identify your namespace:

```
# Set non-default openshift_prometheus_namespace
openshift_prometheus_namespace=${USER_PROJECT}
```

Run the playbook:

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml
```

35.11.2.2. Accessing the Prometheus Web UI

The Prometheus server automatically exposes a Web UI at **localhost:9090**. You can access the Prometheus Web UI with the **view** role.

35.11.2.3. Configuring Prometheus for OpenShift Container Platform**Prometheus Storage Related Variables**

With each Prometheus component (including Prometheus, Alertmanager, Alert Buffer, and OAuth proxy) you can set the PV claim by setting corresponding role variable, for example:

```
openshift_prometheus_storage_type: pvc
openshift_prometheus_alertmanager_pvc_name: alertmanager
openshift_prometheus_alertbuffer_pvc_size: 10G
openshift_prometheus_pvc_access_modes: [ReadWriteOnce]
```

Prometheus Alert Rules File Variable

You can add an external file with alert rules by setting the path to an additional rules variable:

```
openshift_prometheus_additional_rules_file: <PATH>
```

The file must follow [the Prometheus Alert rules format](#). The following example sets a rule to send an alert when one of the cluster nodes is down:

```
groups:
- name: example-rules
  interval: 30s # defaults to global interval
  rules:
  - alert: Node Down
    expr: up{job="kubernetes-nodes"} == 0
    for: 10m 1
    annotations:
      miqTarget: "ContainerNode"
      severity: "HIGH"
      message: "{{ '{{' }}{{ '$labels.instance' }}{{ '}}' }} is down"
```

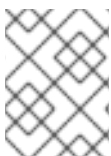
- 1** The optional **for** value specifies the amount of time Prometheus waits before it sends an alert for this element. For example, if you set **10m**, Prometheus waits 10 minutes after it encounters this issue before sending an alert.

Prometheus Variables to Control Resource Limits

With each Prometheus component (including Prometheus, Alertmanager, Alert Buffer, and OAuth proxy) you can specify CPU, memory limits, and requests by setting the corresponding role variable, for example:

```
openshift_prometheus_alertmanager_limits_memory: 1Gi
openshift_prometheus_oauth_proxy_cpu_requests: 100m
```

For more detailed information, see [OpenShift Prometheus](#).



NOTE

Once **openshift_metrics_project: openshift-infra** is installed, metrics can be gathered from the [http://\\${POD_IP}:7575/metrics](http://${POD_IP}:7575/metrics) endpoint.

35.11.3. OpenShift Container Platform Metrics via Prometheus

The state of a system can be gauged by the metrics that it emits. This section describes current and proposed metrics that identify the health of the storage subsystem and cluster.

35.11.3.1. Current Metrics

This section describes the metrics currently emitted from Kubernetes's storage subsystem.

Cloud Provider API Call Metrics

This metric reports the time and count of success and failures of all cloudprovider API calls. These metrics include **aws_attach_time** and **aws_detach_time**. The type of emitted metrics is a histogram, and hence, Prometheus also generates sum, count, and bucket metrics for these metrics.

Example summary of cloudprovider metrics from GCE:

```
cloudprovider_gce_api_request_duration_seconds { request =
"instance_list"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_insert"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_delete"}
cloudprovider_gce_api_request_duration_seconds { request = "attach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "detach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "list_disk"}
```

Example summary of cloudprovider metrics from AWS:

```
cloudprovider_aws_api_request_duration_seconds { request =
"attach_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"detach_volume"}
cloudprovider_aws_api_request_duration_seconds { request = "create_tags"}
cloudprovider_aws_api_request_duration_seconds { request =
"create_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"delete_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"describe_instance"}
cloudprovider_aws_api_request_duration_seconds { request =
"describe_volume"}
```

See [Cloud Provider \(specifically GCE and AWS\) metrics for Storage API calls](#) for more information.

Volume Operation Metrics

These metrics report time taken by a storage operation once started. These metrics keep track of operation time at the plug-in level, but do not include time taken by **goroutine** to run or operation to be picked up from the internal queue. These metrics are a type of histogram.

Example summary of available volume operation metrics

```
storage_operation_duration_seconds { volume_plugin = "aws-efs",
operation_name = "volume_attach" }
storage_operation_duration_seconds { volume_plugin = "aws-efs",
operation_name = "volume_detach" }
storage_operation_duration_seconds { volume_plugin = "glusterfs",
operation_name = "volume_provision" }
storage_operation_duration_seconds { volume_plugin = "gce-pd",
operation_name = "volume_delete" }
storage_operation_duration_seconds { volume_plugin = "vsphere",
operation_name = "volume_mount" }
storage_operation_duration_seconds { volume_plugin = "iscsi" ,
operation_name = "volume_unmount" }
storage_operation_duration_seconds { volume_plugin = "aws-efs",
operation_name = "unmount_device" }
storage_operation_duration_seconds { volume_plugin = "cinder" ,
```



```
operation_name = "verify_volumes_are_attached" }
storage_operation_duration_seconds { volume_plugin = "<n/a>" ,
operation_name = "verify_volumes_are_attached_per_node" }
```

See [Volume operation metrics](#) for more information.

Volume Stats Metrics

These metrics typically report usage stats of PVC (such as used space versus available space). The type of metrics emitted is gauge.

Table 35.4. Volume Stats Metrics

Metric	Type	Labels/tags
volume_stats_capacityBytes	Gauge	namespace,persistentvolumeclaim,persistentvolume=
volume_stats_usedBytes	Gauge	namespace= <persistentvolumeclaim-namespace> persistentvolumeclaim= <persistentvolumeclaim-name> persistentvolume= <persistentvolume-name>
volume_stats_availableBytes	Gauge	namespace= <persistentvolumeclaim-namespace> persistentvolumeclaim= <persistentvolumeclaim-name> persistentvolume=
volume_stats_InodesFree	Gauge	namespace= <persistentvolumeclaim-namespace> persistentvolumeclaim= <persistentvolumeclaim-name> persistentvolume= <persistentvolume-name>
volume_stats_Inodes	Gauge	namespace= <persistentvolumeclaim-namespace> persistentvolumeclaim= <persistentvolumeclaim-name> persistentvolume= <persistentvolume-name>

Metric	Type	Labels/tags
volume_stats_InodesUsed	Gauge	namespace= <persistentvolumeclaim- namespace> persistentvolumeclaim= <persistentvolumeclaim-name> persistentvolume= <persistentvolume-name>

35.11.4. Undeploying Prometheus

To undeploy Prometheus, run:

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-  
prometheus/config.yml -e openshift_prometheus_state=absent
```

CHAPTER 36. CUSTOMIZING THE WEB CONSOLE

36.1. OVERVIEW

Administrators can customize the [web console](#) using extensions, which let you run scripts and load custom stylesheets when the web console loads. Extension scripts allow you to override the default behavior of the web console and customize it for your needs.

For example, extension scripts can be used to add your own company's branding or to add company-specific capabilities. A common use case for this is rebranding or white-labeling for different environments. You can use the same extension code, but provide settings that change the web console.

CAUTION

Take caution making extensive changes to the web console styles or behavior that are not documented below. While you add any scripts or stylesheets, significant customizations might need to be reworked on upgrades as the web console markup and behavior change in future versions.

36.2. LOADING EXTENSION SCRIPTS AND STYLESHEETS

As of OpenShift Container Platform 3.9, extension scripts and stylesheets can be hosted at any **https://** URL as long as the URL is accessible from the browser. The files might be hosted from a pod on the platform using a publicly accessible route, or on another server outside of OpenShift Container Platform.

To add scripts and stylesheets, edit the **webconsole-config** ConfigMap in the **openshift-web-console** namespace. The web console configuration is available in the **webconsole-config.yaml** key of the ConfigMap.

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

To add scripts, update the **extensions.scriptURLs** property. The value is an array of URLs.

To add stylesheets, update the **extensions.stylesheetURLs** property. The value is an array of URLs.

Example extensions.stylesheetURLs Setting

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      scriptURLs:
        - https://example.com/scripts/menu-customization.js
        - https://example.com/scripts/nav-customization.js
      stylesheetURLs:
        - https://example.com/styles/logo.css
        - https://example.com/styles/custom-styles.css
    [...]
```

After saving the ConfigMap, the web console containers will be updated automatically for the new extension files within a few minutes.



NOTE

Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. Scripts must be served with **Content-Type: application/javascript** and stylesheets with **Content-Type: text/css**.

It is a best practice to wrap extension scripts in an Immediately Invoked Function Expression (IIFE). This ensures that you do not create global variables that conflict with the names used by the web console or by other extensions. For example:

```
(function() {
  // Put your extension code here...
})();
```

The examples in the following sections show common ways you can customize the web console.



NOTE

Additional extension examples are available in the [OpenShift Origin](#) repository on GitHub.

36.2.1. Setting Extension Properties

If you have a specific extension, but want to use different text in it for each of the environments, you can define the environment in the web console configuration, and use the same extension script across environments.

To add extension properties, edit the **webconsole-config** ConfigMap in the **openshift-web-console** namespace. The web console configuration is available in the **webconsole-config.yaml** key of the ConfigMap.

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

Update the **extensions.properties** value, which is a map of key-value pairs.

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      [...]
    properties:
      doc_url: https://docs.openshift.com
      key1: value1
      key2: value2
    [...]
```

This results in a global variable that can be accessed by the extension, as if the following code was executed:

```

window.OPENSIFT_EXTENSION_PROPERTIES = {
  doc_url: "https://docs.openshift.com",
  key1: "value1",
  key2: "value2"
}

```

36.3. EXTENSION OPTION FOR EXTERNAL LOGGING SOLUTIONS

As of OpenShift Container Platform 3.6, there is an extension option to link to external logging solutions instead of using OpenShift Container Platform's EFK logging stack:

```

'use strict';
angular.module("mylinkextensions", ['openshiftConsole'])
  .run(function(extensionRegistry) {
    extensionRegistry.add('log-links', _.spread(function(resource,
options) {
      return {
        type: 'dom',
        node: '<span><a href="https://extension-
point.example.com">' + resource.metadata.name + '</a><span class="action-
divider">|</span></span>'
      };
    }));
  });
hawtioPluginLoader.addModule("mylinkextensions");

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.4. CUSTOMIZING AND DISABLING THE GUIDED TOUR

A guided tour will pop up the first time a user logs in on a particular browser. You can enable the **auto_launch** for new users:

```

window.OPENSIFT_CONSTANTS.GUIDED_TOURS.landing_page_tour.auto_launch =
true;

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.5. CUSTOMIZING DOCUMENTATION LINKS

Documentation links on the landing page are customizable.

window.OPENSIFT_CONSTANTS.CATALOG_HELP_RESOURCES is an array of objects containing a title and an **href**. These will be turned into links. You can completely override the array, push or pop additional links, or modify the attributes of existing links. For example:

```

window.OPENSIFT_CONSTANTS.CATALOG_HELP_RESOURCES.push({
  title: 'Blog',
  href: 'https://blog.openshift.com'
});

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.6. CUSTOMIZING THE LOGO

The following style changes the logo in the web console header:

```
#header-logo {
  background-image: url("https://www.example.com/images/logo.png");
  width: 190px;
  height: 20px;
}
```

Replace the **example.com** URL with a URL to an actual image, and adjust the width and height. The ideal height is **20px**.

Add the stylesheet as described in [Loading Extension Scripts and Stylesheets](#).

36.7. CUSTOMIZING THE MEMBERSHIP WHITELIST

The default whitelist in the membership page shows a subset of cluster roles, such as **admin**, **basic-user**, **edit**, and so on. It also shows custom roles defined within a project.

For example, to add your own set of custom cluster roles to the whitelist:

```
window.OPENSIFT_CONSTANTS.MEMBERSHIP_WHITELIST = [
  "admin",
  "basic-user",
  "edit",
  "system:deployer",
  "system:image-builder",
  "system:image-puller",
  "system:image-pusher",
  "view",
  "custom-role-1",
  "custom-role-2"
];
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.8. CHANGING LINKS TO DOCUMENTATION

Links to external documentation are shown in various sections of the web console. The following example changes the URL for two given links to the documentation:

```
window.OPENSIFT_CONSTANTS.HELP['get_started_cli'] =
  "https://example.com/doc1.html";
window.OPENSIFT_CONSTANTS.HELP['basic_cli_operations'] =
  "https://example.com/doc2.html";
```

Alternatively, you can change the base URL for all documentation links.

This example would result in the default help URL
https://example.com/docs/welcome/index.html:

```
window.OPENSIFT_CONSTANTS.HELP_BASE_URL = "https://example.com/docs/"; 1
```

- 1 The path must end in a /.

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.9. ADDING OR CHANGING LINKS TO DOWNLOAD THE CLI

The **About** page in the web console provides download links for the [command line interface \(CLI\)](#) tools. These links can be configured by providing both the link text and URL, so that you can choose to point them directly to file packages, or to an external page that points to the actual packages.

For example, to point directly to packages that can be downloaded, where the link text is the package platform:

```
window.OPENSIFT_CONSTANTS.CLI = {
  "Linux (32 bits)": "https://<cdn>/openshift-client-tools-linux-32bit.tar.gz",
  "Linux (64 bits)": "https://<cdn>/openshift-client-tools-linux-64bit.tar.gz",
  "Windows":        "https://<cdn>/openshift-client-tools-windows.zip",
  "Mac OS X":        "https://<cdn>/openshift-client-tools-mac.zip"
};
```

Alternatively, to point to a page that links the actual download packages, with the **Latest Release** link text:

```
window.OPENSIFT_CONSTANTS.CLI = {
  "Latest Release": "https://<cdn>/openshift-client-tools/latest.html"
};
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.9.1. Customizing the About Page

To provide a custom **About** page for the web console:

1. Write an extension that looks like:

```
angular
  .module('aboutPageExtension', ['openshiftConsole'])
  .config(function($routeProvider) {
    $routeProvider
      .when('/about', {
        templateUrl:
          'https://example.com/extensions/about/about.html',
        controller: 'AboutController'
      });
  })
  );

hawtioPluginLoader.addModule('aboutPageExtension');
```

2. Write a customized template.

Start from the version of [about.html](#) from the OpenShift Container Platform [release](#) you are using. Within the template, there are two angular scope variables available: **version.master.openshift** and **version.master.kubernetes**.

3. Host the template at a URL with the correct Cross-Origin Resource Sharing (CORS) response headers for the web console.
 - a. Set **Access-Control-Allow-Origin** response to allow requests from the web console domain.
 - b. Set **Access-Control-Allow-Methods** to include **GET**.
 - c. Set **Access-Control-Allow-Headers** to include **Content-Type**.

Alternatively, you can include the template directly in your JavaScript using AngularJS [\\$templateCache](#).

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.10. CONFIGURING NAVIGATION MENUS

36.10.1. Top Navigation Dropdown Menus

The top navigation bar of the web console contains the help icon and the user dropdown menus. You can add additional menu items to these using the [angular-extension-registry](#).

The available extension points are:

- **nav-help-dropdown** - the help icon dropdown menu, visible at desktop screen widths
- **nav-user-dropdown** - the user dropdown menu, visible at desktop screen widths
- **nav-dropdown-mobile** - the single menu for top navigation items at mobile screen widths

The following example extends the **nav-help-dropdown** menu, with a name of **<myExtensionModule>**:



NOTE

<myExtensionModule> is a placeholder name. Each dropdown menu extension must be unique enough so that it does not clash with any future angular modules.

```
angular
  .module('<myExtensionModule>', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      extensionRegistry
        .add('nav-help-dropdown', function() {
          return [
            {
              type: 'dom',
              node: '<li><a href="http://www.example.com/report"
target="_blank">Report a Bug</a></li>'
            }, {
```



```

        type: 'dom',
        node: '<li class="divider"></li>' // If you want a
horizontal divider to appear in the menu
      }, {
        type: 'dom',
        node: '<li><a href="http://www.example.com/status"
target="_blank">System Status</a></li>'
      }
    ];
  });
}
]);

hawtioPluginLoader.addModule('<myExtensionModule>');

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.10.2. Application Launcher

The top navigation bar also contains an optional application launcher for linking to other web applications. This dropdown menu is empty by default, but when links are added, appears to the left of the help menu in the masthead.

```

// Add items to the application launcher dropdown menu.
window.OPENSIFT_CONSTANTS.APP_LAUNCHER_NAVIGATION = [{
  title: "Dashboard", // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "http://example.com/dashboard", // Where to go when this item is
clicked
  tooltip: 'View dashboard' // Optional tooltip to display
on hover
}, {
  title: "Manage Account",
  iconClass: "pficon pficon-user",
  href: "http://example.com/account",
  tooltip: "Update email address or password."
}];

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.10.3. System Status Badge

The top navigation bar can also include an optional system status badge in order to notify users of system-wide events such as maintenance windows. To make use of the existing styles using a yellow warning icon for the badge, follow the example below.

```

'use strict';

angular
  .module('mysystemstatusbadgeextension', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      // Replace http://status.example.com/ with your domain
      var system_status_elem = $('<a href="http://status.example.com/"' +

```

```

        'target="_blank" class="nav-item-iconic system-status"><span
title="' +
        'System Status" class="fa status-icon pficon-warning-triangle-o">' +
        '</span></a>');

// Add the extension point to the registry so the badge appears
// To disable the badge, comment this block out
extensionRegistry
    .add('nav-system-status', function() {
        return [{
            type: 'dom',
            node: system_status_elem
        }];
    });
}
]);

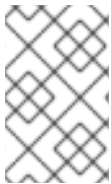
hawtioPluginLoader.addModule('mysystemstatusbadgeextension');

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.10.4. Project Left Navigation

When navigating within a project, a menu appears on the left with primary and secondary navigation. This menu structure is defined as a constant and can be overridden or modified.



NOTE

Significant customizations to the project navigation may affect the user experience and should be done with careful consideration. You may need to update this customization in future upgrades if you modify existing navigation items.

```

// Append a new primary nav item. This is a simple direct navigation item
// with no secondary menu.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.push({
    label: "Dashboard",           // The text label
    iconClass: "fa fa-dashboard", // The icon you want to appear
    href: "/dashboard"           // Where to go when this nav item is
    clicked.

                                // Relative URLs are pre-pended with the
    path

                                // '/project/<project-name>'
});

// Splice a primary nav item to a specific spot in the list. This primary
// item has
// a secondary menu.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.splice(2, 0, { // Insert at
the third spot
    label: "Git",
    iconClass: "fa fa-code",
    secondaryNavSections: [      // Instead of an href, a sub-menu can be
defined
        {
            items: [

```

```

        {
            label: "Branches",
            href: "/git/branches",
            prefixes: [
                "/git/branches/" // Defines prefix URL patterns that will
cause // this nav item to show the active
state, so // tertiary or lower pages show the
right context
            ]
        }
    ],
    {
        header: "Collaboration", // Sections within a sub-menu can have an
optional header
        items: [
            {
                label: "Pull Requests",
                href: "/git/pull-requests",
                prefixes: [
                    "/git/pull-requests/"
                ]
            }
        ]
    }
]
});

// Add a primary item to the top of the list. This primary item is shown
conditionally.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.unshift({
    label: "Getting Started",
    iconClass: "pficon pficon-screen",
    href: "/getting-started",
    prefixes: [ // Primary nav items can also specify
prefixes to trigger // active state
        "/getting-started/"
    ],
    isValid: function() { // Primary or secondary items can define
an isValid // function. If present it will be called
        return isNewUser; // to test whether
        // the item should be shown, it should
return a boolean
    }
});

// Modify an existing menu item
var applicationsMenu =
_.find(window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION, { label:
'Applications' });
applicationsMenu.secondaryNavSections.push({ // Add a new secondary nav

```

```

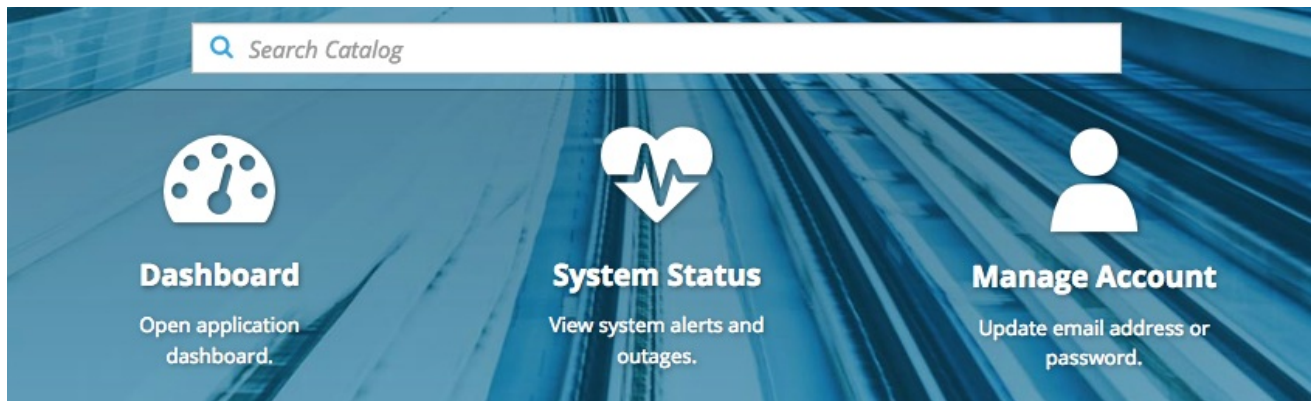
    section to the Applications menu
    // my secondary nav section
  });

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.11. CONFIGURING FEATURED APPLICATIONS

The web console has an optional list of featured application links in its landing page catalog. These appear near the top of the page and can have an icon, a title, a short description, and a link.



```

// Add featured applications to the top of the catalog.
window.OPENSIFT_CONSTANTS.SAAS_OFFERINGS = [{
  title: "Dashboard",                                // The text label
  icon: "fa fa-dashboard",                          // The icon you want to
  appear                                           appear
  url: "http://example.com/dashboard",              // Where to go when this
  item is clicked
  description: "Open application dashboard." // Short description
}, {
  title: "System Status",
  icon: "fa fa-heartbeat",
  url: "http://example.com/status",
  description: "View system alerts and outages."
}, {
  title: "Manage Account",
  icon: "pficon pficon-user",
  url: "http://example.com/account",
  description: "Update email address or password."
}
];

```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.12. CONFIGURING CATALOG CATEGORIES

Catalog categories organize the display of items in the web console catalog landing page. Each category has one or more subcategories. A builder image, template, or service is grouped in a subcategory if it includes a tag listed in the matching subcategory tags, and an item can appear in more than one subcategory. Categories and subcategories only display if they contain at least one item.

**NOTE**

Significant customizations to the catalog categories may affect the user experience and should be done with careful consideration. You may need to update this customization in future upgrades if you modify existing category items.

```
// Find the Languages category.
var category =
_.find(window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,
      { id: 'languages' });
// Add Go as a new subcategory under Languages.
category.subCategories.splice(2,0,{ // Insert at the third spot.
  // Required. Must be unique.
  id: "go",
  // Required.
  label: "Go",
  // Optional. If specified, defines a unique icon for this item.
  icon: "icon-go-gopher",
  // Required. Items matching any tag will appear in this subcategory.
  tags: [
    "go",
    "golang"
  ]
});

// Add a Featured category as the first category tab.
window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "featured",
  // Required
  label: "Featured",
  subCategories: [
    {
      // Required. Must be unique.
      id: "go",
      // Required.
      label: "Go",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-go-gopher",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "go",
        "golang"
      ]
    },
    {
      // Required. Must be unique.
      id: "jenkins",
      // Required.
      label: "Jenkins",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-jenkins",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "jenkins"
      ]
    }
  ]
});
```

```
}
  ]
});
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.13. CONFIGURING QUOTA NOTIFICATION MESSAGES

Whenever a user reaches a quota, a quota notification is put into the notification drawer. A custom quota notification message, per [quota resource type](#), can be added to the notification. For example:

```
Your project is over quota. It is using 200% of 2 cores CPU (Limit).
Upgrade
to <a href='https://www.openshift.com'>OpenShift Online Pro</a> if you
need
additional resources.
```

The "Upgrade to..." part of the notification is the custom message and may contain HTML such as links to additional resources.



NOTE

Since the quota message is HTML markup, any special characters need to be properly escaped for HTML.

Set the `window.OPENSIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE` property in an extension script to customize the message for each resource.

```
// Set custom notification messages per quota type/key
window.OPENSIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE = {
  'pods': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online
Pro</a> if you need additional resources.',
  'limits.memory': 'Upgrade to <a
href="https://www.openshift.com">OpenShift Online Pro</a> if you need
additional resources.'
};
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.14. CONFIGURING THE CREATE FROM URL NAMESPACE WHITELIST

[Create from URL](#) only works with image streams or templates from namespaces that have been explicitly specified in `OPENSIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST`. To add namespaces to the whitelist, follow these steps:



NOTE

`openshift` is included in the whitelist by default. Do not remove it.

```
// Add a namespace containing the image streams and/or templates
window.OPENSIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST.push(
```

```
'shared-stuff'
);
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.15. DISABLING THE COPY LOGIN COMMAND

The web console allows users to copy a login command, including the current access token, to the clipboard from the user menu and the Command Line Tools page. This function can be changed so that the user's access token is not included in the copied command.

```
// Do not copy the user's access token in the copy login command.
window.OPENSIFT_CONSTANTS.DISABLE_COPY_LOGIN_COMMAND = true;
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

36.15.1. Enabling Wildcard Routes

If you enabled wildcard routes for a router, you can also enable wildcard routes in the web console. This lets users enter hostnames starting with an asterisk like `*.example.com` when creating a route. To enable wildcard routes:

```
window.OPENSIFT_CONSTANTS.DISABLE_WILDCARD_ROUTES = false;
```

Add the script as described in [Loading Extension Scripts and Stylesheets](#).

[Learn how to configure HAProxy routers to allow wildcard routes](#)

36.16. CUSTOMIZING THE LOGIN PAGE

You can also change the login page, and the login provider selection page for the web console. Run the following commands to create templates you can modify:

```
$ oc adm create-login-template > login-template.html
$ oc adm create-provider-selection-template > provider-selection-
template.html
```

Edit the file to change the styles or add content, but be careful not to remove any required parameters inside the curly brackets.

To use your custom login page or provider selection page, set the following options in the master configuration file:

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.html
    providerSelection: /path/to/provider-selection-template.html
```

Relative paths are resolved relative to the master configuration file. You must restart the server after changing this configuration.

When there are multiple login providers configured or when the [alwaysShowProviderSelection](#) option in the *master-config.yaml* file is set to **true**, each time a user's token to OpenShift Container Platform expires, the user is presented with this custom page before they can proceed with other tasks.

36.16.1. Example Usage

Custom login pages can be used to create Terms of Service information. They can also be helpful if you use a third-party login provider, like GitHub or Google, to show users a branded page that they trust and expect before being redirected to the authentication provider.

36.17. CUSTOMIZING THE OAUTH ERROR PAGE

When errors occur during authentication, you can change the page shown.

1. Run the following command to create a template you can modify:

```
$ oc adm create-error-template > error-template.html
```

2. Edit the file to change the styles or add content.

You can use the **Error** and **ErrorCode** variables in the template. To use your custom error page, set the following option in the master configuration file:

```
oauthConfig:
  ...
  templates:
    error: /path/to/error-template.html
```

Relative paths are resolved relative to the master configuration file.

3. You must restart the server after changing this configuration.

36.18. CHANGING THE LOGOUT URL

You can change the location a console user is sent to when logging out of the console by modifying the **clusterInfo.logoutPublicURL** parameter in the **webconsole-config** ConfigMap.

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

Here is an example that changes the logout URL to <https://www.example.com/logout>:

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    clusterInfo:
      [...]
      logoutPublicURL: "https://www.example.com/logout"
      [...]
```

This can be useful when authenticating with [Request Header](#) and OAuth or [OpenID](#) identity providers, which require visiting an external URL to destroy single sign-on sessions.

36.19. CONFIGURING WEB CONSOLE CUSTOMIZATIONS WITH ANSIBLE

During cluster installations, many modifications to the web console can be configured using the following parameters, which are configurable in the [inventory file](#):

- `openshift_master_logout_url`
- `openshift_web_console_extension_script_urls`
- `openshift_web_console_extension_stylesheet_urls`
- `openshift_master_oauth_templates`
- `openshift_master_metrics_public_url`
- `openshift_master_logging_public_url`

Example Web Console Customization with Ansible

```
# Configure `clusterInfo.logoutPublicURL` in the web console configuration
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#changing-the-logout-url
#openshift_master_logout_url=https://example.com/logout

# Configure extension scripts for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-custom-scripts-and-stylesheets
#openshift_web_console_extension_script_urls=
['https://example.com/scripts/menu-customization.js', 'https://example.com/scripts/nav-customization.js']

# Configure extension stylesheets for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-custom-scripts-and-stylesheets
#openshift_web_console_extension_stylesheet_urls=
['https://example.com/styles/logo.css', 'https://example.com/styles/custom-styles.css']

# Configure a custom login template in the master config
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#customizing-the-login-page
#openshift_master_oauth_templates={'login': '/path/to/login-template.html'}

# Configure `clusterInfo.metricsPublicURL` in the web console configuration for cluster metrics. Ansible is also able to configure metrics for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/cluster_metrics.html
#openshift_master_metrics_public_url=https://hawkular-
```

```
metrics.example.com/hawkular/metrics

# Configure `clusterInfo.loggingPublicURL` in the web console
configuration for aggregate logging. Ansible is also able to install
logging for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/aggregate_logging.html
#openshift_master_logging_public_url=https://kibana.example.com
```

36.20. CHANGING THE WEB CONSOLE URL PORT AND CERTIFICATES

To ensure your custom certificate is served when users access the web console URL, add the certificate and URL to the **namedCertificates** section of the *master-config.yaml* file. See [Configuring Custom Certificates for the Web Console or CLI](#) for more information.

To set or modify the redirect URL for the web console, modify the **openshift-web-console oauthclient**:

```
$ oc edit oauthclient openshift-web-console
```

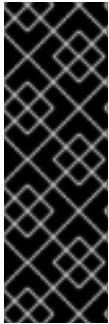
To ensure users are correctly redirected, update the **PublicUrls** for the **openshift-web-console configmap**:

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

Then, update the value for **consolePublicURL**.

CHAPTER 37. DEPLOYING EXTERNAL PERSISTENT VOLUME PROVISIONERS

37.1. OVERVIEW



IMPORTANT

The external provisioner for AWS EFS on OpenShift Container Platform is a Technology Preview feature. Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete, and Red Hat does not recommend using them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information, see [Red Hat Technology Preview Features Support Scope](#).

An external provisioner is an application that enables dynamic provisioning for a particular storage provider. External provisioners can run alongside the provisioner plug-ins provided by OpenShift Container Platform and are configured in a similar way as the **StorageClass** objects are configured, as described in the [Dynamic Provisioning and Creating Storage Classes](#) section. Since these provisioners are external, you can deploy and update them independently of OpenShift Container Platform.

37.2. BEFORE YOU BEGIN

An [Ansible Playbook](#) is also available to deploy and upgrade external provisioners.



NOTE

Before proceeding, familiarize yourself with the [Configuring Cluster Metrics](#) and the [Configuring Cluster Logging](#) sections.

37.2.1. External Provisioners Ansible Role

The OpenShift Ansible **openshift_provisioners** role configures and deploys external provisioners using the variables from the [Ansible](#) inventory file. You must specify which provisioners to install by overriding their respective **install** variables to **true**.

37.2.2. External Provisioners Ansible Variables

Following is a list of role variables that apply to all provisioners for which the **install** variable is **true**.

Table 37.1. Ansible Variables

Variable	Description
openshift_provisioners_install_provisioners	If true , deploy all provisioners that have their respective install variables set as true , otherwise, remove them.

Variable	Description
<code>openshift_provisioners_image_prefix</code>	The prefix for the component images. For example, with <code>openshift3/efs-provisioner:v3.6</code> , set prefix <code>openshift3/</code> .
<code>openshift_provisioners_image_version</code>	The version for the component images. For example, with <code>openshift3/efs-provisioner:v3.6</code> , set version as <code>v3.6</code> .
<code>openshift_provisioners_project</code>	The project to deploy provisioners in. Defaults to <code>openshift-infra</code> .

37.2.3. AWS EFS Provisioner Ansible Variables

The AWS EFS provisioner dynamically provisions [NFS PVs](#) backed by dynamically created directories in a given EFS file system's directory. You must satisfy the following requirements before the AWS EFS Provisioner Ansible variables can be configured:

- An IAM user assigned with the ***AmazonElasticFileSystemReadOnlyAccess*** policy (or better).
- An EFS file system in your cluster's region.
- [Mount targets](#) and [security groups](#) such that any node (in any zone in the cluster's region) can mount the EFS file system by its [File system DNS name](#).

Table 37.2. Required EFS Ansible Variables

Variable	Description
<code>openshift_provisioners_efs_fsid</code>	The File system ID of the EFS file system, for example: <code>fs-47a2c22e</code>
<code>openshift_provisioners_efs_region</code>	The Amazon EC2 region for the EFS file system.
<code>openshift_provisioners_efs_aws_access_key_id</code>	The AWS access key of the IAM user (to check that the specified EFS file system exists).
<code>openshift_provisioners_efs_aws_secret_access_key</code>	The AWS secret access key of the IAM user (to check that the specified EFS file system exists).

Table 37.3. Optional EFS Ansible Variables

Variable	Description
<code>openshift_provisioners_efs</code>	If true , the AWS EFS provisioner is installed or uninstalled according to whether <code>openshift_provisioners_install_provisioners</code> is true or false , respectively. Defaults to false .

Variable	Description
openshift_provisioners_efs_path	The path of the directory in the EFS file system, in which the EFS provisioner will create a directory to back each PV it creates. It must exist and be mountable by the EFS provisioner. Defaults to /persistentvolumes .
openshift_provisioners_efs_name	The provisioner name that StorageClasses specify. Defaults to openshift.org/aws-efs .
openshift_provisioners_efs_nodeselector	A map of labels to select the nodes where the pod will land. For example: {"node": "infra", "region": "west"} .
openshift_provisioners_efs_supplementalgroup	The supplemental group to give the pod, in case it is needed for permission to write to the EFS file system. Defaults to 65534 .

37.3. DEPLOYING THE PROVISIONERS

You can deploy all provisioners at once or one provisioner at a time according to the configuration specified in the OpenShift Ansible variables. The following example shows you how to deploy a given provisioner and then create and configure a corresponding [StorageClass](#).

37.3.1. Deploying the AWS EFS Provisioner

The following command sets the directory in the EFS volume to **/data/persistentvolumes**. This directory must exist in the file system and must be mountable and writeable by the provisioner pod.

```
$ ansible-playbook -v -i <inventory_file> \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
provisioners/config.yml \
  -e openshift_provisioners_install_provisioners=True \
  -e openshift_provisioners_efs=True \
  -e openshift_provisioners_efs_fs_id=fs-47a2c22e \
  -e openshift_provisioners_efs_region=us-west-2 \
  -e openshift_provisioners_efs_aws_access_key_id=AKIAIOSFODNN7EXAMPLE \
  -e
openshift_provisioners_efs_aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPx
RfiCYEXAMPLEKEY \
  -e openshift_provisioners_efs_path=/data/persistentvolumes
```

37.3.1.1. AWS EFS Object Definition

aws-efs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
```

```

    name: slow
  provisioner: openshift.org/aws-efs 1
  parameters:
    gidMin: "40000" 2
    gidMax: "50000" 3

```

- 1** Set this value same as the value of **openshift_provisioners_efs_name** variable, which defaults to **openshift.org/aws-efs**.
- 2** The minimum value of GID range for the **StorageClass**. (Optional)
- 3** The maximum value of GID range for the **StorageClass**. (Optional)

Each dynamically provisioned volume's corresponding NFS directory is assigned a unique GID owner from the range **gidMin-gidMax**. If it is not specified, **gidMin** defaults to **2000** and **gidMax** defaults to **2147483647**. Any pod that consumes a provisioned volume via a claim automatically runs with the needed GID as a supplemental group and is able to read & write to the volume. Other mounters that do not have the supplemental group (and are not running as root) will not be able to read or write to the volume. For more information on using the supplemental groups to manage NFS access, see the [Group IDs](#) section of NFS Volume Security topic.

37.4. CLEANUP

You can remove everything deployed by the OpenShift Ansible **openshift_provisioners** role by running the following command:

```

$ ansible-playbook -v -i <inventory_file> \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
  provisioners/config.yml \
  -e openshift_provisioners_install_provisioners=False

```