



OpenShift Container Platform 3.7

Cluster Administration

OpenShift Container Platform 3.7 Cluster Administration

OpenShift Container Platform 3.7 Cluster Administration

OpenShift Container Platform 3.7 Cluster Administration

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 Cluster Administration topics cover the day to day tasks for managing your OpenShift cluster and other advanced configuration topics.

Table of Contents

CHAPTER 1. OVERVIEW	13
CHAPTER 2. MANAGING NODES	14
2.1. OVERVIEW	14
2.2. LISTING NODES	14
2.3. VIEWING NODES	16
2.4. ADDING NODES	17
2.5. DELETING NODES	17
2.6. UPDATING LABELS ON NODES	17
2.7. LISTING PODS ON NODES	18
2.8. MARKING NODES AS UNSCHEDULABLE OR SCHEDULABLE	18
2.9. EVACUATING PODS ON NODES	18
2.10. REBOOTING NODES	19
2.10.1. Infrastructure nodes	19
2.10.2. Using pod anti-affinity	20
2.10.3. Handling nodes running routers	21
2.11. CONFIGURING NODE RESOURCES	21
2.11.1. Setting maximum pods per node	22
2.12. RESETTING DOCKER STORAGE	22
2.13. CHANGING NODE TRAFFIC INTERFACE	24
CHAPTER 3. RESTORING OPENSIFT CONTAINER PLATFORM COMPONENTS	25
3.1. OVERVIEW	25
3.2. RESTORING A CLUSTER	25
Procedure	25
3.3. RESTORING A MASTER HOST BACKUP	25
Procedure	25
3.4. RESTORING A NODE HOST BACKUP	26
Procedure	27
3.5. RESTORING ETCD	28
3.5.1. Restoring etcd v2 & v3 data	28
Procedure	28
3.5.1.1. Fix the peerURLS parameter	30
3.5.1.1.1. Procedure	30
3.5.2. Restoring etcd for v3	30
Procedure	30
3.6. ADDING AN ETCD NODE	31
3.6.1. Adding a new etcd host using Ansible	31
Procedure	31
3.6.2. Manually adding a new etcd host	32
Procedure	32
Modify the current etcd cluster	32
Modify the new etcd host	35
Modify each OpenShift Container Platform master	37
3.7. BRINGING OPENSIFT CONTAINER PLATFORM SERVICES BACK ONLINE	38
Procedure	38
3.8. RESTORING A PROJECT	39
Procedure	39
3.9. RESTORING APPLICATION DATA	39
Procedure	40
3.10. RESTORING PERSISTENT VOLUME CLAIMS	40
3.10.1. Restoring files to an existing PVC	41

Procedure	41
3.10.2. Restoring data to a new PVC	41
Procedure	41
CHAPTER 4. REPLACING A MASTER HOST	43
4.1. DEPRECATING A MASTER HOST	43
Procedure	43
4.2. ADDING HOSTS	44
Procedure	45
4.3. SCALING ETCD	47
Prerequisites	47
4.3.1. Adding a new etcd host using Ansible	48
Procedure	48
4.3.2. Manually adding a new etcd host	49
Procedure	49
Modify the current etcd cluster	49
Modify the new etcd host	52
Modify each OpenShift Container Platform master	54
CHAPTER 5. MANAGING USERS	56
5.1. OVERVIEW	56
5.2. CREATING A USER	56
5.3. VIEWING USER AND IDENTITY LISTS	56
5.4. CREATING GROUPS	57
5.5. MANAGING USER AND GROUP LABELS	57
5.6. DELETING A USER	58
CHAPTER 6. MANAGING PROJECTS	59
6.1. OVERVIEW	59
6.2. SELF-PROVISIONING PROJECTS	59
6.2.1. Modifying the Template for New Projects	59
6.2.2. Disabling Self-provisioning	60
6.3. USING NODE SELECTORS	61
6.3.1. Setting the Cluster-wide Default Node Selector	61
6.3.2. Setting the Project-wide Node Selector	61
6.3.3. Developer-specified Node Selectors	62
6.4. LIMITING NUMBER OF SELF-PROVISIONED PROJECTS PER USER	62
CHAPTER 7. MANAGING PODS	64
7.1. OVERVIEW	64
7.2. VIEWING PODS	64
7.3. LIMITING RUN-ONCE POD DURATION	64
7.3.1. Configuring the RunOnceDuration Plug-in	64
7.3.2. Specifying a Custom Duration per Project	65
7.3.2.1. Deploying an Egress Router Pod	65
7.3.2.2. Deploying an Egress Router Service	66
7.3.3. Limiting Pod Access with Egress Firewall	67
7.3.3.1. Configuring Pod Access Limits	67
7.4. LIMITING THE BANDWIDTH AVAILABLE TO PODS	69
7.5. SETTING POD DISRUPTION BUDGETS	69
7.6. INJECTING INFORMATION INTO PODS USING POD PRESETS	70
CHAPTER 8. MANAGING NETWORKING	72
8.1. OVERVIEW	72

8.2. MANAGING POD NETWORKS	72
8.2.1. Joining Project Networks	72
8.3. ISOLATING PROJECT NETWORKS	72
8.3.1. Making Project Networks Global	72
8.4. DISABLING HOST NAME COLLISION PREVENTION FOR ROUTES AND INGRESS OBJECTS	73
8.5. CONTROLLING EGRESS TRAFFIC	74
8.5.1. Using an Egress Firewall to Limit Access to External Resources	75
8.5.2. Using an Egress Router to Allow External Resources to Recognize Pod Traffic	77
8.5.2.1. Deploying an Egress Router Pod in Redirect Mode	78
8.5.2.2. Redirecting to Multiple Destinations	80
8.5.2.3. Using a ConfigMap to specify EGRESS_DESTINATION	81
8.5.2.4. Deploying an Egress Router HTTP Proxy Pod	82
8.5.2.5. Enabling Failover for Egress Router Pods	84
8.5.3. Using iptables Rules to Limit Access to External Resources	85
8.6. ENABLING STATIC IPS FOR EXTERNAL PROJECT TRAFFIC	86
8.7. ENABLING MULTICAST	87
8.8. ENABLING NETWORKPOLICY	88
8.8.1. NetworkPolicy and Routers	89
8.8.2. Setting a Default NetworkPolicy for New Projects	90
8.9. ENABLING HTTP STRICT TRANSPORT SECURITY	91
8.10. TROUBLESHOOTING THROUGHPUT ISSUES	92
CHAPTER 9. CONFIGURING SERVICE ACCOUNTS	94
9.1. OVERVIEW	94
9.2. USER NAMES AND GROUPS	94
9.3. MANAGING SERVICE ACCOUNTS	95
9.4. ENABLING SERVICE ACCOUNT AUTHENTICATION	95
9.5. MANAGED SERVICE ACCOUNTS	96
9.6. INFRASTRUCTURE SERVICE ACCOUNTS	97
9.7. SERVICE ACCOUNTS AND SECRETS	97
CHAPTER 10. MANAGING ROLE-BASED ACCESS CONTROL (RBAC)	98
10.1. OVERVIEW	98
10.2. VIEWING ROLES AND BINDINGS	98
10.2.1. Viewing cluster roles	98
10.2.2. Viewing cluster role bindings	103
10.2.3. Viewing local roles and bindings	107
10.3. MANAGING ROLE BINDINGS	108
10.4. GRANTING USERS DAEMONSET PERMISSIONS	110
10.5. CREATING A LOCAL ROLE	110
10.6. CREATING A CLUSTER ROLE	111
10.7. CLUSTER AND LOCAL ROLE BINDINGS	111
CHAPTER 11. IMAGE POLICY	112
11.1. OVERVIEW	112
11.2. CONFIGURING THE IMAGEPOLICY ADMISSION PLUG-IN	112
11.3. TESTING THE IMAGEPOLICY ADMISSION PLUG-IN	114
CHAPTER 12. IMAGE SIGNATURES	116
12.1. OVERVIEW	116
12.2. SIGNING IMAGES USING ATOMIC CLI	116
12.3. VERIFYING IMAGE SIGNATURES USING OPENSIFT CLI	117
12.4. ACCESSING IMAGE SIGNATURES USING REGISTRY API	118
12.4.1. Writing Image Signatures via API	118

12.4.2. Reading Image Signatures via API	119
12.4.3. Importing Image Signatures Automatically from Signature Stores	119
CHAPTER 13. SCOPED TOKENS	121
13.1. OVERVIEW	121
13.2. EVALUATION	121
13.3. USER SCOPES	121
13.4. ROLE SCOPE	121
CHAPTER 14. MONITORING IMAGES	122
14.1. OVERVIEW	122
14.2. VIEWING IMAGES STATISTICS	122
14.3. VIEWING IMAGESTREAMS STATISTICS	122
14.4. PRUNING IMAGES	123
CHAPTER 15. MANAGING SECURITY CONTEXT CONSTRAINTS	124
15.1. OVERVIEW	124
15.2. LISTING SECURITY CONTEXT CONSTRAINTS	124
15.3. EXAMINING A SECURITY CONTEXT CONSTRAINTS OBJECT	124
15.4. CREATING NEW SECURITY CONTEXT CONSTRAINTS	125
15.5. DELETING SECURITY CONTEXT CONSTRAINTS	126
15.6. UPDATING SECURITY CONTEXT CONSTRAINTS	127
15.6.1. Example Security Context Constraints Settings	127
15.7. UPDATING THE DEFAULT SECURITY CONTEXT CONSTRAINTS	128
15.8. HOW DO I?	128
15.8.1. Grant Access to the Privileged SCC	128
15.8.2. Grant a Service Account Access to the Privileged SCC	129
15.8.3. Enable Images to Run with USER in the Dockerfile	129
15.8.4. Enable Container Images that Require Root	130
15.8.5. Use --mount-host on the Registry	130
15.8.6. Provide Additional Capabilities	130
15.8.7. Modify Cluster Default Behavior	131
15.8.8. Use the hostPath Volume Plug-in	131
15.8.9. Ensure That Admission Attempts to Use a Specific SCC First	132
15.8.10. Add an SCC to a User, Group, or Project	132
CHAPTER 16. SCHEDULING	133
16.1. OVERVIEW	133
16.1.1. Overview	133
16.1.2. Default scheduling	133
16.1.3. Advanced scheduling	133
16.1.4. Custom scheduling	133
16.2. DEFAULT SCHEDULING	133
16.2.1. Overview	133
16.2.2. Generic Scheduler	133
16.2.3. Filter the Nodes	134
16.2.3.1. Prioritize the Filtered List of Nodes	134
16.2.3.2. Select the Best Fit Node	134
16.2.4. Scheduler Policy	134
16.2.4.1. Modifying Scheduler Policy	136
16.2.5. Available Predicates	137
16.2.5.1. Static Predicates	137
16.2.5.1.1. Default Predicates	137
16.2.5.1.2. Other Supported Predicates	138

16.2.5.2. General Predicates	139
Non-critical general predicates	139
Essential general predicates	139
16.2.5.3. Configurable Predicates	140
16.2.6. Available Priorities	141
16.2.6.1. Static Priorities	142
16.2.6.1.1. Default Priorities	142
16.2.6.1.2. Other Priorities	143
16.2.6.2. Configurable Priorities	143
16.2.7. Use Cases	145
16.2.7.1. Infrastructure Topological Levels	145
16.2.7.2. Affinity	145
16.2.7.3. Anti Affinity	145
16.2.8. Sample Policy Configurations	145
16.3. CUSTOM SCHEDULING	148
16.3.1. Overview	148
16.3.2. Deploying the Scheduler	148
16.4. CONTROLLING POD PLACEMENT	150
16.4.1. Overview	150
16.4.2. Constraining Pod Placement Using Node Name	150
16.4.3. Constraining Pod Placement Using a Node Selector	150
16.4.4. Control Pod Placement to Projects	152
16.5. ADVANCED SCHEDULING	154
16.5.1. Overview	154
16.5.2. Using Advanced Scheduling	155
16.6. ADVANCED SCHEDULING AND NODE AFFINITY	156
16.6.1. Overview	156
16.6.2. Configuring Node Affinity	156
16.6.2.1. Configuring a Required Node Affinity Rule	158
16.6.2.2. Configuring a Preferred Node Affinity Rule	158
16.6.3. Examples	159
16.6.3.1. Node Affinity with Matching Labels	159
16.6.3.2. Node Affinity with No Matching Labels	160
16.7. ADVANCED SCHEDULING AND POD AFFINITY AND ANTI-AFFINITY	161
16.7.1. Overview	161
16.7.2. Configuring Pod Affinity and Anti-affinity	161
16.7.2.1. Configuring an Affinity Rule	163
16.7.2.2. Configuring an Anti-affinity Rule	164
16.7.3. Examples	165
16.7.3.1. Pod Affinity	165
16.7.3.2. Pod Anti-affinity	166
16.7.3.3. Pod Affinity with no Matching Labels	167
16.8. ADVANCED SCHEDULING AND NODE SELECTORS	167
16.8.1. Overview	167
16.8.2. Configuring Node Selectors	168
16.9. ADVANCED SCHEDULING AND TAINTS AND TOLERATIONS	169
16.9.1. Overview	169
16.9.2. Taints and Tolerations	169
16.9.2.1. Using Multiple Taints	170
16.9.3. Adding a Taint to an Existing Node	171
16.9.4. Adding a Toleration to a Pod	171
16.9.4.1. Using Toleration Seconds to Delay Pod Evictions	172
16.9.4.1.1. Setting a Default Value for Toleration Seconds	172

16.9.5. Preventing Pod Eviction for Node Problems	173
16.9.6. Daemonsets and Tolerations	174
16.9.7. Examples	174
16.9.7.1. Dedicating a Node for a User	175
16.9.7.2. Binding a User to a Node	175
16.9.7.3. Nodes with Special Hardware	175
CHAPTER 17. SETTING QUOTAS	176
17.1. OVERVIEW	176
17.2. RESOURCES MANAGED BY QUOTA	176
17.3. QUOTA SCOPES	177
17.4. QUOTA ENFORCEMENT	178
17.5. REQUESTS VERSUS LIMITS	178
17.6. SAMPLE RESOURCE QUOTA DEFINITIONS	179
17.7. CREATING A QUOTA	182
17.8. VIEWING A QUOTA	182
17.9. CONFIGURING QUOTA SYNCHRONIZATION PERIOD	183
17.10. ACCOUNTING FOR QUOTA IN DEPLOYMENT CONFIGURATIONS	183
17.11. REQUIRE EXPLICIT QUOTA TO CONSUME A RESOURCE	183
17.12. KNOWN ISSUES	184
CHAPTER 18. SETTING MULTI-PROJECT QUOTAS	185
18.1. OVERVIEW	185
18.2. SELECTING PROJECTS	185
18.3. VIEWING APPLICABLE CLUSTERRESOURCEQUOTAS	186
18.4. SELECTION GRANULARITY	187
CHAPTER 19. SETTING LIMIT RANGES	188
19.1. OVERVIEW	188
19.1.1. Container Limits	189
19.1.2. Pod Limits	190
19.1.3. Image Limits	191
19.1.4. Image Stream Limits	192
19.1.4.1. Counting of Image References	192
19.1.5. PersistentVolumeClaim Limits	193
19.2. CREATING A LIMIT RANGE	193
19.3. VIEWING LIMITS	194
19.4. DELETING LIMITS	194
CHAPTER 20. PRUNING OBJECTS	195
20.1. OVERVIEW	195
20.2. BASIC PRUNE OPERATIONS	195
20.3. PRUNING DEPLOYMENTS	195
20.4. PRUNING BUILDS	196
20.5. PRUNING IMAGES	197
20.5.1. Image Prune Conditions	198
20.5.2. Using Secure or Insecure Connections	199
20.5.3. Image Pruning Problems	200
Images Not Being Pruned	200
Using a Secure Connection Against Insecure Registry	200
20.5.3.1. Using an Insecure Connection Against a Secured Registry	201
Using the Wrong Certificate Authority	201
20.6. HARD PRUNING THE REGISTRY	201
20.7. PRUNING CRON JOBS	204

CHAPTER 21. EXTENDING THE KUBERNETES API WITH CUSTOM RESOURCES	206
21.1. CREATING CUSTOM RESOURCE DEFINITIONS	206
21.2. CREATE CUSTOM OBJECTS	207
21.3. MANAGE CUSTOM OBJECTS	208
21.4. FINALIZERS	209
CHAPTER 22. GARBAGE COLLECTION	210
22.1. OVERVIEW	210
22.2. CONTAINER GARBAGE COLLECTION	210
22.2.1. Detecting Containers for Deletion	211
22.3. IMAGE GARBAGE COLLECTION	211
22.3.1. Detecting Images for Deletion	212
CHAPTER 23. ALLOCATING NODE RESOURCES	213
23.1. OVERVIEW	213
23.2. CONFIGURING NODES FOR ALLOCATED RESOURCES	213
23.3. COMPUTING ALLOCATED RESOURCES	213
23.4. VIEWING NODE ALLOCATABLE RESOURCES AND CAPACITY	214
23.5. SYSTEM RESOURCES REPORTED BY NODE	214
23.6. NODE ENFORCEMENT	215
23.7. EVICTION THRESHOLDS	216
23.8. SCHEDULER	217
CHAPTER 24. OPAQUE INTEGER RESOURCES	218
24.1. OVERVIEW	218
24.2. CREATING OPAQUE INTEGER RESOURCES	218
CHAPTER 25. OVERCOMMITTING	221
25.1. OVERVIEW	221
25.2. REQUESTS AND LIMITS	221
25.2.1. Tune Buffer Chunk Limit	221
25.3. COMPUTE RESOURCES	222
25.3.1. CPU	222
25.3.2. Memory	222
25.4. QUALITY OF SERVICE CLASSES	223
25.5. CONFIGURING MASTERS FOR OVERCOMMITMENT	223
25.6. CONFIGURING NODES FOR OVERCOMMITMENT	224
25.6.1. Reserving Memory Across Quality of Service Tiers	225
25.6.2. Enforcing CPU Limits	225
25.6.3. Reserving Resources for System Processes	226
25.6.4. Kernel Tunable Flags	227
25.6.5. Disabling Swap Memory	227
CHAPTER 26. ASSIGNING UNIQUE EXTERNAL IPS FOR INGRESS TRAFFIC	229
26.1. OVERVIEW	229
26.2. RESTRICTIONS	229
26.3. CONFIGURING THE CLUSTER TO USE UNIQUE EXTERNAL IPS	230
26.3.1. Configuring an Ingress IP for a Service	230
26.4. ROUTING THE INGRESS CIDR FOR DEVELOPMENT OR TESTING	231
26.4.1. Service externalIPs	231
CHAPTER 27. HANDLING OUT OF RESOURCE ERRORS	233
27.1. OVERVIEW	233
27.2. CONFIGURING EVICTION POLICIES	233
27.2.1. Using the Node Configuration to Create a Policy	234

27.2.2. Understanding Eviction Signals	235
27.2.3. Understanding Eviction Thresholds	237
27.2.3.1. Understanding Hard Eviction Thresholds	238
27.2.3.1.1. Default Hard Eviction Thresholds	238
27.2.3.2. Understanding Soft Eviction Thresholds	238
27.3. CONFIGURING THE AMOUNT OF RESOURCE FOR SCHEDULING	239
27.4. CONTROLLING NODE CONDITION OSCILLATION	240
27.5. RECLAIMING NODE-LEVEL RESOURCES	240
With Imagefs	241
Without Imagefs	241
27.6. UNDERSTANDING POD EVICTION	241
27.6.1. Understanding Quality of Service and Out of Memory Killer	242
27.7. UNDERSTANDING THE POD SCHEDULER AND OOR CONDITIONS	243
27.8. EXAMPLE SCENARIO	243
27.9. RECOMMENDED PRACTICE	244
27.9.1. DaemonSets and Out of Resource Handling	244
CHAPTER 28. MONITORING AND DEBUGGING ROUTERS	245
28.1. OVERVIEW	245
28.2. VIEWING STATISTICS	245
28.3. DISABLING STATISTICS VIEW	245
28.4. VIEWING LOGS	245
28.5. VIEWING THE ROUTER INTERNALS	246
CHAPTER 29. HIGH AVAILABILITY	247
29.1. OVERVIEW	247
29.2. CONFIGURING IP FAILOVER	248
29.2.1. Virtual IP Addresses	249
29.2.2. Check and Notify Scripts	249
29.2.3. VRRP Preemption	251
29.2.4. Keepalived Multicast	251
29.2.5. Command Line Options and Environment Variables	252
29.2.6. VRRP ID Offset	254
29.2.7. Configuring a Highly-available Service	254
29.2.7.1. Deploy IP Failover Pod	256
29.2.8. Dynamically Updating Virtual IPs for a Highly-available Service	256
29.3. CONFIGURING SERVICE EXTERNALIP AND NODEPORT	257
29.4. HIGH AVAILABILITY FOR INGRESSIP	257
CHAPTER 30. IPTABLES	258
30.1. OVERVIEW	258
30.2. IPTABLES	258
30.3. IPTABLES.SERVICE	258
CHAPTER 31. SECURING BUILDS BY STRATEGY	260
31.1. OVERVIEW	260
31.2. DISABLING A BUILD STRATEGY GLOBALLY	260
31.3. RESTRICTING BUILD STRATEGIES TO A USER GLOBALLY	261
31.4. RESTRICTING BUILD STRATEGIES TO A USER WITHIN A PROJECT	262
CHAPTER 32. RESTRICTING APPLICATION CAPABILITIES USING SECCOMP	263
32.1. OVERVIEW	263
32.2. ENABLING SECCOMP	263
32.3. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR SECCOMP	263

32.4. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR A CUSTOM SECCOMP PROFILE	264
CHAPTER 33. SYSCTLS	265
33.1. OVERVIEW	265
33.2. UNDERSTANDING SYSCTLS	265
33.3. NAMESPACE VERSUS NODE-LEVEL SYSCTLS	265
33.4. SAFE VERSUS UNSAFE SYSCTLS	266
33.5. ENABLING UNSAFE SYSCTLS	266
33.6. SETTING SYSCTLS FOR A POD	267
CHAPTER 34. ENCRYPTING DATA AT DATASTORE LAYER	268
34.1. OVERVIEW	268
34.2. CONFIGURATION AND DETERMINING WHETHER ENCRYPTION IS ALREADY ENABLED	268
34.3. UNDERSTANDING THE ENCRYPTION CONFIGURATION	268
34.3.1. Available Providers	269
34.4. ENCRYPTING DATA	270
34.5. VERIFYING THAT DATA IS ENCRYPTED	271
34.6. ENSURE ALL SECRETS ARE ENCRYPTED	272
34.7. ROTATING A DECRYPTION KEY	272
34.8. DECRYPTING DATA	272
CHAPTER 35. ENCRYPTING HOSTS WITH IPSEC	274
35.1. OVERVIEW	274
35.2. ENCRYPTING HOSTS	274
35.2.1. Step 1: Prerequisites	274
35.2.2. Step 2: Certificates	274
35.2.3. Step 3: libreswan IPsec Policy	275
35.2.3.1. Opportunistic Group Configuration	275
35.2.3.2. Explicit Connection Configuration	276
35.3. IPSEC FIREWALL CONFIGURATION	277
35.4. STARTING AND ENABLING IPSEC	277
35.5. OPTIMIZING IPSEC	278
35.6. TROUBLESHOOTING	278
CHAPTER 36. BUILDING DEPENDENCY TREES	279
36.1. OVERVIEW	279
36.2. USAGE	279
CHAPTER 37. REPLACING A FAILED ETCD MEMBER	280
37.1. REMOVING A FAILED ETCD NODE	280
Procedure	280
37.2. ADDING AN ETCD MEMBER	280
37.2.1. Adding a new etcd host using Ansible	280
Procedure	280
37.2.2. Manually adding a new etcd host	281
Procedure	281
Modify the current etcd cluster	281
Modify the new etcd host	284
Modify each OpenShift Container Platform master	286
CHAPTER 38. RESTORING ETCD QUORUM	288
38.1. BACKING UP ETCD	288
38.1.1. Backing up etcd configuration files	288
Procedure	288
38.1.2. Backing up etcd data	289

Prerequisites	289
Procedure	289
38.2. REMOVING AN ETCD HOST	290
Procedure	290
Procedure	291
38.3. CREATING A SINGLE-NODE ETCD CLUSTER	293
Procedure	293
CHAPTER 39. ADDING ETCD NODES AFTER RESTORING	294
Procedure	294
CHAPTER 40. TROUBLESHOOTING OPENSIFT SDN	296
40.1. OVERVIEW	296
40.2. NOMENCLATURE	296
40.3. DEBUGGING EXTERNAL ACCESS TO AN HTTP SERVICE	297
40.4. DEBUGGING THE ROUTER	298
40.5. DEBUGGING A SERVICE	299
40.6. DEBUGGING NODE TO NODE NETWORKING	300
40.7. DEBUGGING LOCAL NETWORKING	301
40.7.1. The Interfaces on a Node	302
40.7.2. SDN Flows Inside a Node	302
40.7.3. Debugging Steps	302
40.7.3.1. Is IP Forwarding Enabled?	302
40.7.3.2. Are your routes correct?	302
40.7.4. Is the Open vSwitch configured correctly?	303
40.7.4.1. Is the iptables configuration correct?	304
40.7.4.2. Is your external network correct?	304
40.8. DEBUGGING VIRTUAL NETWORKING	304
40.8.1. Builds on a Virtual Network are Failing	304
40.9. DEBUGGING POD EGRESS	305
40.10. READING THE LOGS	305
40.11. DEBUGGING KUBERNETES	305
40.12. FINDING NETWORK ISSUES USING THE DIAGNOSTICS TOOL	306
40.13. MISCELLANEOUS NOTES	306
40.13.1. Other clarifications on ingress	306
40.13.2. TLS Handshake Timeout	306
40.13.3. Other debugging notes	307
CHAPTER 41. DIAGNOSTICS TOOL	308
41.1. OVERVIEW	308
41.2. USING THE DIAGNOSTICS TOOL	308
41.3. RUNNING DIAGNOSTICS IN A SERVER ENVIRONMENT	310
41.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT	311
41.5. ANSIBLE-BASED HEALTH CHECKS	311
41.5.1. Running Health Checks via ansible-playbook	314
41.5.2. Running Health Checks via Docker CLI	314
CHAPTER 42. IDLING APPLICATIONS	316
42.1. OVERVIEW	316
42.2. IDLING APPLICATIONS	316
42.2.1. Idling Single Services	316
42.2.2. Idling Multiple Services	316
42.3. UNIDLING APPLICATIONS	316

CHAPTER 43. ANALYZING CLUSTER CAPACITY	318
43.1. OVERVIEW	318
43.2. RUNNING CLUSTER CAPACITY ANALYSIS ON THE COMMAND LINE	318
43.3. RUNNING CLUSTER CAPACITY AS A JOB INSIDE OF A POD	319
CHAPTER 44. REVISION HISTORY: CLUSTER ADMINISTRATION	322
44.1. TUES MAR 06 2018	322
44.2. FRI FEB 23 2018	322
44.3. WED FEB 21 2018	322
44.4. FRI FEB 16 2018	322
44.5. TUE FEB 06 2018	323
44.6. FRI DEC 22 2017	323
44.7. WED NOV 29 2017	323

CHAPTER 1. OVERVIEW

These Cluster Administration topics cover the day-to-day tasks for managing your OpenShift Container Platform cluster and other advanced configuration topics.

CHAPTER 2. MANAGING NODES

2.1. OVERVIEW

You can manage [nodes](#) in your instance using the [CLI](#).

When you perform node management operations, the CLI interacts with [node objects](#) that are representations of actual node hosts. The [master](#) uses the information from node objects to validate nodes with [health checks](#).

2.2. LISTING NODES

To list all nodes that are known to the master:

```
$ oc get nodes
NAME                                STATUS                                AGE
master.example.com                 Ready,SchedulingDisabled            165d
node1.example.com                  Ready                                165d
node2.example.com                  Ready                                165d
```

To only list information about a single node, replace **<node>** with the full node name:

```
$ oc get node <node>
```

The **STATUS** column in the output of these commands can show nodes with the following conditions:

Table 2.1. Node Conditions

Condition	Description
Ready	The node is passing the health checks performed from the master by returning StatusOK .
NotReady	The node is not passing the health checks performed from the master.
SchedulingDisabled	Pods cannot be scheduled for placement on the node.



NOTE

The **STATUS** column can also show **Unknown** for a node if the CLI cannot find any node condition.

To get more detailed information about a specific node, including the reason for the current condition:

```
$ oc describe node <node>
```

For example:

```
oc describe node node1.example.com
Name:      node1.example.com 1
```

```

Role:
Labels:  beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        kubernetes.io/hostname=node1.example.com
        openshift-infra=apiserver
        region=infra
        zone=default
Annotations:  volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:  <none>
CreationTimestamp:  Sun, 22 Apr 2018 00:25:44 +0530
Conditions:
  Type          Status  LastHeartbeatTime   LastTransitionTime  ReasonMessage
  ----          -
  OutOfDisk     False   Wed, 30 May 2018 15:33:11 +0530  Fri, 27 Apr 2018
19:27:15 +0530  KubeletHasSufficientDisk  kubelet has sufficient disk
space available
  MemoryPressure False   Wed, 30 May 2018 15:33:11 +0530  Fri, 27 Apr 2018
19:27:15 +0530  KubeletHasSufficientMemory kubelet has sufficient memory
available
  DiskPressure  False   Wed, 30 May 2018 15:33:11 +0530  Fri, 27 Apr 2018
19:27:15 +0530  KubeletHasNoDiskPressure  kubelet has no disk pressure
  Ready         True    Wed, 30 May 2018 15:33:11 +0530  Tue, 22 May 2018 16:49:08
+0530  KubeletReady  kubelet is posting ready status
Addresses:
  InternalIP: 10.74.251.93
  Hostname: node1.example.com
Capacity:
  cpu: 2
  memory: 3880072Ki
  pods: 20
Allocatable:
  cpu: 2
  memory: 3777672Ki
  pods: 20
System Info:
  Machine ID: fb032ca6d58a4844de5dc5d
  System UUID: FB032CA6-D58A8734-F5844DE5DC5D
  Boot ID: c8d481da-1dee-50b-d7fe80ced92e
  Kernel Version: 3.10.0-514.el7.x86_64
  OS Image: Employee SKU
  Operating System: linux
  Architecture: amd64
  Container Runtime Version: docker://1.12.6
  Kubelet Version: v1.7.6+a08f5eeb62
  Kube-Proxy Version: v1.7.6+a08f5eeb62
ExternalID: node1.example.com
Non-terminated Pods: (11 in total)
  Namespace      Name           CPU Requests  CPU Limits  Memory Requests  Memory
Limits
  -----
  default        docker-registry-1-d7w49  100m (5%)  0 (0%)  256Mi (6%)  0 (0%)
  default        registry-console-1-4rvvx  0 (0%)  0 (0%)  0 (0%)  0 (0%)
  default        router-1-9mz44  100m (5%)  0 (0%)  256Mi (6%)  0 (0%)

```

```

    ggg      httpd-example-1-8zdz4    0 (0%)  0 (0%)  512Mi (13%) 512Mi (13%)
    http     httpd-example-1-2x87f    0 (0%)  0 (0%)  512Mi (13%) 512Mi (13%)
    kube-service-catalog  apiserver-ffvch    0 (0%)  0 (0%)  0 (0%)0 (0%)
    kube-service-catalog  controller-manager-9lj6q  0 (0%)  0 (0%)  0 (0%)0
(0%)
    openshift-ansible-service-broker asb-2-hb4nc    0 (0%)  0 (0%)  0 (0%)0
(0%)
    openshift-ansible-service-broker asb-etcd-2-ghfhk    0 (0%)  0 (0%)  0
(0%)0 (0%)
    openshift-template-service-broker apiserver-rrvqb    0 (0%)  0 (0%)  0
(0%)0 (0%)
    policy    color-1-5xjsw    0 (0%)  0 (0%)  0 (0%)0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests CPU Limits Memory Requests Memory Limits
  -----
  200m (10%) 0 (0%) 1536Mi (41%) 1Gi (27%)
Events:  <none>

```

- 1 The name of the node.
- 2 The role of the node, either **master** or **compute**.
- 3 The [labels](#) applied to the node.
- 4 The annotations applied to the node.
- 5 The [taints](#) applied to the node.
- 6 [Node conditions](#).
- 7 The IP address and host name of the node.
- 8 The [pod resources and allocatable resources](#).
- 9 Information about the node host.
- 10 The pods on the node.

2.3. VIEWING NODES

You can display usage statistics about nodes, which provide the runtime environments for containers. These usage statistics include CPU, memory, and storage consumption.

To view the usage statistics:

```

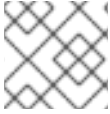
$ oc adm top nodes
NAME          CPU(cores)   CPU%    MEMORY(bytes)  MEMORY%
node-1        297m         29%     4263Mi         55%
node-0        55m          5%      1201Mi         15%
infra-1       85m          8%      1319Mi         17%
infra-0       182m         18%     2524Mi         32%
master-0      178m         8%      2584Mi         16%

```

To view the usage statistics for nodes with labels:

```
$ oc adm top node --selector=''
```

You must choose the selector (label query) to filter on. Supports `=`, `==`, and `!=`.



NOTE

You must have **cluster-reader** permission to view the usage statistics.



NOTE

Metrics must be installed to view the usage statistics.

2.4. ADDING NODES

To add nodes to your existing OpenShift Container Platform cluster, you can run an Ansible playbook that handles installing the node components, generating the required certificates, and other important steps. See the [advanced installation](#) method for instructions on running the playbook directly.

Alternatively, if you used the quick installation method, you can [re-run the installer to add nodes](#), which performs the same steps.

2.5. DELETING NODES

When you delete a node using the CLI, the node object is deleted in Kubernetes, but the pods that exist on the node itself are not deleted. Any bare pods not backed by a replication controller would be inaccessible to OpenShift Container Platform, pods backed by replication controllers would be rescheduled to other available nodes, and [local manifest pods](#) would need to be manually deleted.

To delete a node from the OpenShift Container Platform cluster:

1. [Evacuate pods](#) from the node you are preparing to delete.
2. Delete the node object:

```
$ oc delete node <node>
```

3. Check that the node has been removed from the node list:

```
$ oc get nodes
```

Pods should now be only scheduled for the remaining nodes that are in **Ready** state.

4. If you want to uninstall all OpenShift Container Platform content from the node host, including all pods and containers, continue to [Uninstalling Nodes](#) and follow the procedure using the ***uninstall.yml*** playbook. The procedure assumes general understanding of the [advanced installation method](#) using Ansible.

2.6. UPDATING LABELS ON NODES

To add or update [labels](#) on a node:

```
$ oc label node <node> <key_1>=<value_1> ... <key_n>=<value_n>
```

To see more detailed usage:

```
$ oc label -h
```

2.7. LISTING PODS ON NODES

To list all or selected pods on one or more nodes:

```
$ oc adm manage-node <node1> <node2> \
  --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

To list all or selected pods on selected nodes:

```
$ oc adm manage-node --selector=<node_selector> \
  --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

2.8. MARKING NODES AS UNSCHEDULABLE OR SCHEDULABLE

By default, healthy nodes with a **Ready** status are marked as schedulable, meaning that new pods are allowed for placement on the node. Manually marking a node as unschedulable blocks any new pods from being scheduled on the node. Existing pods on the node are not affected.

To mark a node or nodes as unschedulable:

```
$ oc adm manage-node <node1> <node2> --schedulable=false
```

For example:

```
$ oc adm manage-node node1.example.com --schedulable=false
NAME                                LABELS
STATUS
node1.example.com    kubernetes.io/hostname=node1.example.com
Ready,SchedulingDisabled
```

To mark a currently unschedulable node or nodes as schedulable:

```
$ oc adm manage-node <node1> <node2> --schedulable
```

Alternatively, instead of specifying specific node names (e.g., **<node1> <node2>**), you can use the **--selector=<node_selector>** option to mark selected nodes as schedulable or unschedulable.

2.9. EVACUATING PODS ON NODES

Evacuating pods allows you to migrate all or selected pods from a given node or nodes. Nodes must first be **marked unschedulable** to perform pod evacuation.

Only pods backed by a **replication controller** can be evacuated; the replication controllers create new pods on other nodes and remove the existing pods from the specified node(s). Bare pods, meaning those not backed by a replication controller, are unaffected by default. You can evacuate a subset of pods by specifying a pod-selector. Pod selector is based on labels, so all the pods with the specified label will be evacuated.

To list pods that will be migrated without actually performing the evacuation, use the **--dry-run** option:

```
$ oc adm manage-node <node1> <node2> \
  --evacuate --dry-run [--pod-selector=<pod_selector>]
```

To actually evacuate all or selected pods on one or more nodes:

```
$ oc adm manage-node <node1> <node2> \
  --evacuate [--pod-selector=<pod_selector>]
```

You can force deletion of bare pods by using the **--force** option:

```
$ oc adm manage-node <node1> <node2> \
  --evacuate --force [--pod-selector=<pod_selector>]
```

Alternatively, instead of specifying specific node names (e.g., **<node1> <node2>**), you can use the **--selector=<node_selector>** option to evacuate pods on selected nodes.

To list objects that will be migrated without actually performing the evacuation, use the **--dry-run** option and set it to **true**:

```
$ oc adm drain <node1> <node2> --dry-run=true
```

2.10. REBOOTING NODES

To reboot a node without causing an outage for applications running on the platform, it is important to first [evacuate the pods](#). For pods that are made highly available by the routing tier, nothing else needs to be done. For other pods needing storage, typically databases, it is critical to ensure that they can remain in operation with one pod temporarily going offline. While implementing resiliency for stateful pods is different for each application, in all cases it is important to configure the scheduler to use [node anti-affinity](#) to ensure that the pods are properly spread across available nodes.

Another challenge is how to handle nodes that are running critical infrastructure such as the router or the registry. The same node evacuation process applies, though it is important to understand certain edge cases.

2.10.1. Infrastructure nodes

Infrastructure nodes are nodes that are labeled to run pieces of the OpenShift Container Platform environment. Currently, the easiest way to manage node reboots is to ensure that there are at least three nodes available to run infrastructure. The scenario below demonstrates a common mistake that can lead to service interruptions for the applications running on OpenShift Container Platform when only two nodes are available.

- Node A is marked unschedulable and all pods are evacuated.
- The registry pod running on that node is now redeployed on node B. This means node B is now running both registry pods.
- Node B is now marked unschedulable and is evacuated.
- The service exposing the two pod endpoints on node B, for a brief period of time, loses all endpoints until they are redeployed to node A.

The same process using three infrastructure nodes does not result in a service disruption. However, due to pod scheduling, the last node that is evacuated and brought back in to rotation is left running zero registries. The other two nodes will run two and one registries respectively. The best solution is to rely on pod anti-affinity. This is an alpha feature in Kubernetes that is available for testing now, but is not yet supported for production workloads.

2.10.2. Using pod anti-affinity

[Pod anti-affinity](#) is slightly different than [node anti-affinity](#). Node anti-affinity can be violated if there are no other suitable locations to deploy a pod. Pod anti-affinity can be set to either required or preferred.

Using the **docker-registry** pod as an example, the first step in enabling this feature is to set the **scheduler.alpha.kubernetes.io/affinity** on the pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: ❶
      preferredDuringSchedulingIgnoredDuringExecution: ❷
        - weight: 100 ❸
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: docker-registry ❹
                  operator: In ❺
                  values:
                    - default
            topologyKey: kubernetes.io/hostname
```

- ❶ Stanza to configure pod anti-affinity.
- ❷ Defines a preferred rule.
- ❸ Specifies a weight for a preferred rule. The node with the highest weight is preferred.
- ❹ Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.
- ❺ The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.



IMPORTANT

scheduler.alpha.kubernetes.io/affinity is internally stored as a string even though the contents are JSON. The above example shows how this string can be added as an annotation to a YAML deployment configuration.

This example assumes the Docker registry pod has a label of **docker-registry=default**. Pod anti-affinity can use any Kubernetes match expression.

The last required step is to enable the **MatchInterPodAffinity** scheduler predicate in */etc/origin/master/scheduler.json*. With this in place, if only two infrastructure nodes are available and one is rebooted, the Docker registry pod is prevented from running on the other node. **oc get pods** reports the pod as unready until a suitable node is available. Once a node is available and all pods are back in ready state, the next node can be restarted.

2.10.3. Handling nodes running routers

In most cases, a pod running an OpenShift Container Platform router will expose a host port. The **PodFitsPorts** scheduler predicate ensures that no router pods using the same port can run on the same node, and pod anti-affinity is achieved. If the routers are relying on [IP failover](#) for high availability, there is nothing else that is needed. For router pods relying on an external service such as AWS Elastic Load Balancing for high availability, it is that service's responsibility to react to router pod restarts.

In rare cases, a router pod may not have a host port configured. In those cases, it is important to follow the [recommended restart process](#) for infrastructure nodes.

2.11. CONFIGURING NODE RESOURCES

You can configure node resources by adding kubelet arguments to the node configuration file (*/etc/origin/node/node-config.yaml*). Add the **kubeletArguments** section and include any desired options:

```
kubeletArguments:
  max-pods: ❶
    - "40"
  resolv-conf: ❷
    - "/etc/resolv.conf"
  image-gc-high-threshold: ❸
    - "90"
  image-gc-low-threshold: ❹
    - "80"
```

- ❶ [Maximum number of pods that can run on this kubelet.](#)
- ❷ [Resolver configuration file used as the basis for the container DNS resolution configuration.](#)
- ❸ [The percent of disk usage after which image garbage collection is always run. Default: 90%](#)
- ❹ [The percent of disk usage before which image garbage collection is never run. Lowest disk usage to garbage collect to. Default: 80%](#)

To view all available kubelet options:

```
$ kubelet -h
```

This can also be set during an [advanced installation](#) using the **openshift_node_kubelet_args** variable. For example:

```
openshift_node_kubelet_args={'max-pods': ['40'], 'resolv-conf':
['/etc/resolv.conf'], 'image-gc-high-threshold': ['90'], 'image-gc-low-
threshold': ['80']}
```

2.11.1. Setting maximum pods per node



NOTE

See the [Cluster Limits](#) page for the maximum supported limits for each version of OpenShift Container Platform.

In the `/etc/origin/node/node-config.yaml` file, two parameters control the maximum number of pods that can be scheduled to a node: **pods-per-core** and **max-pods**. When both options are in use, the lower of the two limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization on both OpenShift Container Platform and Docker.
- Slow pod scheduling.
- Potential out-of-memory scenarios (depends on the amount of memory in the node).
- Exhausting the pool of IP addresses.
- Resource overcommitting, leading to poor user application performance.

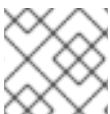


NOTE

In Kubernetes, a pod that is holding a single container actually uses two containers. The second container is used to set up networking prior to the actual container starting. Therefore, a system running 10 pods will actually have 20 containers running.

pods-per-core sets the number of pods the node can run based on the number of processor cores on the node. For example, if **pods-per-core** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be 40.

```
kubeletArguments:
  pods-per-core:
    - "10"
```



NOTE

Setting **pods-per-core** to 0 disables this limit.

max-pods sets the number of pods the node can run to a fixed value, regardless of the properties of the node. [Cluster Limits](#) documents maximum supported values for **max-pods**.

```
kubeletArguments:
  max-pods:
    - "250"
```

Using the above example, the default value for **pods-per-core** is **10** and the default value for **max-pods** is **250**. This means that unless the node has 25 cores or more, by default, **pods-per-core** will be the limiting factor.

2.12. RESETTNG DOCKER STORAGE

As you download Docker images and run and delete containers, Docker does not always free up mapped disk space. As a result, over time you can run out of space on a node, which might prevent OpenShift Container Platform from being able to create new pods or cause pod creation to take several minutes.

For example, the following shows pods that are still in the **ContainerCreating** state after six minutes and the events log shows a **FailedSync event**.

```
$ oc get pod
NAME                                READY   STATUS
RESTARTS   AGE
cakephp-mysql-persistent-1-build    0/1     ContainerCreating   0
6m
mysql-1-9767d                       0/1     ContainerCreating   0
2m
mysql-1-deploy                      0/1     ContainerCreating   0
6m

$ oc get events
LASTSEEN   FIRSTSEEN   COUNT   NAME                                KIND
SUBJECT                                         REASON
SOURCE                                           MESSAGE
6m         6m          1       cakephp-mysql-persistent-1-build    Pod
Normal     Scheduled   default-scheduler
Successfully assigned cakephp-mysql-persistent-1-build to ip-172-31-71-
195.us-east-2.compute.internal
2m         5m          4       cakephp-mysql-persistent-1-build    Pod
Warning    FailedSync  kubelet, ip-172-31-71-195.us-
east-2.compute.internal   Error syncing pod
2m         4m          4       cakephp-mysql-persistent-1-build    Pod
Normal     SandboxChanged  kubelet, ip-172-31-71-195.us-
east-2.compute.internal   Pod sandbox changed, it will be killed and re-
created.
```

One solution to this problem is to reset Docker storage to remove artifacts not needed by Docker.

On the node where you want to restart Docker storage:

1. Run the following command to mark the node as unschedulable:

```
$ oc adm manage-node <node> --schedulable=false
```

2. Run the following command to shut down Docker and the **atomic-openshift-node** service:

```
$ systemctl stop docker atomic-openshift-node
```

3. Run the following command to remove the local volume directory:

```
$ rm -rf /var/lib/origin/openshift.local.volumes
```

This command clears the local image cache. As a result, images, including **ose-*** images, will need to be re-pulled. This might result in slower pod start times while the image store recovers.

4. Remove the **/var/lib/docker** directory:

```
$ rm -rf /var/lib/docker
```

5. Run the following command to reset the Docker storage:

```
$ docker-storage-setup --reset
```

6. Run the following command to recreate the Docker storage:

```
$ docker-storage-setup
```

7. Recreate the **/var/lib/docker** directory:

```
$ mkdir /var/lib/docker
```

8. Run the following command to restart Docker and the **atomic-openshift-node** service:

```
$ systemctl start docker atomic-openshift-node
```

9. Run the following command to mark the node as schedulable:

```
$ oc adm manage-node <node> --schedulable=true
```

2.13. CHANGING NODE TRAFFIC INTERFACE

By default, DNS routes all node traffic. During node registration, the master receives the node IP addresses from the DNS configuration, and therefore accessing nodes via DNS is the most flexible solution for most deployments.

If your deployment is using a cloud provider, then the node gets the IP information from the cloud provider. However, **openshift-sdn** attempts to determine the IP through a variety of methods, including a DNS lookup on the nodeName (if set), or on the system hostname (if nodeName is not set).

However, you may need to change the node traffic interface. For example, where:

- OpenShift Container Platform is installed in a cloud provider where internal hostnames are not configured/resolvable by all hosts.
- The node's IP from the master's perspective is not the same as the node's IP from its own perspective.

Configuring the **openshift_set_node_ip** Ansible variable forces node traffic through an interface other than the default network interface.

To change the node traffic interface:

1. Set the **openshift_set_node_ip** Ansible variable to **true**.
2. Set the **openshift_ip** to the IP address for the node you want to configure.

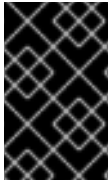
Although **openshift_set_node_ip** can be useful as a workaround for the cases stated in this section, it is generally not suited for production environments. This is because the node will no longer function properly if it receives a new IP address.

CHAPTER 3. RESTORING OPENSIFT CONTAINER PLATFORM COMPONENTS

3.1. OVERVIEW

In OpenShift Container Platform, you can *restore* your cluster and its components by recreating cluster elements, including nodes and applications, from separate storage.

To restore a cluster, you must first [back it up](#).



IMPORTANT

The following process describes a generic way of restoring applications and the OpenShift Container Platform cluster. It cannot take into account custom requirements. You might need to take additional actions to restore your cluster.

3.2. RESTORING A CLUSTER

To restore a cluster, first reinstall OpenShift Container Platform.

Procedure

1. Reinstall OpenShift Container Platform in the [same way](#) that you originally installed OpenShift Container Platform.
2. Run all of your custom post-installation steps, such as changing services outside of the control of OpenShift Container Platform or installing extra services like monitoring agents.

3.3. RESTORING A MASTER HOST BACKUP

After creating a backup of important master host files, if they become corrupted or accidentally removed, you can restore the files by copying the files back to master, ensuring they contain the proper content, and restarting the affected services.

Procedure

1. Restore the `/etc/origin/master/master-config.yaml` file:

```
# MYBACKUPDIR=*/backup/${hostname}/${date +%Y%m%d}*
# cp /etc/origin/master/master-config.yaml
/etc/origin/master/master-config.yaml.old
# cp /backup/${hostname}/${date +%Y%m%d}/origin/master/master-
config.yaml /etc/origin/master/master-config.yaml
# systemctl restart atomic-openshift-master-api
# systemctl restart atomic-openshift-master-controllers
```

**WARNING**

Restarting the master services can lead to downtime. However, you can remove the master host from the highly available load balancer pool, then perform the restore operation. Once the service has been properly restored, you can add the master host back to the load balancer pool.

**NOTE**

Perform a full reboot of the affected instance to restore the **iptables** configuration.

2. If you cannot restart OpenShift Container Platform because packages are missing, reinstall the packages.

- a. Get the list of the current installed packages:

```
$ rpm -qa | sort > /tmp/current_packages.txt
```

- b. View the differences between the package lists:

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt
> ansible-2.4.0.0-5.el7.noarch
```

- c. Reinstall the missing packages:

```
# yum reinstall -y <packages> 1
```

- 1** Replace **<packages>** with the packages that are different between the package lists.

3. Restore a system certificate by copying the certificate to the **/etc/pki/ca-trust/source/anchors/** directory and execute the **update-ca-trust**:

```
$ MYBACKUPDIR=*/backup/${hostname}/${date +%Y%m%d}*
$ sudo cp ${MYBACKUPDIR}/external_certificates/my_company.crt
/etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust
```

**NOTE**

Always ensure the user ID and group ID are restored when the files are copied back, as well as the **SELinux** context.

3.4. RESTORING A NODE HOST BACKUP

After creating a backup of important node host files, if they become corrupted or accidentally removed, you can restore the file by copying back the file, ensuring it contains the proper content and restart the affected services.

Procedure

1. Restore the `/etc/origin/node/node-config.yaml` file:

```
# MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
# cp /etc/origin/node/node-config.yaml /etc/origin/node/node-
  config.yaml.old
# cp /backup/$(hostname)/$(date +%Y%m%d)/etc/origin/node/node-
  config.yaml /etc/origin/node/node-config.yaml
# systemctl restart atomic-openshift-node
```



WARNING

Restarting the services can lead to downtime. See [Node maintenance](#), for tips on how to ease the process.



NOTE

Perform a full reboot of the affected instance to restore the **iptables** configuration.

1. If you cannot restart OpenShift Container Platform because packages are missing, reinstall the packages.

- a. Get the list of the current installed packages:

```
$ rpm -qa | sort > /tmp/current_packages.txt
```

- b. View the differences between the package lists:

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt
> ansible-2.4.0.0-5.el7.noarch
```

- c. Reinstall the missing packages:

```
# yum reinstall -y <packages> 1
```

- 1** Replace **<packages>** with the packages that are different between the package lists.

2. Restore a system certificate by copying the certificate to the `/etc/pki/ca-trust/source/anchors/` directory and execute the **update-ca-trust**:

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/etc/pki/ca-
```

```
trust/source/anchors/my_company.crt /etc/pki/ca-
trust/source/anchors/
$ sudo update-ca-trust
```

**NOTE**

Always ensure proper user ID and group ID are restored when the files are copied back, as well as the **SELinux** context.

3.5. RESTORING ETCD

The restore procedure for etcd configuration files replaces the appropriate files, then restarts the service.

If an etcd host has become corrupted and the **/etc/etcd/etcd.conf** file is lost, restore it using:

```
$ ssh master-0
# cp /backup/yesterday/master-0-files/etcd.conf /etc/etcd/etcd.conf
# restorecon -Rv /etc/etcd/etcd.conf
# systemctl restart etcd.service
```

In this example, the backup file is stored in the **/backup/yesterday/master-0-files/etcd.conf** path where it can be used as an external NFS share, S3 bucket, or other storage solution.

3.5.1. Restoring etcd v2 & v3 data

The following process restores healthy data files and starts the etcd cluster as a single node, then adds the rest of the nodes if an etcd cluster is required.

Procedure

1. Stop all etcd services:

```
# systemctl stop etcd.service
```

2. To ensure the proper backup is restored, delete the etcd directories:

- To back up the current etcd data before you delete the directory, run the following command:

```
# mv /var/lib/etcd /var/lib/etcd.old
# mkdir /var/lib/etcd
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd/
```

- Or, to delete the directory and the etcd, data, run the following command:

```
# rm -Rf /var/lib/etcd/*
```

**NOTE**

In an all-in-one cluster, the etcd data directory is located in the **/var/lib/origin/openshift.local.etcd** directory.

3. Restore a healthy backup data file to each of the etcd nodes. Perform this step on all etcd hosts, including master hosts collocated with etcd.

```
# cp -R /backup/etcd-xxx/* /var/lib/etcd/
# mv /var/lib/etcd/db /var/lib/etcd/member/snap/db
# chcon -R --reference /backup/etcd-xxx/* /var/lib/etcd/
# chown -R etcd:etcd /var/lib/etcd/R
```

4. Run the etcd service on each host, forcing a new cluster.

This creates a custom file for the etcd service, which overwrites the execution command adding the **--force-new-cluster** option:

```
# mkdir -p /etc/systemd/system/etcd.service.d/
# echo "[Service]" > /etc/systemd/system/etcd.service.d/temp.conf
# echo "ExecStart=" >> /etc/systemd/system/etcd.service.d/temp.conf
# sed -n '/ExecStart/s/"$/ --force-new-cluster"/p' \
    /usr/lib/systemd/system/etcd.service \
    >> /etc/systemd/system/etcd.service.d/temp.conf

# systemctl daemon-reload
# systemctl restart etcd
```

5. Check for error messages:

```
$ journalctl -fu etcd.service
```

6. Check for health status:

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy
```

7. Restart the etcd service in cluster mode:

```
# rm -f /etc/systemd/system/etcd.service.d/temp.conf
# systemctl daemon-reload
# systemctl restart etcd
```

8. Check for health status and member list:

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy

# etcdctl2 member list
5ee217d17301: name=master-0.example.com
peerURLs=http://localhost:2380 clientURLs=https://192.168.55.8:2379
isLeader=true
```

9. After the first instance is running, you can restore the rest of your etcd servers.

3.5.1.1. Fix the peerURLs parameter

After restoring the data and creating a new cluster, the **peerURLs** parameter shows **localhost** instead of the IP where etcd is listening for peer communication:

```
# etcdctl member list
5ee217d17301: name=master-0.example.com peerURLs=http://*localhost*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

3.5.1.1.1. Procedure

1. Get the member ID using **etcdctl member list**:

```
`etcdctl member list`
```

2. Get the IP where etcd listens for peer communication:

```
$ ss -ltn | grep 2380
```

3. Update the member information with that IP:

```
# etcdctl member update 5ee217d17301 https://192.168.55.8:2380
Updated member with ID 5ee217d17301 in cluster
```

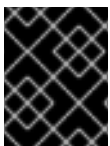
4. To verify, check that the IP is in the member list:

```
$ etcdctl member list
5ee217d17301: name=master-0.example.com
peerURLs=https://*192.168.55.8*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

3.5.2. Restoring etcd for v3

The restore procedure for v3 data is similar to the restore procedure for the v2 data.

Snapshot integrity may be optionally verified at restore time. If the snapshot is taken with **etcdctl snapshot save**, it will have an integrity hash that is checked by **etcdctl snapshot restore**. If the snapshot is copied from the data directory, there is no integrity hash and it will only restore by using **--skip-hash-check**.



IMPORTANT

The procedure to restore only the v3 data must be performed on a single etcd host. You can then add the rest of the nodes to the cluster.

Procedure

1. Stop all etcd services:

```
# systemctl stop etcd.service
```

2. Clear all old data, because **etcdctl** recreates it in the node where the restore procedure is going to be performed:

```
# rm -Rf /var/lib/etcd
```

3. Run the **snapshot restore** command, substituting the values from the **/etc/etcd/etcd.conf** file:

```
# etcdctl3 snapshot restore /backup/etcd-xxxxxx/backup.db \
--data-dir /var/lib/etcd \
--name master-0.example.com \
--initial-cluster "master-0.example.com=https://192.168.55.8:2380" \
--initial-cluster-token "etcd-cluster-1" \
--initial-advertise-peer-urls https://192.168.55.8:2380

2017-10-03 08:55:32.440779 I | mvcc: restore compact to 1041269
2017-10-03 08:55:32.468244 I | etcdserver/membership: added member
40bef1f6c79b3163 [https://192.168.55.8:2380] to cluster
26841ebcf610583c
```

4. Restore permissions and **selinux** context to the restored files:

```
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd
```

5. Start the etcd service:

```
# systemctl start etcd
```

6. Check for any error messages:

```
$ journalctl -fu etcd.service
```

3.6. ADDING AN ETCD NODE

After you restore etcd, you can add more etcd nodes to the cluster. You can either add an etcd host by using an Ansible playbook or by manual steps.

3.6.1. Adding a new etcd host using Ansible

Procedure

1. In the Ansible inventory file, create a new group named **[new_etcd]** and add the new host. Then, add the **new_etcd** group as a child of the **[OSEv3]** group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd 1

... [OUTPUT ABBREVIATED] ...
```

```
[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
```

```
[new_etcd] 2
etcd0.example.com 3
```

1 2 3 Add these lines.

2. From the host that installed OpenShift Container Platform and hosts the Ansible inventory file, run the `etcd scaleup` playbook:

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-etcd/scaleup.yml
```

3. After the playbook runs, modify the inventory file to reflect the current status by moving the new `etcd` host from the `[new_etcd]` group to the `[etcd]` group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. If you use Flannel, modify the `flanneld` service configuration on every OpenShift Container Platform host, located at `/etc/sysconfig/flanneld`, to include the new `etcd` host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379,https://etcd0.example.com:2379
```

5. Restart the `flanneld` service:

```
# systemctl restart flanneld.service
```

3.6.2. Manually adding a new etcd host

Procedure

Modify the current etcd cluster

To create the `etcd` certificates, run the `openssl` command, replacing the values with those from your environment.

1. Create some environment variables:

■

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}"
export PREFIX="/etc/etcd/generated_certs/etcd-${CN}/"
export OPENSSECFG="/etc/etcd/ca/openssl.cnf"
```



NOTE

The custom **openssl** extensions used as **etcd_v3_ca_*** include the **\$SAN** environment variable as **subjectAltName**. See **/etc/etcd/ca/openssl.cnf** for more information.

2. Create the directory to store the configuration and certificates:

```
# mkdir -p ${PREFIX}
```

3. Create the server certificate request and sign it: (**server.csr** and **server.crt**)

```
# openssl req -new -config ${OPENSSECFG} \
  -keyout ${PREFIX}server.key \
  -out ${PREFIX}server.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSECFG} \
  -out ${PREFIX}server.crt \
  -in ${PREFIX}server.csr \
  -extensions etcd_v3_ca_server -batch
```

4. Create the peer certificate request and sign it: (**peer.csr** and **peer.crt**)

```
# openssl req -new -config ${OPENSSECFG} \
  -keyout ${PREFIX}peer.key \
  -out ${PREFIX}peer.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSECFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. Copy the current etcd configuration and **ca.crt** files from the current node as examples to modify later:

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. While still on the surviving etcd host, add the new host to the cluster. To add additional etcd members to the cluster, you must first adjust the default **localhost** peer in the **peerURLs** value for the first member:

- a. Get the member ID for the first member using the **member list** command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
172.18.0.75:2379" \ ❶
member list
```

- ❶ Ensure that you specify the URLs of only active etcd members in the **--peers** parameter value.

- b. Obtain the IP address where etcd listens for cluster peers:

```
$ ss -ltn | grep 2380
```

- c. Update the value of **peerURLs** using the **etcdctl member update** command by passing the member ID and IP address obtained from the previous steps:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
172.18.0.75:2379" \
member update 511b7fb6cc0001 https://172.18.1.18:2380
```

- d. Re-run the **member list** command and ensure the peer URLs no longer include **localhost**.

7. Add the new host to the etcd cluster. Note that the new host is not yet configured, so the status stays as **unstarted** until the you configure the new host.



WARNING

You must add each member and bring it online one at a time. When you add each additional member to the cluster, you must adjust the **peerURLs** list for the current peers. The **peerURLs** list grows by one for each member added. The **etcdctl member add** command outputs the values that you must set in the **etcd.conf** file as you add each member, as described in the following instructions.

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 ❶
```

```
Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster
```

```
ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https://<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

- 1 In this line, **10.3.9.222** is a label for the etcd member. You can specify the host name, IP address, or a simple name.

8. Update the sample **\${PREFIX}/etcd.conf** file.

- a. Replace the following values with the values generated in the previous step:

- ETCD_NAME
- ETCD_INITIAL_CLUSTER
- ETCD_INITIAL_CLUSTER_STATE

- b. Modify the following variables with the new host IP from the output of the previous step. You can use **\${NEW_ETCD_IP}** as the value.

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

- c. If you previously used the member system as an etcd node, you must overwrite the current values in the **/etc/etcd/etcd.conf** file.
- d. Check the file for syntax errors or missing IP addresses, otherwise the etcd service might fail:

```
# vi ${PREFIX}/etcd.conf
```

9. On the node that hosts the installation files, update the **[etcd]** hosts group in the **/etc/ansible/hosts** inventory file. Remove the old etcd hosts and add the new ones.
10. Create a **tgz** file that contains the certificates, the sample configuration file, and the **ca** and copy it to the new host:

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

Modify the new etcd host

1. Install **iptables-services** to provide iptables utilities to open the required ports for etcd:

```
# yum install -y iptables-services
```

2. Create the **OS_FIREWALL_ALLOW** firewall rules to allow etcd to communicate:

- Port 2379/tcp for clients
- Port 2380/tcp for peer communication

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2380 -j ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```

**NOTE**

In this example, a new chain **OS_FIREWALL_ALLOW** is created, which is the standard naming the OpenShift Container Platform installer uses for firewall rules.

**WARNING**

If the environment is hosted in an IaaS environment, modify the security groups for the instance to allow incoming traffic to those ports as well.

3. Install etcd:

```
# yum install -y etcd
```

Ensure version **etcd-2.3.7-4.el7.x86_64** or greater is installed,

4. Ensure the etcd service is not running:

```
# systemctl disable etcd --now
```

5. Remove any etcd configuration and data:

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. Extract the certificates and configuration files:

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. Modify the file ownership permissions:

```
# chown -R etcd:etcd /etc/etcd/*
# chown -R etcd:etcd /var/lib/etcd/
```


- 8. Start etcd on the new host:

```
# systemctl enable etcd --now
```

- 9. Verify that the host is part of the cluster and the current cluster health:

- If you use the v2 etcd api, run the following command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --peers="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

- If you use the v3 etcd api, run the following command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
  --key=/etc/etcd/peer.key \
  --cacert="/etc/etcd/ca.crt" \
  --endpoints="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed
proposal: took = 1.498829ms
```

Modify each OpenShift Container Platform master

1. Modify the master configuration in the **etcdClientInfo** section of the **/etc/origin/master/master-config.yaml** file on every master. Add the new etcd host to the list of the etcd servers OpenShift Container Platform uses to store the data, and remove any failed etcd hosts:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
```

```
certFile: master.etcd-client.crt
keyFile: master.etcd-client.key
urls:
  - https://master-0.example.com:2379
  - https://master-1.example.com:2379
  - https://master-2.example.com:2379
  - https://etcd0.example.com:2379
```

2. Restart the master API service:

- On every master:

```
# systemctl restart atomic-openshift-master-api
```

- Or, on a single master cluster installation:

```
# systemctl restart atomic-openshift-master
```



WARNING

The number of etcd nodes must be odd, so you must add at least two hosts.

3. If you use Flannel, modify the **flanneld** service configuration located at **/etc/sysconfig/flanneld** on every OpenShift Container Platform host to include the new etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

3.7. BRINGING OPENSIFT CONTAINER PLATFORM SERVICES BACK ONLINE

After you finish your changes, bring OpenShift Container Platform back online.

Procedure

1. On each OpenShift Container Platform master, restore your master and node configuration from backup and enable and restart all relevant services:

```
# cp ${MYBACKUPDIR}/etc/sysconfig/atomic-openshift-master-api
/etc/sysconfig/atomic-openshift-master-api
# cp ${MYBACKUPDIR}/etc/sysconfig/atomic-openshift-master-
```

```

controllers /etc/sysconfig/atomic-openshift-master-controllers
# cp ${MYBACKUPDIR}/etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp ${MYBACKUPDIR}/etc/origin/node/node-config.yaml.<timestamp>
/etc/origin/node/node-config.yaml
# systemctl enable atomic-openshift-master-api
# systemctl enable atomic-openshift-master-controllers
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master-api
# systemctl start atomic-openshift-master-controllers
# systemctl start atomic-openshift-node

```

2. On each OpenShift Container Platform node, restore your **node-config.yaml** file from backup and enable and restart the **atomic-openshift-node** service:

```

# cp /etc/origin/node/node-config.yaml.<timestamp> /etc/origin/node/node-
config.yaml
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-node

```

3.8. RESTORING A PROJECT

To restore a project, create the new project, then restore any exported files by running **oc create -f pods.json**. However, restoring a project from scratch requires a specific order because some objects depend on others. For example, you must create the **configmaps** before you create any **pods**.

Procedure

1. If the project was exported as a single file, import it by running the following commands:

```

$ oc new-project <projectname>
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml

```



WARNING

Some resources, such as pods and default service accounts, can fail to be created.

3.9. RESTORING APPLICATION DATA

You can restore application data by using the **oc rsync** command, assuming **rsync** is installed within the container image. The Red Hat **rhel7** base image contains **rsync**. Therefore, all images that are based on **rhel7** contain it as well. See [Troubleshooting and Debugging CLI Operations - rsync](#).

**WARNING**

This is a *generic* restoration of application data and does not take into account application-specific backup procedures, for example, special export and import procedures for database systems.

Other means of restoration might exist depending on the type of the persistent volume you use, for example, Cinder, NFS, or Gluster.

Procedure**Example of restoring a Jenkins deployment's application data**

1. Verify the backup:

```
$ ls -la /tmp/jenkins-backup/
total 8
drwxrwxr-x.  3 user      user   20 Sep  6 11:14 .
drwxrwxrwt. 17 root      root  4096 Sep  6 11:16 ..
drwxrwsrwx. 12 user      user  4096 Sep  6 11:14 jenkins
```

2. Use the **oc rsync** tool to copy the data into the running pod:

```
$ oc rsync /tmp/jenkins-backup/jenkins jenkins-1-37nux:/var/lib
```

**NOTE**

Depending on the application, you may be required to restart the application.

3. Optionally, restart the application with new data:

```
$ oc delete pod jenkins-1-37nux
```

Alternatively, you can scale down the deployment to 0, and then up again:

```
$ oc scale --replicas=0 dc/jenkins
$ oc scale --replicas=1 dc/jenkins
```

3.10. RESTORING PERSISTENT VOLUME CLAIMS

This topic describes two methods for restoring data. The first involves deleting the file, then placing the file back in the expected location. The second example shows migrating persistent volume claims. The migration would occur in the event that the storage needs to be moved or in a disaster scenario when the backend storage no longer exists.

Check with the restore procedures for the specific application on any steps required to restore data to the application.

3.10.1. Restoring files to an existing PVC

Procedure

1. Delete the file:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ ls */opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
sh-4.2$ *rm -rf /opt/app-root/src/uploaded/ocp_sop.txt*
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found
```

2. Replace the file from the server that contains the rsync backup of the files that were in the pvc:

```
$ oc rsync uploaded demo-2-fxx6d:/opt/app-root/src/
```

3. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
```

3.10.2. Restoring data to a new PVC

The following steps assume that a new **pvc** has been created.

Procedure

1. Overwrite the currently defined **claim-name**:

```
$ oc volume dc/demo --add --name=persistent-volume \
  --type=persistentVolumeClaim --claim-name=filestore \ --mount-
  path=/opt/app-root/src/uploaded --overwrite
```

2. Validate that the pod is using the new PVC:

```
$ oc describe dc/demo
Name: demo
Namespace: test
Created: 3 hours ago
Labels: app=demo
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 3
Selector: app=demo,deploymentconfig=demo
Replicas: 1
Triggers: Config, Image(demo@latest, auto=true)
Strategy: Rolling
Template:
  Labels: app=demo
  deploymentconfig=demo
  Annotations: openshift.io/container.demo.image.entrypoint=
["container-entrypoint", "/bin/sh", "-c", "$STI_SCRIPTS_PATH/usage"]
```

```
openshift.io/generated-by=OpenShiftNewApp
Containers:
  demo:
    Image: docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port: 8080/TCP
    Volume Mounts:
      /opt/app-root/src/uploaded from persistent-volume (rw)
    Environment Variables: <none>
  Volumes:
    persistent-volume:
      Type: PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
      *ClaimName: filestore*
      ReadOnly: false
    ...omitted...
```

3. Now that the deployment configuration uses the new **pvc**, run **oc rsync** to place the files onto the new **pvc**:

```
$ oc rsync uploaded demo-3-2b8gs:/opt/app-root/src/
sending incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 181 bytes  received 39 bytes  146.67 bytes/sec
total size is 32  speedup is 0.15
```

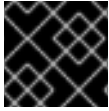
4. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory:

```
$ oc rsh demo-3-2b8gs
sh-4.2$ ls /opt/app-root/src/uploaded/
lost+found  ocp_sop.txt
```

CHAPTER 4. REPLACING A MASTER HOST

You can replace a failed master host.

First, remove the failed master host from your cluster, and then add a replacement master host. If the failed master host ran etcd, scale up etcd by adding etcd to the new master host.



IMPORTANT

You must complete all sections of this topic.

4.1. DEPRECATING A MASTER HOST

Master hosts run important services, such as the OpenShift Container Platform API and controllers services. In order to deprecate a master host, these services must be stopped.

The OpenShift Container Platform API service is an active/active service, so stopping the service does not affect the environment as long as the requests are sent to a separate master server. However, the OpenShift Container Platform controllers service is an active/passive service, where the services leverage etcd to decide the active master.

Deprecating a master host in a multi-master architecture includes removing the master from the load balancer pool to avoid new connections attempting to use that master. This process depends heavily on the load balancer used. The steps below show the details of removing the master from **haproxy**. In the event that OpenShift Container Platform is running on a cloud provider, or using a **F5** appliance, see the specific product documents to remove the master from rotation.

Procedure

1. Remove the **backend** section in the `/etc/haproxy/haproxy.cfg` configuration file. For example, if deprecating a master named **master-0.example.com** using **haproxy**, ensure the host name is removed from the following:

```
backend mgmt8443
    balance source
    mode tcp
    # MASTERS 8443
    server master-1.example.com 192.168.55.12:8443 check
    server master-2.example.com 192.168.55.13:8443 check
```

2. Then, restart the **haproxy** service.

```
$ sudo systemctl restart haproxy
```

3. Once the master is removed from the load balancer, disable the API and controller services:

```
$ sudo systemctl disable --now atomic-openshift-master-api
$ sudo systemctl disable --now atomic-openshift-master-controllers
```

4. Because the master host is a unschedulable OpenShift Container Platform node, follow the steps in the [Deprecating a node host](#) section.

5. Remove the master host from the **[masters]** and **[nodes]** groups in the **/etc/ansible/hosts** Ansible inventory file to avoid issues if running any Ansible tasks using that inventory file.



WARNING

Deprecating the first master host listed in the Ansible inventory file requires extra precautions.

The **/etc/origin/master/ca.serial.txt** file is generated on only the first master listed in the Ansible host inventory. If you deprecate the first master host, copy the **/etc/origin/master/ca.serial.txt** file to the rest of master hosts before the process.

6. The **kubernetes** service includes the master host IPs as endpoints. To verify that the master has been properly deprecated, review the **kubernetes** service output and see if the deprecated master has been removed:

```
$ oc describe svc kubernetes -n default
Name:      kubernetes
Namespace: default
Labels:    component=apiserver
           provider=kubernetes
Annotations: <none>
Selector:  <none>
Type:      ClusterIP
IP:        10.111.0.1
Port:      https 443/TCP
Endpoints: 192.168.55.12:8443,192.168.55.13:8443
Port:      dns 53/UDP
Endpoints: 192.168.55.12:8053,192.168.55.13:8053
Port:      dns-tcp 53/TCP
Endpoints: 192.168.55.12:8053,192.168.55.13:8053
Session Affinity: ClientIP
Events:    <none>
```

After the master has been successfully deprecated, the host where the master was previously running can be safely deleted.

4.2. 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_labels="{ 'region': 'primary',
'zone': 'east' }"
node2.example.com openshift_node_labels="{ 'region': 'primary',
'zone': 'west' }"
infra-node1.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default' }"
infra-node2.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default' }"

[new_nodes]
node3.example.com openshift_node_labels="{ 'region': 'primary',
'zone': 'west' }"
```

See [Configuring Host Variables](#) for more options.

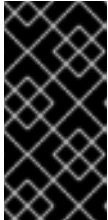
When adding new masters, add hosts to both the ***[new_masters]*** section and the ***[new_nodes]*** section to ensure that the new master host is part of the OpenShift SDN.

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_labels="{ 'region': 'primary',
'zone': 'east' }"
node2.example.com openshift_node_labels="{ 'region': 'primary',
'zone': 'west' }"
infra-node1.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default' }"
infra-node2.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default' }"

[new_nodes]
master3.example.com
```



IMPORTANT

If you label a master host with the **region=infra** 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/byo/openshift-
    node/scaleup.yml
```

- For additional masters:

```
# ansible-playbook [-i /path/to/file] \
    /usr/share/ansible/openshift-ansible/playbooks/byo/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_labels="{ 'region': 'primary',
'zone': 'east' }"
node2.example.com openshift_node_labels="{ 'region': 'primary',
```

```
'zone': 'west'}"
node3.example.com openshift_node_labels="{ 'region': 'primary',
'zone': 'west'}"
infra-node1.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default'}"
infra-node2.example.com openshift_node_labels="{ 'region': 'infra',
'zone': 'default'}"

[new_nodes]
```

4.3. SCALING ETCD

You can scale the etcd cluster vertically by adding more resources to the etcd hosts or horizontally by adding more etcd hosts.



NOTE

Due to the voting system etcd uses, the cluster must always contain an odd number of members.

Having a cluster with an odd number of etcd hosts can account for fault tolerance. Having an odd number of etcd hosts does not change the number needed for a quorum but increases the tolerance for failure. For example, with a cluster of three members, quorum is two, which leaves a failure tolerance of one. This ensures the cluster continues to operate if two of the members are healthy.

Having an in-production cluster of three etcd hosts is recommended.

The new host requires a fresh Red Hat Enterprise Linux version 7 dedicated host. The etcd storage should be located on an SSD disk to achieve maximum performance and on a dedicated disk mounted in **/var/lib/etcd**.

Prerequisites

1. Before you add a new etcd host, [perform a backup of both etcd configuration and data](#) to prevent data loss.
2. Check the current etcd cluster status to avoid adding new hosts to an unhealthy cluster.
 - If you use the v2 etcd api, run this command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key \
--ca-file=/etc/etcd/ca.crt \
--peers="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379" \
cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

- If you use the v3 etcd api, run this command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
  --key="/etc/etcd/peer.key" \
  --cacert="/etc/etcd/ca.crt" \
  --endpoints="https://*master-0.example.com*:2379,\
    https://*master-1.example.com*:2379,\
    https://*master-2.example.com*:2379"
  endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
```

3. Before running the **scaleup** playbook, ensure the new host is registered to the proper Red Hat software channels:

```
# subscription-manager register \
  --username=*<username>* --password=*<password>*
# subscription-manager attach --pool=*<poolid>*
# subscription-manager repos --disable=""
# subscription-manager repos \
  --enable=rhel-7-server-rpms \
  --enable=rhel-7-server-extras-rpms
```

etcd is hosted in the **rhel-7-server-extras-rpms** software channel.

4. Upgrade etcd and iptables on the current etcd nodes:

```
# yum update etcd iptables-services
```

5. Back up the **/etc/etcd** configuration for the etcd hosts.
6. If the new etcd members will also be OpenShift Container Platform nodes, [add the desired number of hosts to the cluster](#).
7. The rest of this procedure assumes you added one host, but if you add multiple hosts, perform all steps on each host.

4.3.1. Adding a new etcd host using Ansible

Procedure

1. In the Ansible inventory file, create a new group named **[new_etcd]** and add the new host. Then, add the **new_etcd** group as a child of the **[OSEv3]** group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd 1
... [OUTPUT ABBREVIATED] ...
```

```
[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
```

```
[new_etcd] 2
etcd0.example.com 3
```

1 2 3 Add these lines.

2. From the host that installed OpenShift Container Platform and hosts the Ansible inventory file, run the etcd **scaleup** playbook:

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-etcd/scaleup.yml
```

3. After the playbook runs, modify the inventory file to reflect the current status by moving the new etcd host from the **[new_etcd]** group to the **[etcd]** group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. If you use Flannel, modify the **flanneld** service configuration on every OpenShift Container Platform host, located at **/etc/sysconfig/flanneld**, to include the new etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379,https://etcd0.example.com:2379
```

5. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

4.3.2. Manually adding a new etcd host

Procedure

Modify the current etcd cluster

To create the etcd certificates, run the **openssl** command, replacing the values with those from your environment.

1. Create some environment variables:

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}"
export PREFIX="/etc/etcd/generated_certs/etcd-${CN}/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```



NOTE

The custom **openssl** extensions used as **etcd_v3_ca_*** include the **\$SAN** environment variable as **subjectAltName**. See **/etc/etcd/ca/openssl.cnf** for more information.

2. Create the directory to store the configuration and certificates:

```
# mkdir -p ${PREFIX}
```

3. Create the server certificate request and sign it: (**server.csr** and **server.crt**)

```
# openssl req -new -config ${OPENSSLCFG} \
  -keyout ${PREFIX}server.key \
  -out ${PREFIX}server.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}server.crt \
  -in ${PREFIX}server.csr \
  -extensions etcd_v3_ca_server -batch
```

4. Create the peer certificate request and sign it: (**peer.csr** and **peer.crt**)

```
# openssl req -new -config ${OPENSSLCFG} \
  -keyout ${PREFIX}peer.key \
  -out ${PREFIX}peer.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. Copy the current etcd configuration and **ca.crt** files from the current node as examples to modify later:

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. While still on the surviving etcd host, add the new host to the cluster. To add additional etcd members to the cluster, you must first adjust the default **localhost** peer in the **peerURLs** value for the first member:

- a. Get the member ID for the first member using the **member list** command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
/172.18.0.75:2379" \ 1
member list
```

- 1 Ensure that you specify the URLs of only active etcd members in the **--peers** parameter value.

- b. Obtain the IP address where etcd listens for cluster peers:

```
$ ss -ltn | grep 2380
```

- c. Update the value of **peerURLs** using the **etcdctl member update** command by passing the member ID and IP address obtained from the previous steps:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
/172.18.0.75:2379" \
member update 511b7fb6cc0001 https://172.18.1.18:2380
```

- d. Re-run the **member list** command and ensure the peer URLs no longer include **localhost**.

7. Add the new host to the etcd cluster. Note that the new host is not yet configured, so the status stays as **unstarted** until the you configure the new host.



WARNING

You must add each member and bring it online one at a time. When you add each additional member to the cluster, you must adjust the **peerURLs** list for the current peers. The **peerURLs** list grows by one for each member added. The **etcdctl member add** command outputs the values that you must set in the **etcd.conf** file as you add each member, as described in the following instructions.

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
```

```
--ca-file=/etc/etcd/ca.crt      \
--cert-file=/etc/etcd/peer.crt  \
--key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 1
```

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

```
ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https://<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

- 1 In this line, **10.3.9.222** is a label for the etcd member. You can specify the host name, IP address, or a simple name.

8. Update the sample `${PREFIX}/etcd.conf` file.

- a. Replace the following values with the values generated in the previous step:

- ETCD_NAME
- ETCD_INITIAL_CLUSTER
- ETCD_INITIAL_CLUSTER_STATE

- b. Modify the following variables with the new host IP from the output of the previous step. You can use `${NEW_ETCD_IP}` as the value.

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

- c. If you previously used the member system as an etcd node, you must overwrite the current values in the `/etc/etcd/etcd.conf` file.
- d. Check the file for syntax errors or missing IP addresses, otherwise the etcd service might fail:

```
# vi ${PREFIX}/etcd.conf
```

9. On the node that hosts the installation files, update the `[etcd]` hosts group in the `/etc/ansible/hosts` inventory file. Remove the old etcd hosts and add the new ones.

10. Create a `tgz` file that contains the certificates, the sample configuration file, and the `ca` and copy it to the new host:

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

Modify the new etcd host

1. Install **iptables-services** to provide iptables utilities to open the required ports for etcd:

```
# yum install -y iptables-services
```

2. Create the **OS_FIREWALL_ALLOW** firewall rules to allow etcd to communicate:

- Port 2379/tcp for clients
- Port 2380/tcp for peer communication

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2380 -j ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```



NOTE

In this example, a new chain **OS_FIREWALL_ALLOW** is created, which is the standard naming the OpenShift Container Platform installer uses for firewall rules.



WARNING

If the environment is hosted in an IaaS environment, modify the security groups for the instance to allow incoming traffic to those ports as well.

3. Install etcd:

```
# yum install -y etcd
```

Ensure version **etcd-2.3.7-4.el7.x86_64** or greater is installed,

4. Ensure the etcd service is not running:

```
# systemctl disable etcd --now
```

5. Remove any etcd configuration and data:

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. Extract the certificates and configuration files:

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. Modify the file ownership permissions:

```
# chown -R etcd/etcd /etc/etcd/*
# chown -R etcd/etcd /var/lib/etcd/
```

8. Start etcd on the new host:

```
# systemctl enable etcd --now
```

9. Verify that the host is part of the cluster and the current cluster health:

- If you use the v2 etcd api, run the following command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --peers="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

- If you use the v3 etcd api, run the following command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
  --key=/etc/etcd/peer.key \
  --cacert="/etc/etcd/ca.crt" \
  --endpoints="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed
proposal: took = 1.498829ms
```

Modify each OpenShift Container Platform master

1. Modify the master configuration in the **etcClientInfo** section of the **/etc/origin/master/master-config.yaml** file on every master. Add the new etcd host to the list of the etcd servers OpenShift Container Platform uses to store the data, and remove

any failed etcd hosts:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379
    - https://master-2.example.com:2379
    - https://etcd0.example.com:2379
```

2. Restart the master API service:

- On every master:

```
# systemctl restart atomic-openshift-master-api
```

- Or, on a single master cluster installation:

```
# systemctl restart atomic-openshift-master
```



WARNING

The number of etcd nodes must be odd, so you must add at least two hosts.

3. If you use Flannel, modify the **flanneld** service configuration located at **/etc/sysconfig/flanneld** on every OpenShift Container Platform host to include the new etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

CHAPTER 5. MANAGING USERS

5.1. OVERVIEW

A user is an entity that interacts with the OpenShift Container Platform API. These can be a developer for developing applications or an administrator for managing the cluster. Users can be assigned to groups, which set the permissions applied to all the group's members. For example, you can give API access to a group, which give all members of the group API access.

This topic describes the management of [user](#) accounts, including how new user accounts are created in OpenShift Container Platform and how they can be deleted.

5.2. CREATING A USER

The process for creating a user depends on the configured [identity provider](#). By default, OpenShift Container Platform uses the **DenyAll** identity provider, which denies access for all user names and passwords.

The following process creates a new user, then adds a role to the user:

1. Create the user account depending on your identity provider. This can depend on the **mappingmethod** used as part of the [identity provider configuration](#).
2. Give the new user the desired role:

```
# oc create clusterrolebinding <clusterrolebinding_name> \
  --clusterrole=<role> --user=<user>
```

Where the **--clusterrole** option is the desired cluster role. For example, to give the new user **cluster-admin** privileges, which gives the user access to everything within a cluster:

```
# oc create clusterrolebinding registry-controller \
  --clusterrole=cluster-admin --user=admin
```

For an explanation and list of roles, see the [Cluster Roles and Local Roles section of the Architecture Guide](#).

As a cluster administrator, you can also [manage the access level of each user](#).



NOTE

Depending on the identity provider, and on the defined group structure, some roles may be given to users automatically. See the [Synching groups with LDAP section](#) for more information.

5.3. VIEWING USER AND IDENTITY LISTS

OpenShift Container Platform user configuration is stored in several locations within OpenShift Container Platform. Regardless of the identity provider, OpenShift Container Platform internally stores details like role-based access control (RBAC) information and group membership. To completely remove user information, this data must be removed in addition to the user account.

In OpenShift Container Platform, two object types contain user data outside the identification provider: **user** and **identity**.

To get the current list of users:

```
$ oc get user
```

NAME	UID	FULL NAME	IDENTITIES
demo	75e4b80c-dbf1-11e5-8dc6-0e81e52cc949		
htpasswd_auth:demo			

To get the current list of identities:

```
$ oc get identity
```

NAME	UID	IDP NAME	IDP USER NAME	USER NAME	USER
htpasswd_auth:demo	75e4b80c-dbf1-11e5-8dc6-0e81e52cc949	htpasswd_auth	demo	demo	

Note the matching UID between the two object types. If you attempt to change the authentication provider after starting to use OpenShift Container Platform, the user names that overlap will not work because of the entries in the identity list, which will still point to the old authentication method.

5.4. CREATING GROUPS

While a user is an entity making requests to OpenShift Container Platform, users can be organized into one or more groups made up from a set of users. Groups are useful for managing many users at one time, such as for authorization policies, or to grant permissions to multiple users at once.

If your organization is using LDAP, you can synchronize any LDAP records to OpenShift Container Platform so that you can configure groups on one place. This presumes that information about your users is in an MDAP server. See the [Synching groups with LDAP section](#) for more information.

If you are not using LDAP, you can use the following procedure to manually create groups.

To create a new group:

```
# oc adm groups new <group_name> <user1> <user2>
```

For example, to create the **west** groups and in it place the **john** and **betty** users:

```
# oc adm groups new west john betty
```

To verify that the group has been created, and list the users associated with the group, run the following:

```
# oc get groups
```

NAME	USERS
west	john, betty

Next steps:

- [Managing role bindings](#)

5.5. MANAGING USER AND GROUP LABELS

To add a label to a user or group:

```
$ oc label user/<user_name> <label_name>
```

For example, if the user name is **theuser** and the label is **level=gold**:

```
$ oc label user/theuser level=gold
```

To remove the label:

```
$ oc label user/<user_name> <label_name>-
```

To show labels for a user or group:

```
$ oc describe user/<user_name>
```

5.6. DELETING A USER

To delete a user:

1. Delete the user record:

```
$ oc delete user demo
user "demo" deleted
```

2. Delete the user identity.

The identity of the user is related to the identification provider you use. Get the provider name from the user record in **oc get user**.

In this example, the identity provider name is **htpasswd_auth**. The command is:

```
# oc delete identity htpasswd_auth:demo
identity "htpasswd_auth:demo" deleted
```

If you skip this step, the user will not be able to log in again.

After you complete these steps, a new account will be created in OpenShift Container Platform when the user logs in again.

If your intention is to prevent the user from being able to log in again (for example, if an employee has left the company and you want to permanently delete the account), you can also remove the user from your authentication back end (like **htpasswd**, **kerberos**, or others) for the configured identity provider.

For example, if you are using **htpasswd**, delete the entry in the **htpasswd** file that is configured for OpenShift Container Platform with the user name and password.

For external identification management like Lightweight Directory Access Protocol (LDAP) or Red Hat Identity Management (IdM), use the user management tools to remove the user entry.

CHAPTER 6. MANAGING PROJECTS

6.1. OVERVIEW

In OpenShift Container Platform, projects are used to group and isolate related objects. As an administrator, you can give developers access to certain projects, allow them to create their own, and give them administrative rights within individual projects.

6.2. SELF-PROVISIONING PROJECTS

You can allow developers to create their own projects. There is an endpoint that will provision a project according to a [template](#). The web console and `oc new-project` command use this endpoint when a developer [creates a new project](#).

6.2.1. Modifying the Template for New Projects

The API server automatically provisions projects based on the template that is identified by the `projectRequestTemplate` parameter of the [master-config.yaml](#) file. If the parameter is not defined, the API server creates a default template that creates a project with the requested name, and assigns the requesting user to the "admin" role for that project.

To create your own custom project template:

1. Start with the current default project template:

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

2. Use a text editor to modify the **template.yaml** file by adding objects or modifying existing objects.

3. Load the template:

```
$ oc create -f template.yaml -n default
```

4. Modify the **master-config.yaml** file to reference the loaded template:

```
...
projectConfig:
  projectRequestTemplate: "default/project-request"
...
```

When a project request is submitted, the API substitutes the following parameters into the template:

Parameter	Description
PROJECT_NAME	The name of the project. Required.
PROJECT_DISPLAYNAME	The display name of the project. May be empty.
PROJECT_DESCRIPTION	The description of the project. May be empty.

Parameter	Description
PROJECT_ADMIN_USER	The username of the administrating user.
PROJECT_REQUESTING_USER	The username of the requesting user.

Access to the API is granted to developers with the [self-provisioner](#) role and the **self-provisioners** cluster role binding. This role is available to all authenticated developers by default.

6.2.2. Disabling Self-provisioning

You can prevent an authenticated user group from self-provisioning new projects.

1. Log in as a user with [cluster-admin](#) privileges.
2. Remove the **self-provisioners** [cluster role](#) from the group.

```
$ oc adm policy remove-cluster-role-from-group self-provisioner
system:authenticated system:authenticated:oauth
```

3. Set the **projectRequestMessage** parameter value in the *master-config.yaml* file to instruct developers how to request a new project. This parameter value is a string that will be presented to a user in the web console and command line when the user attempts to self-provision a project. You might use one of the following messages:
 - To request a project, contact your system administrator at projectname@example.com.
 - To request a new project, fill out the project request form located at <https://internal.example.com/openshift-project-request>.

Example YAML file

```
...
projectConfig:
  ProjectRequestMessage: "message"
...
```

4. Edit the **self-provisioners** cluster role binding to prevent [automatic updates](#) to the role. Automatic updates reset the cluster roles to the default state.

- To update the role binding from the command line:

- i. Run the following command:

```
$ oc edit clusterrolebinding.rbac self-provisioners
```

- ii. In the displayed role binding, set the **rbac.authorization.kubernetes.io/autoupdate** parameter value to **false**, as shown in the following example:

```
apiVersion: authorization.openshift.io/v1
```



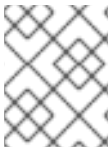
```
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false"
...
```

- To update the role binding by using a single command:

```
$ oc patch clusterrolebinding.rbac self-provisioners -p '{
"metadata": { "annotations": {
"rbac.authorization.kubernetes.io/autoupdate": "false" } } }'
```

6.3. USING NODE SELECTORS

Node selectors are used in conjunction with labeled nodes to control pod placement.



NOTE

Labels can be assigned [during an advanced installation](#), or [added to a node after installation](#).

6.3.1. Setting the Cluster-wide Default Node Selector

As a cluster administrator, you can set the cluster-wide default node selector to restrict pod placement to specific nodes.

Edit the master configuration file at */etc/origin/master/master-config.yaml* and add a value for a default node selector. This is applied to the pods created in all projects without a specified **nodeSelector** value:

```
...
projectConfig:
  defaultNodeSelector: "type=user-node,region=east"
...
```

Restart the OpenShift service for the changes to take effect:

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-
controllers
```

6.3.2. Setting the Project-wide Node Selector

To create an individual project with a node selector, use the **--node-selector** option when creating a project. For example, if you have an OpenShift Container Platform topology with multiple regions, you can use a node selector to restrict specific OpenShift Container Platform projects to only deploy pods onto nodes in a specific region.

The following creates a new project named **myproject** and dictates that pods be deployed onto nodes labeled **user-node** and **east**:

```
$ oc adm new-project myproject \
  --node-selector='type=user-node,region=east'
```

Once this command is run, this becomes the administrator-set node selector for all pods contained in the specified project.



NOTE

While the **new-project** subcommand is available for both **oc adm** and **oc**, the cluster administrator and developer commands respectively, creating a new project with a node selector is only available with the **oc adm** command. The **new-project** subcommand is not available to project developers when self-provisioning projects.

Using the **oc adm new-project** command adds an **annotation** section to the project. You can edit a project, and change the **openshift.io/node-selector** value to override the default:

```
...
metadata:
  annotations:
    openshift.io/node-selector: type=user-node,region=east
...
```

You can also override the default value for an existing project namespace by using the following command:

```
# oc patch namespace myproject -p \
  '{"metadata":{"annotations":{"openshift.io/node-selector":"region=infra"}}}'
```

If **openshift.io/node-selector** is set to an empty string (**oc adm new-project --node-selector=""**), the project will not have an administrator-set node selector, even if the cluster-wide default has been set. This means that, as a cluster administrator, you can set a default to restrict developer projects to a subset of nodes and still enable infrastructure or other projects to schedule the entire cluster.

6.3.3. Developer-specified Node Selectors

OpenShift Container Platform developers [can set a node selector on their pod configuration](#) if they wish to restrict nodes even further. This will be in addition to the project node selector, meaning that you can still dictate node selector values for all projects that have a node selector value.

For example, if a project has been created with the above annotation (**openshift.io/node-selector: type=user-node,region=east**) and a developer sets another node selector on a pod in that project, for example **clearance=classified**, the pod will only ever be scheduled on nodes that have all three labels (**type=user-node**, **region=east**, and **clearance=classified**). If they set **region=west** on a pod, their pods would be demanding nodes with labels **region=east** and **region=west**, which cannot work. The pods will never be scheduled, because labels can only be set to one value.

6.4. LIMITING NUMBER OF SELF-PROVISIONED PROJECTS PER USER

The number of self-provisioned projects requested by a given user can be limited with the **ProjectRequestLimit** [admission control plug-in](#).



IMPORTANT

If your project request template was created in OpenShift Container Platform 3.1 or earlier using the process described in [Modifying the Template for New Projects](#), then the generated template does not include the annotation **openshift.io/requester: \${PROJECT_REQUESTING_USER}**, which is used for the **ProjectRequestLimitConfig**. You must add the annotation.

In order to specify limits for users, a configuration must be specified for the plug-in within the master configuration file (**/etc/origin/master/master-config.yaml**). The plug-in configuration takes a list of user label selectors and the associated maximum project requests.

Selectors are evaluated in order. The first one matching the current user will be used to determine the maximum number of projects. If a selector is not specified, a limit applies to all users. If a maximum number of projects is not specified, then an unlimited number of projects are allowed for a specific selector.

The following configuration sets a global limit of 2 projects per user while allowing 10 projects for users with a label of **level=advanced** and unlimited projects for users with a label of **level=admin**.

```
admissionConfig:
  pluginConfig:
    ProjectRequestLimit:
      configuration:
        apiVersion: v1
        kind: ProjectRequestLimitConfig
        limits:
          - selector:
              level: admin ❶
          - selector:
              level: advanced ❷
            maxProjects: 10
          - maxProjects: 2 ❸
```

- ❶ For selector **level=admin**, no **maxProjects** is specified. This means that users with this label will not have a maximum of project requests.
- ❷ For selector **level=advanced**, a maximum number of 10 projects will be allowed.
- ❸ For the third entry, no selector is specified. This means that it will be applied to any user that doesn't satisfy the previous two rules. Because rules are evaluated in order, this rule should be specified last.



NOTE

[Managing User and Group Labels](#) provides further guidance on how to add, remove, or show labels for users and groups.

Once your changes are made, restart OpenShift Container Platform for the changes to take effect.

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

CHAPTER 7. MANAGING PODS

7.1. OVERVIEW

This topic describes the management of [pods](#), including limiting their run-once duration, and how much bandwidth they can use.

7.2. VIEWING PODS

You can display usage statistics about pods, which provide the runtime environments for containers. These usage statistics include CPU, memory, and storage consumption.

To view the usage statistics:

```
$ oc adm top pods
```

NAME	CPU(cores)	MEMORY(bytes)
hawkular-cassandra-1-pqx6l	219m	1240Mi
hawkular-metrics-rddnv	20m	1765Mi
heapster-n94r4	3m	37Mi

To view the usage statistics for pods with labels:

```
$ oc adm top pod --selector=''
```

You must choose the selector (label query) to filter on. Supports `=`, `==`, and `!=`.



NOTE

You must have **cluster-reader** permission to view the usage statistics.



NOTE

Metrics must be installed to view the usage statistics.

7.3. LIMITING RUN-ONCE POD DURATION

OpenShift Container Platform relies on run-once pods to perform tasks such as deploying a pod or performing a build. Run-once pods are pods that have a **RestartPolicy** of **Never** or **OnFailure**.

The cluster administrator can use the **RunOnceDuration** admission control plug-in to force a limit on the time that those run-once pods can be active. Once the time limit expires, the cluster will try to actively terminate those pods. The main reason to have such a limit is to prevent tasks such as builds to run for an excessive amount of time.

7.3.1. Configuring the RunOnceDuration Plug-in

The plug-in configuration should include the default active deadline for run-once pods. This deadline is enforced globally, but can be superseded on a per-project basis.

```
admissionConfig:
  pluginConfig:
    RunOnceDuration:
```

```

configuration:
  apiVersion: v1
  kind: RunOnceDurationConfig
  activeDeadlineSecondsOverride: 3600 1
....

```

- 1 Specify the global default for run-once pods in seconds.

7.3.2. Specifying a Custom Duration per Project

In addition to specifying a global maximum duration for run-once pods, an administrator can add an annotation (**openshift.io/active-deadline-seconds-override**) to a specific project to override the global default.

- For a new project, define the annotation in the project specification *.yaml* file.

```

apiVersion: v1
kind: Project
metadata:
  annotations:
    openshift.io/active-deadline-seconds-override: "1000" 1
  name: myproject

```

- 1 Overrides the default active deadline seconds for run-once pods to 1000 seconds. Note that the value of the override must be specified in string form.

- For an existing project,
 - Run **oc edit** and add the **openshift.io/active-deadline-seconds-override: 1000** annotation in the editor.

```
$ oc edit namespace <project-name>
```

Or

- Use the **oc patch** command:

```
$ oc patch namespace <project_name> -p '{"metadata":
{"annotations":{"openshift.io/active-deadline-seconds-
override":"1000"}}}'
```

7.3.2.1. Deploying an Egress Router Pod

Example 7.1. Example Pod Definition for an Egress Router

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:

```

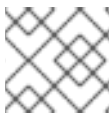
```

    pod.network.openshift.io/assign-macvlan: "true"
spec:
  containers:
  - name: egress-router
    image: openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE ❶
        value: 192.168.12.99
      - name: EGRESS_GATEWAY ❷
        value: 192.168.12.1
      - name: EGRESS_DESTINATION ❸
        value: 203.0.113.25
    nodeSelector:
      site: springfield-1 ❹

```

- ❶ IP address on the node subnet reserved by the cluster administrator for use by this pod.
- ❷ Same value as the default gateway used by the node itself.
- ❸ Connections to the pod are redirected to 203.0.113.25, with a source IP address of 192.168.12.99
- ❹ The pod will only be deployed to nodes with the label site **springfield-1**.

The **pod.network.openshift.io/assign-macvlan** annotation creates a Macvlan network interface on the primary network interface, and then moves it into the pod's network name space before starting the **egress-router** container.



NOTE

Preserve the the quotation marks around **"true"**. Omitting them will result in errors.

The pod contains a single container, using the **openshift3/ose-egress-router** image, and that container is run privileged so that it can configure the Macvlan interface and set up **iptables** rules.

The environment variables tell the **egress-router** image what addresses to use; it will configure the Macvlan interface to use **EGRESS_SOURCE** as its IP address, with **EGRESS_GATEWAY** as its gateway.

NAT rules are set up so that connections to any TCP or UDP port on the pod's cluster IP address are redirected to the same port on **EGRESS_DESTINATION**.

If only some of the nodes in your cluster are capable of claiming the specified source IP address and using the specified gateway, you can specify a **nodeName** or **nodeSelector** indicating which nodes are acceptable.

7.3.2.2. Deploying an Egress Router Service

Though not strictly necessary, you normally want to create a service pointing to the egress router:

```
apiVersion: v1
```

```

kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1

```

Your pods can now connect to this service. Their connections are redirected to the corresponding ports on the external server, using the reserved egress IP address.

7.3.3. Limiting Pod Access with Egress Firewall

As an OpenShift Container Platform cluster administrator, you can use egress policy to limit the external addresses that some or all pods can access from within the cluster, so that:

- A pod can only talk to internal hosts, and cannot initiate connections to the public Internet.
Or,
- A pod can only talk to the public Internet, and cannot initiate connections to internal hosts (outside the cluster).
Or,
- A pod cannot reach specified internal subnets/hosts that it should have no reason to contact.

For example, you can configure projects with different egress policies, allowing **<project A>** access to a specified IP range, but denying the same access to **<project B>**.

CAUTION

You must have the [ovs-multitenant plug-in](#) enabled in order to limit pod access via egress policy.

Project administrators can neither create **EgressNetworkPolicy** objects, nor edit the ones you create in their project. There are also several other restrictions on where **EgressNetworkPolicy** can be created:

1. The **default** project (and any other project that has been made global via **oc adm pod-network make-projects-global**) cannot have egress policy.
2. If you merge two projects together (via **oc adm pod-network join-projects**), then you cannot use egress policy in *any* of the joined projects.
3. No project may have more than one egress policy object.

Violating any of these restrictions will result in broken egress policy for the project, and may cause all external network traffic to be dropped.

7.3.3.1. Configuring Pod Access Limits

To configure pod access limits, you must use the **oc** command or the REST API. You can use **oc [create|replace|delete]** to manipulate **EgressNetworkPolicy** objects. The *api/swagger-spec/oapi-v1.json* file has API-level details on how the objects actually work.

To configure pod access limits:

1. Navigate to the project you want to affect.
2. Create a JSON file for the pod limit policy:

```
# oc create -f <policy>.json
```

3. Configure the JSON file with policy details. For example:

```
{
  "kind": "EgressNetworkPolicy",
  "apiVersion": "v1",
  "metadata": {
    "name": "default"
  },
  "spec": {
    "egress": [
      {
        "type": "Allow",
        "to": {
          "cidrSelector": "1.2.3.0/24"
        }
      },
      {
        "type": "Allow",
        "to": {
          "dnsName": "www.foo.com"
        }
      },
      {
        "type": "Deny",
        "to": {
          "cidrSelector": "0.0.0.0/0"
        }
      }
    ]
  }
}
```

When the example above is added in a project, it allows traffic to IP range **1.2.3.0/24** and domain name **www.foo.com**, but denies access to all other external IP addresses. (Traffic to other pods is not affected because the policy only applies to *external* traffic.)

The rules in an **EgressNetworkPolicy** are checked in order, and the first one that matches takes effect. If the three rules in the above example were reversed, then traffic would not be allowed to **1.2.3.0/24** and **www.foo.com** because the **0.0.0.0/0** rule would be checked first, and it would match and deny all traffic.

Domain name updates are reflected within 30 minutes. In the above example, suppose **www.foo.com** resolved to **10.11.12.13**, but later it was changed to **20.21.22.23**. Then, OpenShift Container Platform will take up to 30 minutes to adapt to these DNS updates.

7.4. LIMITING THE BANDWIDTH AVAILABLE TO PODS

You can apply quality-of-service traffic shaping to a pod and effectively limit its available bandwidth. Egress traffic (from the pod) is handled by policing, which simply drops packets in excess of the configured rate. Ingress traffic (to the pod) is handled by shaping queued packets to effectively handle data. The limits you place on a pod do not affect the bandwidth of other pods.

To limit the bandwidth on a pod:

1. Write an object definition JSON file, and specify the data traffic speed using **kubernetes.io/ingress-bandwidth** and **kubernetes.io/egress-bandwidth** annotations. For example, to limit both pod egress and ingress bandwidth to 10M/s:

Limited Pod Object Definition

```
{
  "kind": "Pod",
  "spec": {
    "containers": [
      {
        "image": "openshift/hello-openshift",
        "name": "hello-openshift"
      }
    ]
  },
  "apiVersion": "v1",
  "metadata": {
    "name": "iperf-slow",
    "annotations": {
      "kubernetes.io/ingress-bandwidth": "10M",
      "kubernetes.io/egress-bandwidth": "10M"
    }
  }
}
```

2. Create the pod using the object definition:

```
oc create -f <file_or_dir_path>
```

7.5. SETTING POD DISRUPTION BUDGETS

A *pod disruption budget* is part of the [Kubernetes API](#), which can be managed with **oc** commands like other [object types](#). They allow the specification of safety constraints on pods during operations, such as draining a node for maintenance.



NOTE

Starting in OpenShift Container Platform 3.6, pod disruption budgets are now fully supported.

PodDisruptionBudget is an API object that specifies the minimum number or percentage of replicas that must be up at a time. Setting these in projects can be helpful during node maintenance (such as scaling a cluster down or a cluster upgrade) and is only honored on voluntary evictions (not on node failures).

A **PodDisruptionBudget** object's configuration consists of the following key parts:

- A label selector, which is a label query over a set of pods.
- An availability level, which specifies the minimum number of pods that must be available simultaneously.

The following is an example of a **PodDisruptionBudget** resource:

```
apiVersion: policy/v1beta1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  selector: 2
    matchLabels:
      foo: bar
  minAvailable: 2 3
```

- 1** **PodDisruptionBudget** is part of the **policy/v1beta1** API group.
- 2** A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined.
- 3** The minimum number of pods that must be available simultaneously. This can be either an integer or a string specifying a percentage (for example, **20%**).

If you created a YAML file with the above object definition, you could add it to project with the following:

```
$ oc create -f </path/to/file> -n <project_name>
```

You can check for pod disruption budgets across all projects with the following:

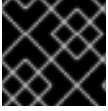
```
$ oc get poddisruptionbudget --all-namespaces
```

NAMESPACE	NAME	MIN-AVAILABLE	SELECTOR
another-project	another-pdb	4	bar=foo
test-project	my-pdb	2	foo=bar

The **PodDisruptionBudget** is considered healthy when there are at least **minAvailable** pods running in the system. Every pod above that limit can be [evicted](#).

7.6. INJECTING INFORMATION INTO PODS USING POD PRESETS

A *pod preset* is an object that injects user-specified information into pods as they are created.



IMPORTANT

As of OpenShift Container Platform 3.7, pod presets are no longer supported.

Using pod preset objects you can inject:

- [secret objects](#)
- [ConfigMap objects](#)
- [storage volumes](#)
- container volume mounts
- environment variables

Developers only need make sure the pod labels match the label selector on the PodPreset in order to add all that information to the pod. The [label](#) on a pod associates the pod with one or more pod preset objects that have a matching [label selectors](#).

Using pod presets, a developer can provision pods without needing to know the details about the services the pod will consume. An administrator can keep configuration items of a service invisible from a developer without preventing the developer from deploying pods. For example, an administrator can create a pod preset that provides the name, user name, and password for a database through a secret and the database port through environment variables. The pod developer only needs to know the label to use to include all the information in pods. A developer can also create pod presets and perform all the same tasks. For example, the developer can create a preset that injects environment variable automatically into multiple pods.



NOTE

The Pod Preset feature is available only if the [Service Catalog](#) has been installed.

You can exclude specific pods from being injected using the **podpreset.admission.kubernetes.io/exclude: "true"** parameter in the pod specification. See the [example pod specification](#).

For more information, see [Injecting Information into Pods Using Pod Presets](#).

CHAPTER 8. MANAGING NETWORKING

8.1. OVERVIEW

This topic describes the management of the overall [cluster network](#), including project isolation and outbound traffic control.

Pod-level networking features, such as per-pod bandwidth limits, are discussed in [Managing Pods](#).

8.2. MANAGING POD NETWORKS

When your cluster is configured to use [the ovs-multitenant SDN plugin](#), you can manage the separate pod overlay networks for projects using the administrator CLI. See the [Configuring the SDN](#) section for plug-in configuration steps, if necessary.

8.2.1. Joining Project Networks

To join projects to an existing project network:

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

In the above example, all the pods and services in **<project2>** and **<project3>** can now access any pods and services in **<project1>** and vice versa. Services can be accessed either by IP or fully-qualified DNS name (**<service>.<pod_namespace>.svc.cluster.local**). For example, to access a service named **db** in a project **myproject**, use **db.myproject.svc.cluster.local**.

Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option.

To verify the networks you have joined together:

```
$ oc get netnamespaces
```

Then look at the **NETID** column. Projects in the same pod-network will have the same NetID.

8.3. ISOLATING PROJECT NETWORKS

To isolate the project network in the cluster and vice versa, run:

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

In the above example, all of the pods and services in **<project1>** and **<project2>** can *not* access any pods and services from other non-global projects in the cluster and vice versa.

Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option.

8.3.1. Making Project Networks Global

To allow projects to access all pods and services in the cluster and vice versa:

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

In the above example, all the pods and services in **<project1>** and **<project2>** can now access any pods and services in the cluster and vice versa.

Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option.

8.4. DISABLING HOST NAME COLLISION PREVENTION FOR ROUTES AND INGRESS OBJECTS

In OpenShift Container Platform, host name collision prevention for routes and ingress objects is enabled by default. This means that users without the **cluster-admin** role can set the host name in a route or ingress object only on creation and cannot change it afterwards. However, you can relax this restriction on routes and ingress objects for some or all users.



WARNING

Because OpenShift Container Platform uses the object creation timestamp to determine the oldest route or ingress object for a given host name, a route or ingress object can hijack a host name of a newer route if the older route changes its host name, or if an ingress object is introduced.

As an OpenShift Container Platform cluster administrator, you can edit the host name in a route even after creation. You can also create a role to allow specific users to do so:

```
$ oc create -f - <<EOF
apiVersion: v1
kind: ClusterRole
metadata:
  name: route-editor
rules:
- apiGroups:
  - route.openshift.io
  - ""
  resources:
  - routes/custom-host
  verbs:
  - update
EOF
```

You can then bind the new role to a user:

```
$ oc adm policy add-cluster-role-to-user route-editor user
```

You can also disable host name collision prevention for ingress objects. Doing so lets users without the **cluster-admin** role edit a host name for ingress objects after creation. This is useful to OpenShift Container Platform installations that depend upon Kubernetes behavior, including allowing the host

names in ingress objects be edited.

1. Add the following to the **master.yaml** file:

```
admissionConfig:
  pluginConfig:
    openshift.io/IngressAdmission:
      configuration:
        apiVersion: v1
        allowHostnameChanges: true
        kind: IngressAdmissionConfig
        location: ""
```

2. Restart the master services for the changes to take effect:

```
$ systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

8.5. CONTROLLING EGRESS TRAFFIC

As a cluster administrator you can allocate a number of static IP addresses to a specific node at the host level. If an application developer needs a dedicated IP address for their application service, they can request one during the process they use to ask for firewall access. They can then deploy an egress router from the developer's project, using a **nodeSelector** in the deployment configuration to ensure that the pod lands on the host with the pre-allocated static IP address.

The egress pod's deployment declares one of the source IPs, the destination IP of the protected service, and a gateway IP to reach the destination. After the pod is deployed, you can [create a service](#) to access the egress router pod, then add that source IP to the corporate firewall. The developer then has access information to the egress router service that was created in their project, for example, **service.project.cluster.domainname.com**.

When the developer needs to access the external, firewalled service, they can call out to the egress router pod's service (**service.project.cluster.domainname.com**) in their application (for example, the JDBC connection information) rather than the actual protected service URL.

You can also assign static IP addresses to projects, ensuring that all outgoing external connections from the specified project have recognizable origins. This is different from the default egress router, which is used to send traffic to specific destinations. See the [Enabling Fixed IPs for External Project Traffic](#) section for more information.



NOTE

The egress router is not available for OpenShift Dedicated.

As an OpenShift Container Platform cluster administrator, you can control egress traffic in these ways:

Firewall

Using an egress firewall allows you to enforce the acceptable outbound traffic policies, so that specific endpoints or IP ranges (subnets) are the only acceptable targets for the dynamic endpoints (pods within OpenShift Container Platform) to talk to.

Router

Using an egress router allows you to create identifiable services to send traffic to certain destinations,

ensuring those external destinations treat traffic as though it were coming from a known source. This helps with security, because it allows you to secure an external database so that only specific pods in a namespace can talk to a service (the egress router), which proxies the traffic to your database.

iptables

In addition to the above OpenShift Container Platform-internal solutions, it is also possible to create iptables rules that will be applied to outgoing traffic. These rules allow for more possibilities than the egress firewall, but cannot be limited to particular projects.

8.5.1. Using an Egress Firewall to Limit Access to External Resources

As an OpenShift Container Platform cluster administrator, you can use egress firewall policy to limit the external addresses that some or all pods can access from within the cluster, so that:

- A pod can only talk to internal hosts, and cannot initiate connections to the public Internet.
Or,
- A pod can only talk to the public Internet, and cannot initiate connections to internal hosts (outside the cluster).
Or,
- A pod cannot reach specified internal subnets/hosts that it should have no reason to contact.

You can configure projects to have different egress policies. For example, allowing **<project A>** access to a specified IP range, but denying the same access to **<project B>**. Or restrict application developers from updating from (Python) pip mirrors, and forcing updates to only come from desired sources.

CAUTION

You must have the [ovs-multitenant](#) plugin enabled in order to limit pod access via egress policy.

Project administrators can neither create **EgressNetworkPolicy** objects, nor edit the ones you create in their project. There are also several other restrictions on where **EgressNetworkPolicy** can be created:

- The **default** project (and any other project that has been made global via **oc adm pod-network make-projects-global**) cannot have egress policy.
- If you merge two projects together (via **oc adm pod-network join-projects**), then you cannot use egress policy in *any* of the joined projects.
- No project may have more than one egress policy object.

Violating any of these restrictions results in broken egress policy for the project, and may cause all external network traffic to be dropped.

Use the **oc** command or the REST API to configure egress policy. You can use **oc [create|replace|delete]** to manipulate **EgressNetworkPolicy** objects. The **api/swagger-spec/oapi-v1.json** file has API-level details on how the objects actually work.

To configure egress policy:

1. Navigate to the project you want to affect.

2. Create a JSON file with the desired policy details. For example:

```
{
  "kind": "EgressNetworkPolicy",
  "apiVersion": "v1",
  "metadata": {
    "name": "default"
  },
  "spec": {
    "egress": [
      {
        "type": "Allow",
        "to": {
          "cidrSelector": "1.2.3.0/24"
        }
      },
      {
        "type": "Allow",
        "to": {
          "dnsName": "www.foo.com"
        }
      },
      {
        "type": "Deny",
        "to": {
          "cidrSelector": "0.0.0.0/0"
        }
      }
    ]
  }
}
```

When the example above is added to a project, it allows traffic to IP range **1.2.3.0/24** and domain name **www.foo.com**, but denies access to all other external IP addresses. Traffic to other pods is not affected because the policy only applies to *external* traffic.

The rules in an **EgressNetworkPolicy** are checked in order, and the first one that matches takes effect. If the three rules in the above example were reversed, then traffic would not be allowed to **1.2.3.0/24** and **www.foo.com** because the **0.0.0.0/0** rule would be checked first, and it would match and deny all traffic.

Domain name updates are polled based on the TTL (time to live) value of the domain returned by the local non-authoritative servers. The pod should also resolve the domain from the same local nameservers when necessary, otherwise the IP addresses for the domain perceived by the egress network policy controller and the pod will be different, and the egress network policy may not be enforced as expected. Since egress network policy controller and pod are asynchronously polling the same local nameserver, there could be a race condition where pod may get the updated IP before the egress controller. Due to this current limitation, domain name usage in **EgressNetworkPolicy** is only recommended for domains with infrequent IP address changes.

**NOTE**

The egress firewall always allows pods access to the external interface of the node the pod is on for DNS resolution. If your DNS resolution is not handled by something on the local node, then you will need to add egress firewall rules allowing access to the DNS server's IP addresses if you are using domain names in your pods. The [default installer](#) sets up a local dnsmasq, so if you are using that setup you will not need to add extra rules.

3. Use the JSON file to create an EgressNetworkPolicy object:

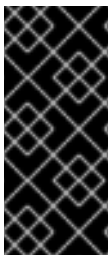
```
$ oc create -f <policy>.json
```

CAUTION

Exposing services by creating [routes](#) will ignore **EgressNetworkPolicy**. Egress network policy service endpoint filtering is done at the node **kubeproxy**. When the router is involved, **kubeproxy** is bypassed and egress network policy enforcement is not applied. Administrators can prevent this bypass by limiting access to create routes.

8.5.2. Using an Egress Router to Allow External Resources to Recognize Pod Traffic

The OpenShift Container Platform egress router runs a service that redirects traffic to a specified remote server, using a private source IP address that is not used for anything else. The service allows pods to talk to servers that are set up to only allow access from whitelisted IP addresses.

**IMPORTANT**

The egress router is not intended for every outgoing connection. Creating large numbers of egress routers can push the limits of your network hardware. For example, creating an egress router for every project or application could exceed the number of local MAC addresses that the network interface can handle before falling back to filtering MAC addresses in software.

**IMPORTANT**

Currently, the egress router is not compatible with Amazon AWS, Azure Cloud, or any other cloud platform that does not support layer 2 manipulations due to their incompatibility with macvlan traffic.

Deployment Considerations

The Egress router adds a second IP address and MAC address to the node's primary network interface. If you are not running OpenShift Container Platform on bare metal, you may need to configure your hypervisor or cloud provider to allow the additional address.

Red Hat OpenStack Platform

If you are deploying OpenShift Container Platform on Red Hat OpenStack Platform, you need to whitelist the IP and MAC addresses on your OpenStack environment, otherwise [communication will fail](#):

```
neutron port-update $neutron_port_uuid \
  --allowed_address_pairs list=true \
  type=dict mac_address=<mac_address>,ip_address=<ip_address>
```

Red Hat Enterprise Virtualization

If you are using [Red Hat Enterprise Virtualization](#), you should set **EnableMACAntiSpoofingFilterRules** to **false**.

VMware vSphere

If you are using VMware vSphere, see the [VMWare documentation for securing vSphere standard switches](#). View and change VMWare vSphere default settings by selecting the host's virtual switch from the vSphere Web Client.

Specifically, ensure that the following are enabled:

- [MAC Address Changes](#)
- [Forged Transits](#)
- [Promiscuous Mode Operation](#)

Egress Router Modes

The egress router can run in two different modes: [redirect mode](#) and [HTTP proxy mode](#). Redirect mode works for all services except for HTTP and HTTPS. For HTTP and HTTPS services, use HTTP proxy mode.

8.5.2.1. Deploying an Egress Router Pod in Redirect Mode

In *redirect mode*, the egress router sets up iptables rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that want to make use of the reserved source IP address must be modified to connect to the egress router rather than connecting directly to the destination IP.

1. Create a pod configuration using the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE ❷
    value: 192.168.12.99
  - name: EGRESS_GATEWAY ❸
    value: 192.168.12.1
  - name: EGRESS_DESTINATION ❹
```

```

    value: 203.0.113.25
  - name: EGRESS_ROUTER_MODE 5
    value: init
containers:
  - name: egress-router-wait
    image: registry.access.redhat.com/openshift3/ose-pod
nodeSelector:
  site: springfield-1 6

```

- 1 The **pod.network.openshift.io/assign-macvlan** annotation creates a Macvlan network interface on the primary network interface, and then moves it into the pod's network name space before starting the **egress-router** container. Preserve the quotation marks around **"true"**. Omitting them results in errors.
- 2 IP address from the physical network that the node is on and is reserved by the cluster administrator for use by this pod.
- 3 Same value as the default gateway used by the node.
- 4 The external server to direct traffic to. Using this example, connections to the pod are redirected to 203.0.113.25, with a source IP address of 192.168.12.99.
- 5 This tells the egress router image that it is being deployed as an "init container". Previous versions of OpenShift Container Platform (and the egress router image) did not support this mode and had to be run as an ordinary container.
- 6 The pod is only deployed to nodes with the label **site=springfield-1**.

2. Create the pod using the above definition:

```
$ oc create -f <pod_name>.json
```

To check to see if the pod has been created:

```
$ oc get pod <pod_name>
```

3. Ensure other pods can find the pod's IP address by creating a service to point to the egress router:

```

apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
  type: ClusterIP
  selector:
    name: egress-1

```

Your pods can now connect to this service. Their connections are redirected to the corresponding ports on the external server, using the reserved egress IP address.

The egress router setup is performed by an "init container" created from the **openshift3/ose-egress-router** image, and that container is run privileged so that it can configure the Macvlan interface and set up **iptables** rules. After it finishes setting up the **iptables** rules, it exits and the **openshift3/ose-pod** container will run (doing nothing) until the pod is killed.

The environment variables tell the **egress-router** image what addresses to use; it will configure the Macvlan interface to use **EGRESS_SOURCE** as its IP address, with **EGRESS_GATEWAY** as its gateway.

NAT rules are set up so that connections to any TCP or UDP port on the pod's cluster IP address are redirected to the same port on **EGRESS_DESTINATION**.

If only some of the nodes in your cluster are capable of claiming the specified source IP address and using the specified gateway, you can specify a **nodeName** or **nodeSelector** indicating which nodes are acceptable.

8.5.2.2. Redirecting to Multiple Destinations

In the previous example, connections to the egress pod (or its corresponding service) on any port are redirected to a single destination IP. You can also configure different destination IPs depending on the port:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE
        value: 192.168.12.99
      - name: EGRESS_GATEWAY
        value: 192.168.12.1
      - name: EGRESS_DESTINATION
        value: | 1
                80    tcp 203.0.113.25
                8080  tcp 203.0.113.26 80
                8443  tcp 203.0.113.26 443
                203.0.113.27
      - name: EGRESS_ROUTER_MODE
        value: init
  containers:
  - name: egress-router-wait
    image: registry.access.redhat.com/openshift3/ose-pod
```

- 1 This uses the YAML syntax for a multi-line string; see below for details.

Each line of **EGRESS_DESTINATION** can be one of three types:

- **<port> <protocol> <IP address>** - This says that incoming connections to the given **<port>** should be redirected to the same port on the given **<IP address>**. **<protocol>** is either **tcp** or **udp**. In the example above, the first line redirects traffic from local port 80 to port 80 on 203.0.113.25.
- **<port> <protocol> <IP address> <remote port>** - As above, except that the connection is redirected to a different **<remote port>** on **<IP address>**. In the example above, the second and third lines redirect local ports 8080 and 8443 to remote ports 80 and 443 on 203.0.113.26.
- **<fallback IP address>** - If the last line of **EGRESS_DESTINATION** is a single IP address, then any connections on any other port will be redirected to the corresponding port on that IP address (eg, 203.0.113.27 in the example above). If there is no fallback IP address then connections on other ports would simply be rejected.)

8.5.2.3. Using a ConfigMap to specify EGRESS_DESTINATION

For a large or frequently-changing set of destination mappings, you can use a ConfigMap to externally maintain the list, and have the egress router pod read it from there. This comes with the advantage of project administrators being able to edit the ConfigMap, whereas they may not be able to edit the Pod definition directly, because it contains a privileged container.

1. Create a file containing the **EGRESS_DESTINATION** data:

```
$ cat my-egress-destination.txt
# Egress routes for Project "Test", version 3

80    tcp 203.0.113.25

8080  tcp 203.0.113.26 80
8443  tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

Note that you can put blank lines and comments into this file

2. Create a ConfigMap object from the file:

```
$ oc delete configmap egress-routes --ignore-not-found
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

Here **egress-routes** is the name of the ConfigMap object being created and **my-egress-destination.txt** is the name of the file the data is being read from.

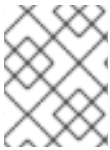
3. Create a egress router pod definition as above, but specifying the ConfigMap for **EGRESS_DESTINATION** in the environment section:

```
...
```

```

env:
- name: EGRESS_SOURCE
  value: 192.168.12.99
- name: EGRESS_GATEWAY
  value: 192.168.12.1
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
- name: EGRESS_ROUTER_MODE
  value: init
...

```

**NOTE**

The egress router does not automatically update when the ConfigMap changes. Restart the pod to get updates.

8.5.2.4. Deploying an Egress Router HTTP Proxy Pod

In *HTTP proxy mode*, the egress router runs as an HTTP proxy on port **8080**. This only works for clients talking to HTTP or HTTPS-based services, but usually requires fewer changes to the client pods to get them to work. Programs can be told to use an HTTP proxy by setting an environment variable.

1. Create the pod using the following as an example:

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-http-proxy
  labels:
    name: egress-http-proxy
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router-setup
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE ❷
    value: 192.168.12.99
  - name: EGRESS_GATEWAY ❸
    value: 192.168.12.1
  - name: EGRESS_ROUTER_MODE ❹
    value: http-proxy
  containers:
  - name: egress-router-proxy
    image: registry.access.redhat.com/openshift3/ose-egress-http-
proxy
    env:
    - name: EGRESS_HTTP_PROXY_DESTINATION ❺
      value: |

```

```
!*.example.com
!192.168.1.0/24
*
```

- 1 The **pod.network.openshift.io/assign-macvlan** annotation creates a Macvlan network interface on the primary network interface, then moves it into the pod's network name space before starting the **egress-router** container. Preserve the quotation marks around **"true"**. Omitting them results in errors.
- 2 An IP address from the physical network that the node itself is on and is reserved by the cluster administrator for use by this pod.
- 3 Same value as the default gateway used by the node itself.
- 4 This tells the egress router image that it is being deployed as part of an HTTP proxy, and so it should not set up iptables redirecting rules.
- 5 A string or YAML multi-line string specifying how to configure the proxy. Note that this is specified as an environment variable in the HTTP proxy container, not with the other environment variables in the init container.

You can specify any of the following for the **EGRESS_HTTP_PROXY_DESTINATION** value. You can also use *****, meaning "allow connections to all remote destinations". Each line in the configuration specifies one group of connections to allow or deny:

- An IP address (eg, **192.168.1.1**) allows connections to that IP address.
- A CIDR range (eg, **192.168.1.0/24**) allows connections to that CIDR range.
- A host name (eg, **www.example.com**) allows proxying to that host.
- A domain name preceded by *****. (eg, ***.example.com**) allows proxying to that domain and all of its subdomains.
- A **!** followed by any of the above denies connections rather than allowing them
- If the last line is *****, then anything that hasn't been denied will be allowed. Otherwise, anything that hasn't been allowed will be denied.

2. Ensure other pods can find the pod's IP address by creating a service to point to the egress router:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http-proxy
      port: 8080 1
  type: ClusterIP
selector:
  name: egress-1
```

- 1 Ensure the **http** port is always set to **8080**.

- Configure the client pod (not the egress proxy pod) to use the HTTP proxy by setting the `http_proxy` or `https_proxy` variables:

```
...
env:
- name: http_proxy
  value: http://egress-1:8080/ 1
- name: https_proxy
  value: http://egress-1:8080/
...
```

- The service created in step 2.



NOTE

Using the `http_proxy` and `https_proxy` environment variables is not necessary for all setups. If the above does not create a working setup, then consult the documentation for the tool or software you are running in the pod.

You can also specify the `EGRESS_HTTP_PROXY_DESTINATION` using a ConfigMap, similarly to [the redirecting egress router example above](#).

8.5.2.5. Enabling Failover for Egress Router Pods

Using a replication controller, you can ensure that there is always one copy of the egress router pod in order to prevent downtime.

- Create a replication controller configuration file using the following:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 1
  selector:
    name: egress-demo
  template:
    metadata:
      name: egress-demo
      labels:
        name: egress-demo
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec:
      initContainers:
        - name: egress-demo-init
          image: registry.access.redhat.com/openshift3/ose-egress-
router
      env:
        - name: EGRESS_SOURCE
          value: 192.168.12.99
        - name: EGRESS_GATEWAY
```



```

        value: 192.168.12.1
      - name: EGRESS_DESTINATION
        value: 203.0.113.25
      - name: EGRESS_ROUTER_MODE
        value: init
    securityContext:
      privileged: true
    containers:
      - name: egress-demo-wait
        image: registry.access.redhat.com/openshift3/ose-pod
    nodeSelector:
      site: springfield-1

```

1. Ensure **replicas** is set to **1**, because only one pod can be using a given **EGRESS_SOURCE** value at any time. This means that only a single copy of the router will be running, on a node with the label **site=springfield-1**.

2. Create the pod using the definition:

```
$ oc create -f <replication_controller>.json
```

3. To verify, check to see if the replication controller pod has been created:

```
$ oc describe rc <replication_controller>
```

8.5.3. Using iptables Rules to Limit Access to External Resources

Some cluster administrators may want to perform actions on outgoing traffic that do not fit within the model of **EgressNetworkPolicy** or the egress router. In some cases, this can be done by creating iptables rules directly.

For example, you could create rules that log traffic to particular destinations, or to prevent more than a certain number of outgoing connections per second.

OpenShift Container Platform does not provide a way to add custom iptables rules automatically, but it does provide a place where such rules can be added manually by the administrator. Each node, on startup, will create an empty chain called **OPENSIFT-ADMIN-OUTPUT-RULES** in the **filter** table (assuming that the chain does not already exist). Any rules added to that chain by an administrator will be applied to all traffic going from a pod to a destination outside the cluster (and not to any other traffic).

There are a few things to watch out for when using this functionality:

1. It is up to you to ensure that rules get created on each node; OpenShift Container Platform does not provide any way to make that happen automatically.
2. The rules are not applied to traffic that exits the cluster via an egress router, and they run after **EgressNetworkPolicy** rules are applied (and so will not see traffic that is denied by an **EgressNetworkPolicy**).
3. The handling of connections from pods to nodes or pods to the master is complicated, because nodes have both "external" IP addresses and "internal" SDN IP addresses. Thus, some pod-to-node/master traffic may pass through this chain, but other pod-to-node/master traffic may bypass it.

8.6. ENABLING STATIC IPS FOR EXTERNAL PROJECT TRAFFIC

As a cluster administrator, you can assign specific, static IP addresses to projects, so that traffic is externally easily recognizable. This is different from the default egress router, which is used to send traffic to specific destinations.

Recognizable IP traffic increases cluster security by ensuring the origin is visible. Once enabled, all outgoing external connections from the specified project will share the same, fixed source IP, meaning that any external resources can recognize the traffic.

Unlike the egress router, this is subject to **EgressNetworkPolicy** firewall rules.



IMPORTANT

Enabling static IPs for external project traffic 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/>.

To enable static source IPs:

1. Update the **NetNamespace** with the desired IP:

```
$ oc patch netnamespace <project_name> -p '{"egressIPs": [  
  <IP_address>"]}'
```

For example, to assign the **MyProject** project to an IP address of 192.168.1.100:

```
$ oc patch netnamespace MyProject -p '{"egressIPs":  
  ["192.168.1.100"]}'
```

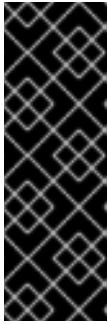
The **egressIPs** field is an array, but must be set to a single IP address. If setting multiple IPs, the other IPs will be ignored.

2. Manually assign the egress IP to the desired node hosts. Set the **egressIPs** field on the **HostSubnet** object on the node host. Include as many IPs as you want to assign to that node host:

```
$ oc patch hostsubnet <node_name> -p \  
  '{"egressIPs": ["<IP_address_1>", "<IP_address_2>"]}'
```

For example, to say that **node1** should have the egress IPs 192.168.1.100, 192.168.1.101, and 192.168.1.102:

```
$ oc patch hostsubnet node1 -p \  
  '{"egressIPs": ["192.168.1.100", "192.168.1.101",  
  "192.168.1.102"]}'
```



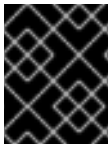
IMPORTANT

Egress IPs are implemented as additional IP addresses on the primary network interface, and must be in the same subnet as the node's primary IP. Additionally, any external IPs should not be configured in any Linux network configuration files, such as *ifcfg-eth0*.

Allowing additional IP addresses on the primary network interface might require extra configuration when using some cloud or VM solutions.

If the above is enabled for a project, all egress traffic from that project will be routed to the node hosting that egress IP, then connected (using NAT) to that IP address. If **egressIPs** is set on a **NetNamespace**, but there is no node hosting that egress IP, then egress traffic from the namespace will be dropped.

8.7. ENABLING MULTICAST



IMPORTANT

At this time, multicast is best used for low bandwidth coordination or service discovery and not a high-bandwidth solution.

Multicast traffic between OpenShift Container Platform pods is disabled by default. If you are using the **ovs-multitenant** or **ovs-networkpolicy** plugin, you can enable multicast on a per-project basis by setting an annotation on the project's corresponding **netnamespace** object:

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

Disable multicast by removing the annotation:

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled-
```

When using the **ovs-multitenant** plugin:

1. In an isolated project, multicast packets sent by a pod will be delivered to all other pods in the project.
2. If you have [joined networks together](#), you will need to enable multicast in each project's **netnamespace** in order for it to take effect in any of the projects. Multicast packets sent by a pod in a joined network will be delivered to all pods in all of the joined-together networks.
3. To enable multicast in the **default** project, you must also enable it in the **kube-service-catalog** project and all other projects that have been [made global](#). Global projects are not "global" for purposes of multicast; multicast packets sent by a pod in a global project will only be delivered to pods in other global projects, not to all pods in all projects. Likewise, pods in global projects will only receive multicast packets sent from pods in other global projects, not from all pods in all projects.

When using the **ovs-networkpolicy** plugin:

1. Multicast packets sent by a pod will be delivered to all other pods in the project, regardless of **NetworkPolicy** objects. (Pods may be able to communicate over multicast even when they can't communicate over unicast.)
2. Multicast packets sent by a pod in one project will never be delivered to pods in any other project, even if there are **NetworkPolicy** objects allowing communication between the two projects.

8.8. ENABLING NETWORKPOLICY

The **ovs-subnet** and **ovs-multitenant** plug-ins have their own legacy models of network isolation and do not support Kubernetes **NetworkPolicy**. However, **NetworkPolicy** support is available by using the **ovs-networkpolicy** plug-in.



NOTE

Only the **v1** NetworkPolicy features are available in OpenShift Container Platform. This means that egress policy types, IPBlock, and combining **podSelector** and **namespaceSelector** are not available in OpenShift Container Platform.

In a cluster [configured to use the ovs-networkpolicy plugin](#), network isolation is controlled entirely by **NetworkPolicy** objects. By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Project administrators can create and delete **NetworkPolicy** objects within their own project.

Pods that do not have **NetworkPolicy** objects pointing to them are fully accessible, whereas, pods that have one or more **NetworkPolicy** objects pointing to them are isolated. These isolated pods only accept connections that are accepted by at least one of their **NetworkPolicy** objects.

Following are a few sample **NetworkPolicy** object definitions supporting different scenarios:

- **Deny All Traffic**

To make a project "deny by default" add a **NetworkPolicy** object that matches all pods but accepts no traffic.

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

- **Only Accept connections from pods within project**

To make pods accept connections from other pods in the same project, but reject all other connections from pods in other projects:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-same-namespace
spec:
```

```

podSelector:
  ingress:
  - from:
    - podSelector: {}

```

- **Only allow HTTP and HTTPS traffic based on pod labels**

To enable only HTTP and HTTPS access to the pods with a specific label (**role=frontend** in following example), add a **NetworkPolicy** object similar to:

```

kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443

```

NetworkPolicy objects are additive, which means you can combine multiple **NetworkPolicy** objects together to satisfy complex network requirements.

For example, for the **NetworkPolicy** objects defined in previous samples, you can define both **allow-same-namespace** and **allow-http-and-https** policies within the same project. Thus allowing the pods with the label **role=frontend**, to accept any connection allowed by each policy. That is, connections on any port from pods in the **same** namespace, and connections on ports **80** and **443** from pods in **any** namespace.

8.8.1. NetworkPolicy and Routers

When using the **ovs-multitenant** plugin, traffic from the routers is automatically allowed into all namespaces. This is because the routers are usually in the *default* namespace, and all namespaces allow connections from pods in that namespace. With the **ovs-networkpolicy** plugin, this does not happen automatically. Therefore, if you have a policy that isolates a namespace by default, you need to take additional steps to allow routers to access it.

One option is to create a policy for each service, allowing access from all sources. for example,

```

kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-to-database-service
spec:
  podSelector:
    matchLabels:
      role: database
  ingress:

```

- ports:
 - protocol: TCP
 - port: 5432

This allows routers to access the service, but will also allow pods in other users' namespaces to access it as well. This should not cause any issues, as those pods can normally access the service by using the public router.

Alternatively, you can create a policy allowing full access from the default namespace, as in the **ovs-multitenant** plugin:

1. Add a label to the default namespace.



IMPORTANT

If you labeled the default project with the **default** label in a previous procedure, then skip this step. The cluster administrator role is required to add labels to namespaces.

```
$ oc label namespace default name=default
```

2. Create policies allowing connections from that namespace.



NOTE

Perform this step for each namespace you want to allow connections into. Users with the Project Administrator role can create policies.

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-from-default-namespace
spec:
  podSelector:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                name: default
```

8.8.2. Setting a Default NetworkPolicy for New Projects

The cluster administrators can modify the default project template to enable automatic creation of default **NetworkPolicy** objects (one or more), whenever a new project is created. To do this:

1. Create a custom project template and configure the master to use it, as described in [Modifying the Template for New Projects](#).
2. Label the **default** project with the **default** label:



IMPORTANT

If you labeled the default project with the **default** label in a previous procedure, then skip this step. The cluster administrator role is required to add labels to namespaces.

```
$ oc label namespace default name=default
```

3. Edit the template to include the desired **NetworkPolicy** objects:

```
$ oc edit template project-request -n default
```



NOTE

To include **NetworkPolicy** objects into existing template, use the **oc edit** command. Currently, it is not possible to use **oc patch** to add objects to a **Template** resource.

- a. Add each default policy as an element in the **objects** array:

```
objects:
...
- apiVersion: extensions/v1beta1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector:
      ingress:
        - from:
            - podSelector: {}
- apiVersion: extensions/v1beta1
  kind: NetworkPolicy
  metadata:
    name: allow-from-default-namespace
  spec:
    podSelector:
      ingress:
        - from:
            - namespaceSelector:
                matchLabels:
                  name: default
...

```

8.9. ENABLING HTTP STRICT TRANSPORT SECURITY

HTTP Strict Transport Security (HSTS) policy is a security enhancement, which ensures that only HTTPS traffic is allowed on the host. Any HTTP requests are dropped by default. This is useful for ensuring secure interactions with websites, or to offer a secure application for the user's benefit.

When HSTS is enabled, HSTS adds a Strict Transport Security header to HTTPS responses from the site. You can use the **insecureEdgeTerminationPolicy** value in a route to redirect to send HTTP

to HTTPS. However, when HSTS is enabled, the client changes all requests from the HTTP URL to HTTPS before the request is sent, eliminating the need for a redirect. This is not required to be supported by the client, and can be disabled by setting **max-age=0**.



IMPORTANT

HSTS works only with secure routes (either edge terminated or re-encrypt). The configuration is ineffective on HTTP or passthrough routes.

To enable HSTS to a route, add the **haproxy.router.openshift.io/hsts_header** value to the edge terminated or re-encrypt route:

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-
age=31536000;includeSubDomains;preload
```



IMPORTANT

Ensure there are no spaces and no other values in the parameters in the **haproxy.router.openshift.io/hsts_header** value. Only **max-age** is required.

The required **max-age** parameter indicates the length of time, in seconds, the HSTS policy is in effect for. The client updates **max-age** whenever a response with a HSTS header is received from the host. When **max-age** times out, the client discards the policy.

The optional **includeSubDomains** parameter tells the client that all subdomains of the host are to be treated the same as the host.

If **max-age** is greater than 0, the optional **preload** parameter allows external services to include this site in their HSTS preload lists. For example, sites such as Google can construct a list of sites that have **preload** set. Browsers can then use these lists to determine which sites to only talk to over HTTPS, even before they have interacted with the site. Without **preload** set, they need to have talked to the site over HTTPS to get the header.

8.10. TROUBLESHOOTING THROUGHPUT ISSUES

Sometimes applications deployed through OpenShift Container Platform can cause network throughput issues such as unusually high latency between specific services.

Use the following methods to analyze performance issues if pod logs do not reveal any cause of the problem:

- Use a packet analyzer, such as ping or [tcpdump](#) to analyze traffic between a pod and its node. For example, run the tcpdump tool on each pod while reproducing the behavior that led to the issue. Review the captures on both sides to compare send and receive timestamps to analyze the latency of traffic to/from a pod. Latency can occur in OpenShift Container Platform if a node interface is overloaded with traffic from other pods, storage devices, or the data plane.

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host
<podip 2> 1
```


■

- 1 **podip** is the IP address for the pod. Run the following command to get the IP address of the pods:

```
# oc get pod <podname> -o wide
```

tcpdump generates a file at **/tmp/dump.pcap** containing all traffic between these two pods. Ideally, run the analyzer shortly before the issue is reproduced and stop the analyzer shortly after the issue is finished reproducing to minimize the size of the file. You can also run a packet analyzer between the nodes (eliminating the SDN from the equation) with:

```
# tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- Use a bandwidth measuring tool, such as iperf, to measure streaming throughput and UDP throughput. Run the tool from the pods first, then from the nodes to attempt to locate any bottlenecks. The iperf3 tool is included as part of RHEL 7.

For information on installing and using iperf3, see this [Red Hat Solution](#).

CHAPTER 9. CONFIGURING SERVICE ACCOUNTS

9.1. OVERVIEW

When a person uses the OpenShift Container Platform CLI or web console, their API token authenticates them to the OpenShift Container Platform API. However, when a regular user's credentials are not available, it is common for components to make API calls independently. For example:

- Replication controllers make API calls to create or delete pods.
- Applications inside containers can make API calls for discovery purposes.
- External applications can make API calls for monitoring or integration purposes.

Service accounts provide a flexible way to control API access without sharing a regular user's credentials.

9.2. USER NAMES AND GROUPS

Every service account has an associated user name that can be granted roles, just like a regular user. The user name is derived from its project and name:

```
system:serviceaccount:<project>:<name>
```

For example, to add the **view** role to the **robot** service account in the **top-secret** project:

```
$ oc policy add-role-to-user view system:serviceaccount:top-secret:robot
```

IMPORTANT

If you want to grant access to a specific service account in a project, you can use the **-z** flag. From the project to which the service account belongs, use the **-z** flag and specify the **<serviceaccount_name>**. This is highly recommended, as it helps prevent typos and ensures that access is granted only to the specified service account. For example:

```
$ oc policy add-role-to-user <role_name> -z
<serviceaccount_name>
```

If not in the project, use the **-n** option to indicate the project namespace it applies to, as shown in the examples below.

Every service account is also a member of two groups:

system:serviceaccount

Includes all service accounts in the system.

system:serviceaccount:<project>

Includes all service accounts in the specified project.

For example, to allow all service accounts in all projects to view resources in the **top-secret** project:

```
$ oc policy add-role-to-group view system:serviceaccount -n top-secret
```

To allow all service accounts in the **managers** project to edit resources in the **top-secret** project:

```
$ oc policy add-role-to-group edit system:serviceaccount:managers -n top-secret
```

9.3. MANAGING SERVICE ACCOUNTS

Service accounts are API objects that exist within each project. To manage service accounts, you can use the **oc** command with the **sa** or **serviceaccount** object type or use the web console.

To get a list of existing service accounts in the current project:

```
$ oc get sa
NAME          SECRETS  AGE
builder       2        2d
default       2        2d
deployer      2        2d
```

To create a new service account:

```
$ oc create sa robot
serviceaccount "robot" created
```

As soon as a service account is created, two secrets are automatically added to it:

- an API token
- credentials for the OpenShift Container Registry

These can be seen by describing the service account:

```
$ oc describe sa robot
Name: robot
Namespace: project1
Labels: <none>
Annotations: <none>

Image pull secrets: robot-dockercfg-qzbhb

Mountable secrets: robot-token-f4khf
                  robot-dockercfg-qzbhb

Tokens:
    robot-token-f4khf
    robot-token-z8h44
```

The system ensures that service accounts always have an API token and registry credentials.

The generated API token and registry credentials do not expire, but they can be revoked by deleting the secret. When the secret is deleted, a new one is automatically generated to take its place.

9.4. ENABLING SERVICE ACCOUNT AUTHENTICATION

Service accounts authenticate to the API using tokens signed by a private RSA key. The authentication layer verifies the signature using a matching public RSA key.

To enable service account token generation, update the **serviceAccountConfig** stanza in the */etc/origin/master/master-config.yml* file on the master to specify a **privateKeyFile** (for signing), and a matching public key file in the **publicKeyFiles** list:

```
serviceAccountConfig:
  ...
  masterCA: ca.crt 1
  privateKeyFile: serviceaccount.private.key 2
  publicKeyFiles:
  - serviceaccount.public.key 3
  - ...
```

- 1 CA file used to validate the API server's serving certificate.
- 2 Private RSA key file (for token signing).
- 3 Public RSA key files (for token verification). If private key files are provided, then the public key component is used. Multiple public key files can be specified, and a token will be accepted if it can be validated by one of the public keys. This allows rotation of the signing key, while still accepting tokens generated by the previous signer.

9.5. MANAGED SERVICE ACCOUNTS

Service accounts are required in each project to run builds, deployments, and other pods. The **managedNames** setting in the */etc/origin/master/master-config.yml* file on the master controls which service accounts are automatically created in every project:

```
serviceAccountConfig:
  ...
  managedNames: 1
  - builder 2
  - deployer 3
  - default 4
  - ...
```

- 1 List of service accounts to automatically create in every project.
- 2 A **builder** service account in each project is required by build pods, and is given the **system:image-builder** role, which allows pushing images to any image stream in the project using the internal container registry.
- 3 A **deployer** service account in each project is required by deployment pods, and is given the **system:deployer** role, which allows viewing and modifying replication controllers and pods in the project.
- 4 A **default** service account is used by all other pods unless they specify a different service account.

All service accounts in a project are given the **system:image-puller** role, which allows pulling images from any image stream in the project using the internal container registry.

9.6. INFRASTRUCTURE SERVICE ACCOUNTS

Several infrastructure controllers run using service account credentials. The following service accounts are created in the OpenShift Container Platform infrastructure project (**openshift-infra**) at server start, and given the following roles cluster-wide:

Service Account	Description
replication-controller	Assigned the system:replication-controller role
deployment-controller	Assigned the system:deployment-controller role
build-controller	Assigned the system:build-controller role. Additionally, the build-controller service account is included in the privileged security context constraint in order to create privileged build pods.

To configure the project where those service accounts are created, set the **openshiftInfrastructureNamespace** field in the */etc/origin/master/master-config.yml* file on the master:

```
policyConfig:
  ...
  openshiftInfrastructureNamespace: openshift-infra
```

9.7. SERVICE ACCOUNTS AND SECRETS

Set the **limitSecretReferences** field in the */etc/origin/master/master-config.yml* file on the master to **true** to require pod secret references to be whitelisted by their service accounts. Set its value to **false** to allow pods to reference any secret in the project.

```
serviceAccountConfig:
  ...
  limitSecretReferences: false
```

CHAPTER 10. MANAGING ROLE-BASED ACCESS CONTROL (RBAC)

10.1. OVERVIEW

You can use [the CLI](#) to view [RBAC resources](#) and the administrator CLI to manage the [roles and bindings](#).

10.2. VIEWING ROLES AND BINDINGS

[Roles](#) can be used to grant various levels of access both [cluster-wide](#) as well as at the [project-scope](#). [Users and groups](#) can be associated with, or *bound* to, multiple roles at the same time. You can view details about the roles and their bindings using the **oc describe** command.

Users with the **cluster-admin** [default cluster role](#) bound cluster-wide can perform any action on any resource. Users with the **admin** [default cluster role](#) bound locally can manage roles and bindings in that project.



NOTE

Review a full list of verbs in the [Evaluating Authorization](#) section.

10.2.1. Viewing cluster roles

To view the cluster roles and their associated rule sets:

```
$ oc describe clusterrole.rbac
Name: admin
Labels: <none>
Annotations: openshift.io/description=A user that has edit rights within
the project and can change the project's membership.
rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources          Non-Resource URLs Resource Names Verbs
  -----
  appliedclusterresourcequotas      [] [] [get list watch]
  appliedclusterresourcequotas.quota.openshift.io [] [] [get list
watch]
  bindings          [] [] [get list watch]
  buildconfigs      [] [] [create delete deletecollection get list
patch update watch]
  buildconfigs.build.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  buildconfigs/instantiate [] [] [create]
  buildconfigs.build.openshift.io/instantiate [] [] [create]
  buildconfigs/instantiatebinary [] [] [create]
  buildconfigs.build.openshift.io/instantiatebinary [] [] [create]
  buildconfigs/webhooks [] [] [create delete deletecollection get
list patch update watch]
  buildconfigs.build.openshift.io/webhooks [] [] [create delete
deletecollection get list patch update watch]
  buildlogs [] [] [create delete deletecollection get list patch
update watch]
```

```

    buildlogs.build.openshift.io      [] [] [create delete
deletecollection get list patch update watch]
    builds                             [] [] [create delete deletecollection get list patch
update watch]
    builds.build.openshift.io          [] [] [create delete deletecollection
get list patch update watch]
    builds/clone                       [] [] [create]
    builds.build.openshift.io/clone    [] [] [create]
    builds/details                     [] [] [update]
    builds.build.openshift.io/details  [] [] [update]
    builds/log                         [] [] [get list watch]
    builds.build.openshift.io/log      [] [] [get list watch]
    configmaps                         [] [] [create delete deletecollection get list patch
update watch]
    cronjobs.batch                     [] [] [create delete deletecollection get list
patch update watch]
    daemonsets.extensions              [] [] [get list watch]
    deploymentconfigrollbacks          [] [] [create]
    deploymentconfigrollbacks.apps.openshift.io [] [] [create]
    deploymentconfigs                  [] [] [create delete deletecollection get list
patch update watch]
    deploymentconfigs.apps.openshift.io [] [] [create delete
deletecollection get list patch update watch]
    deploymentconfigs/instantiate       [] [] [create]
    deploymentconfigs.apps.openshift.io/instantiate [] [] [create]
    deploymentconfigs/log               [] [] [get list watch]
    deploymentconfigs.apps.openshift.io/log [] [] [get list watch]
    deploymentconfigs/rollback          [] [] [create]
    deploymentconfigs.apps.openshift.io/rollback [] [] [create]
    deploymentconfigs/scale             [] [] [create delete deletecollection get
list patch update watch]
    deploymentconfigs.apps.openshift.io/scale [] [] [create delete
deletecollection get list patch update watch]
    deploymentconfigs/status            [] [] [get list watch]
    deploymentconfigs.apps.openshift.io/status [] [] [get list watch]
    deployments.apps                   [] [] [create delete deletecollection get list
patch update watch]
    deployments.extensions              [] [] [create delete deletecollection get
list patch update watch]
    deployments.extensions/rollback     [] [] [create delete
deletecollection get list patch update watch]
    deployments.apps/scale              [] [] [create delete deletecollection get
list patch update watch]
    deployments.extensions/scale        [] [] [create delete
deletecollection get list patch update watch]
    deployments.apps/status             [] [] [create delete deletecollection get
list patch update watch]
    endpoints                          [] [] [create delete deletecollection get list patch
update watch]
    events                             [] [] [get list watch]
    horizontalpodautoscalers.autoscaling [] [] [create delete
deletecollection get list patch update watch]
    horizontalpodautoscalers.extensions [] [] [create delete
deletecollection get list patch update watch]
    imagestreamimages                  [] [] [create delete deletecollection get list
patch update watch]

```

```

    imagestreamimages.image.openshift.io    []    []    [create delete
deletecollection get list patch update watch]
    imagestreamimports    []    []    [create]
    imagestreamimports.image.openshift.io    []    []    [create]
    imagestreammappings    []    []    [create delete deletecollection get
list patch update watch]
    imagestreammappings.image.openshift.io    []    []    [create delete
deletecollection get list patch update watch]
    imagestreams    []    []    [create delete deletecollection get list
patch update watch]
    imagestreams.image.openshift.io    []    []    [create delete
deletecollection get list patch update watch]
    imagestreams/layers    []    []    [get update]
    imagestreams.image.openshift.io/layers    []    []    [get update]
    imagestreams/secrets    []    []    [create delete deletecollection get
list patch update watch]
    imagestreams.image.openshift.io/secrets    []    []    [create delete
deletecollection get list patch update watch]
    imagestreams/status    []    []    [get list watch]
    imagestreams.image.openshift.io/status    []    []    [get list watch]
    imagestreamtags    []    []    [create delete deletecollection get list
patch update watch]
    imagestreamtags.image.openshift.io    []    []    [create delete
deletecollection get list patch update watch]
    jenkins.build.openshift.io    []    []    [admin edit view]
    jobs.batch    []    []    [create delete deletecollection get list patch
update watch]
    limitranges    []    []    [get list watch]
    localresourceaccessreviews    []    []    [create]
    localresourceaccessreviews.authorization.openshift.io    []    []    [create]
    localsubjectaccessreviews    []    []    [create]
    localsubjectaccessreviews.authorization.k8s.io    []    []    [create]
    localsubjectaccessreviews.authorization.openshift.io    []    []    [create]
    namespaces    []    []    [get list watch]
    namespaces/status    []    []    [get list watch]
    networkpolicies.extensions    []    []    [create delete deletecollection
get list patch update watch]
    persistentvolumeclaims    []    []    [create delete deletecollection get
list patch update watch]
    pods    []    []    [create delete deletecollection get list patch
update watch]
    pods/attach    []    []    [create delete deletecollection get list
patch update watch]
    pods/exec    []    []    [create delete deletecollection get list patch
update watch]
    pods/log    []    []    [get list watch]
    pods/portforward    []    []    [create delete deletecollection get list
patch update watch]
    pods/proxy    []    []    [create delete deletecollection get list patch
update watch]
    pods/status    []    []    [get list watch]
    podsecuritypolicyreviews    []    []    [create]
    podsecuritypolicyreviews.security.openshift.io    []    []    [create]
    podsecuritypolicyselfsubjectreviews    []    []    [create]
    podsecuritypolicyselfsubjectreviews.security.openshift.io    []    []
[create]

```



```

    podsecuritypolicyreviews [] [] [create]
    podsecuritypolicyreviews.security.openshift.io [] [] [create]
    processedtemplates [] [] [create delete deletecollection get
list patch update watch]
    processedtemplates.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
    projects [] [] [delete get patch update]
    projects.project.openshift.io [] [] [delete get patch update]
    replicaset.extensions [] [] [create delete deletecollection get
list patch update watch]
    replicaset.extensions/scale [] [] [create delete
deletecollection get list patch update watch]
    replicationcontrollers [] [] [create delete deletecollection get
list patch update watch]
    replicationcontrollers/scale [] [] [create delete
deletecollection get list patch update watch]
    replicationcontrollers.extensions/scale [] [] [create delete
deletecollection get list patch update watch]
    replicationcontrollers/status [] [] [get list watch]
    resourceaccessreviews [] [] [create]
    resourceaccessreviews.authorization.openshift.io [] [] [create]
    resourcequotas [] [] [get list watch]
    resourcequotas/status [] [] [get list watch]
    resourcequotausages [] [] [get list watch]
    rolebindingrestrictions [] [] [get list watch]
    rolebindingrestrictions.authorization.openshift.io [] [] [get list
watch]
    rolebindings [] [] [create delete deletecollection get list
patch update watch]
    rolebindings.authorization.openshift.io [] [] [create delete
deletecollection get list patch update watch]
    rolebindings.rbac.authorization.k8s.io [] [] [create delete
deletecollection get list patch update watch]
    roles [] [] [create delete deletecollection get list patch
update watch]
    roles.authorization.openshift.io [] [] [create delete
deletecollection get list patch update watch]
    roles.rbac.authorization.k8s.io [] [] [create delete
deletecollection get list patch update watch]
    routes [] [] [create delete deletecollection get list patch
update watch]
    routes.route.openshift.io [] [] [create delete deletecollection
get list patch update watch]
    routes/custom-host [] [] [create]
    routes.route.openshift.io/custom-host [] [] [create]
    routes/status [] [] [get list watch update]
    routes.route.openshift.io/status [] [] [get list watch update]
    scheduledjobs.batch [] [] [create delete deletecollection get
list patch update watch]
    secrets [] [] [create delete deletecollection get list patch
update watch]
    serviceaccounts [] [] [create delete deletecollection get list
patch update watch impersonate]
    services [] [] [create delete deletecollection get list patch
update watch]
    services/proxy [] [] [create delete deletecollection get list

```

```

patch update watch]
  statefulsets.apps      [] [] [create delete deletecollection get list
patch update watch]
  subjectaccessreviews   [] [] [create]
  subjectaccessreviews.authorization.openshift.io [] [] [create]
  subjectrulesreviews    [] [] [create]
  subjectrulesreviews.authorization.openshift.io [] [] [create]
  templateconfigs        [] [] [create delete deletecollection get list
patch update watch]
  templateconfigs.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  templateinstances      [] [] [create delete deletecollection get list
patch update watch]
  templateinstances.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  templates              [] [] [create delete deletecollection get list patch
update watch]
  templates.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]

```

Name: basic-user

Labels: <none>

Annotations: openshift.io/description=A user that can get basic information about projects.

rbac.authorization.kubernetes.io/autoupdate=true

PolicyRule:

Resources	Non-Resource URLs	Resource Names	Verbs
clusterroles			[get list]
clusterroles.authorization.openshift.io			[get list]
clusterroles.rbac.authorization.k8s.io			[get list watch]
projectrequests			[list]
projectrequests.project.openshift.io			[list]
projects			[list watch]
projects.project.openshift.io			[list watch]
selfsubjectaccessreviews.authorization.k8s.io			[create]
selfsubjectrulesreviews			[create]
selfsubjectrulesreviews.authorization.openshift.io			[create]
storageclasses.storage.k8s.io			[get list]
users			[get]
users.user.openshift.io			[get]

Name: cluster-admin

Labels: <none>

Annotations: authorization.openshift.io/system-only=true

openshift.io/description=A super-user that can perform any action in the cluster. When granted to a user within a project, they have full control over quota and membership and can perform every action...

rbac.authorization.kubernetes.io/autoupdate=true

PolicyRule:

Resources	Non-Resource URLs	Resource Names	Verbs
[*]			[*]
.*			[*]

```

Name: cluster-debugger
Labels: <none>
Annotations: authorization.openshift.io/system-only=true
             rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  -----
  [/debug/pprof] [] [get]
  [/debug/pprof/*] [] [get]
  [/metrics] [] [get]

Name: cluster-reader
Labels: <none>
Annotations: authorization.openshift.io/system-only=true
             rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  -----
  [*] [] [get]
  apiservices.apiregistration.k8s.io [] [] [get list watch]
  apiservices.apiregistration.k8s.io/status [] [] [get list watch]
  appliedclusterresourcequotas [] [] [get list watch]

...

```

10.2.2. Viewing cluster role bindings

To view the current set of cluster role bindings, which show the users and groups that are bound to various roles:

```

$ oc describe clusterrolebinding.rbac
Name: admin
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind  Name  Namespace
  ----  -
  ServiceAccount template-instance-controller openshift-infra

Name: basic-users
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind Name  Namespace
  ---- -

```

Group system:authenticated

```
Name: cluster-admin
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind  Name  Namespace
  ----  -
  ServiceAccount pinstaller default
  Group system:masters
```

```
Name: cluster-admins
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind Name  Namespace
  ---- -
  Group system:cluster-admins
  User system:admin
```

```
Name: cluster-readers
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-reader
Subjects:
  Kind Name  Namespace
  ---- -
  Group system:cluster-readers
```

```
Name: cluster-status-binding
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: cluster-status
Subjects:
  Kind Name  Namespace
  ---- -
  Group system:authenticated
  Group system:unauthenticated
```

```
Name: registry-registry-role
Labels: <none>
```

```

Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:registry
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount registry default

Name: router-router-role
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:router
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount router default

Name: self-access-reviewers
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-access-reviewer
Subjects:
  Kind Name    Namespace
  ---- -
  Group system:authenticated
  Group system:unauthenticated

Name: self-provisioners
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name    Namespace
  ---- -
  Group system:authenticated:oauth

Name: system:basic-user
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:basic-user
Subjects:
  Kind Name    Namespace
  ---- -

```

```

Group system:authenticated
Group system:unauthenticated

```

```

Name: system:build-strategy-docker-binding
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-docker
Subjects:
  Kind Name      Namespace
  ---- ----      -
  Group system:authenticated

```

```

Name: system:build-strategy-jenkinspipeline-binding
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-jenkinspipeline
Subjects:
  Kind Name      Namespace
  ---- ----      -
  Group system:authenticated

```

```

Name: system:build-strategy-source-binding
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:build-strategy-source
Subjects:
  Kind Name      Namespace
  ---- ----      -
  Group system:authenticated

```

```

Name: system:controller:attachdetach-controller
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:controller:attachdetach-controller
Subjects:
  Kind Name      Namespace
  ---- ----      -
  ServiceAccount attachdetach-controller kube-system

```

```

Name: system:controller:certificate-controller
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:

```

```

Kind: ClusterRole
Name: system:controller:certificate-controller
Subjects:
  Kind      Name      Namespace
  ----      -
  ServiceAccount certificate-controller kube-system

Name: system:controller:cronjob-controller
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true

...

```

10.2.3. Viewing local roles and bindings

All of the [default cluster roles](#) can be bound locally to users or groups.

[Custom local roles](#) can be created.

The local role bindings are also viewable.

To view the current set of local role bindings, which show the users and groups that are bound to various roles:

```
$ oc describe rolebinding.rbac
```

By default, the current project is used when viewing local role bindings. Alternatively, a project can be specified with the `-n` flag. This is useful for viewing the local role bindings of another project, if the user already has the [admin default cluster role](#) in it.

```

$ oc describe rolebinding.rbac -n joe-project
Name: admin
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name Namespace
  ---- -
  User joe

Name: system:deployers
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind Name Namespace
  ---- -
  ServiceAccount deployer joe-project

```

```

Name:  system:image-builders
Labels:  <none>
Annotations:  <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount builder joe-project

Name:  system:image-pullers
Labels:  <none>
Annotations:  <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind Name    Namespace
  ---- ----    -
  Group system:serviceaccounts:joe-project

```

10.3. MANAGING ROLE BINDINGS

Adding, or *binding*, a [role](#) to [users](#) or [groups](#) gives the user or group the relevant access granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

When managing a user or group's associated roles for local role bindings using the following operations, a project may be specified with the **-n** flag. If it is not specified, then the current project is used.

Table 10.1. Local role binding operations

Command	Description
\$ oc adm policy who-can <verb> <resource>	Indicates which users can perform an action on a resource.
\$ oc adm policy add-role-to-user <role> <username>	Binds a given role to specified users in the current project.
\$ oc adm policy remove-role-from-user <role> <username>	Removes a given role from specified users in the current project.
\$ oc adm policy remove-user <username>	Removes specified users and all of their roles in the current project.
\$ oc adm policy add-role-to-group <role> <groupname>	Binds a given role to specified groups in the current project.

Command	Description
<code>\$ oc adm policy remove-role-from-group <role> <groupname></code>	Removes a given role from specified groups in the current project.
<code>\$ oc adm policy remove-group <groupname></code>	Removes specified groups and all of their roles in the current project.

You can also manage cluster role bindings using the following operations. The **-n** flag is not used for these operations because cluster role bindings use non-namespaced resources.

Table 10.2. Cluster role binding operations

Command	Description
<code>\$ oc adm policy add-cluster-role-to-user <role> <username></code>	Binds a given role to specified users for all projects in the cluster.
<code>\$ oc adm policy remove-cluster-role-from-user <role> <username></code>	Removes a given role from specified users for all projects in the cluster.
<code>\$ oc adm policy add-cluster-role-to-group <role> <groupname></code>	Binds a given role to specified groups for all projects in the cluster.
<code>\$ oc adm policy remove-cluster-role-from-group <role> <groupname></code>	Removes a given role from specified groups for all projects in the cluster.

For example, you can add the **admin** role to the **alice** user in **joe-project** by running:

```
$ oc adm policy add-role-to-user admin alice -n joe-project
```

You can then view the local role bindings and verify the addition in the output:

```
$ oc describe rolebinding.rbac -n joe-project
Name: admin
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name Namespace
  ----
  User joe
  User alice 1
```

```
Name: system:deployers
Labels: <none>
Annotations: <none>
Role:
```

```

Kind: ClusterRole
Name: system:deployer
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount deployer joe-project

Name: system:image-builders
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount builder joe-project

Name: system:image-pullers
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind    Name    Namespace
  ----    -
  Group system:serviceaccounts:joe-project

```

- 1 The **alice** user has been added to the **admins RoleBinding**.

10.4. GRANTING USERS DAEMONSET PERMISSIONS

By default, project developers do not have the permission to create [daemonsets](#). As a cluster administrator, you can grant them the abilities.

1. Create the cluster role:

```
$ oc create clusterrole daemonset-admin --
verb=create,delete,get,list,update,watch,patch --
resource=daemonsets.extensions
```

2. Create the local role binding:

```
$ oc adm policy add-role-to-user daemonset-admin <user>
```

10.5. CREATING A LOCAL ROLE

You can create a local role for a project and then bind it to a user.

1. To create a local role for a project, run the following command:

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n
<project>
```

In this command, specify: * **<name>**, the local role's name * **<verb>**, a comma-separated list of the verbs to apply to the role * **<resource>**, the resources that the role applies to * **<project>**, the project name

+ For example, to create a local role that allows a user to view pods in the **blue** project, run the following command:

+

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. To bind the new role to a user, run the following command:

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n
blue
```

10.6. CREATING A CLUSTER ROLE

To create a cluster role, run the following command:

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

In this command, specify:

- **<name>**, the local role's name
- **<verb>**, a comma-separated list of the verbs to apply to the role
- **<resource>**, the resources that the role applies to

For example, to create a cluster role that allows a user to view pods, run the following command:

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

10.7. CLUSTER AND LOCAL ROLE BINDINGS

A cluster role binding is a binding that exists at the cluster level. A role binding exists at the project level. The cluster role *view* must be bound to a user using a local role binding for that user to view the project. Create local roles only if a cluster role does not provide the set of permissions needed for a particular situation.

Some cluster role names are initially confusing. You can bind the **cluster-admin** to a user, using a local role binding, making it appear that this user has the privileges of a cluster administrator. This is not the case. Binding the **cluster-admin** to a certain project is more like a super administrator for that project, granting the permissions of the cluster role **admin**, plus a few additional permissions like the ability to edit rate limits. This can appear confusing especially via the web console UI, which does not list cluster role bindings that are bound to true cluster administrators. However, it does list local role bindings that you can use to locally bind **cluster-admin**.

CHAPTER 11. IMAGE POLICY

11.1. OVERVIEW

You can control which images are allowed to run on your cluster using the ImagePolicy admission plug-in (currently considered beta). It allows you to control:

- **The source of images:** which registries can be used to pull images
- **Image resolution:** force pods to run with immutable digests to ensure the image does not change due to a re-tag
- **Container image label restrictions:** force an image to have or not have particular labels
- **Image annotation restrictions:** force an image in the integrated container registry to have or not have particular annotations

11.2. CONFIGURING THE IMAGEPOLICY ADMISSION PLUG-IN

To configure which images can run on your cluster, configure the ImagePolicy Admission plug-in in the *master-config.yaml* file. You can set one or more rules as required.

- **Reject images with a particular annotation:**

Use this rule to reject all images that have a specific annotation set on them. The following rejects all images using the `images.openshift.io/deny-execution` annotation:

```
- name: execution-denied
  onResources:
    - resource: pods
    - resource: builds
  reject: true
  matchImageAnnotations:
    - key: images.openshift.io/deny-execution 1
      value: "true"
  skipOnResolutionFailure: true
```

- 1 If a particular image has been deemed harmful, administrators can set this annotation to flag those images.

- **Enable user to run images from Docker Hub:**

Use this rule to allow users to use images from Docker Hub:

```
- name: allow-images-from-dockerhub
  onResources:
    - resource: pods
    - resource: builds
  matchRegistries:
    - docker.io
```

Following is an example configuration for setting multiple ImagePolicy admission plugin rules in the *master-config.yaml* file:

Annotated Example File

```

admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
        kind: ImagePolicyConfig
        apiVersion: v1
        resolveImages: AttemptRewrite ❶
        executionRules: ❷
        - name: execution-denied
          # Reject all images that have the annotation
          images.openshift.io/deny-execution set to true.
          # This annotation may be set by infrastructure that wishes to
          flag particular images as dangerous
        onResources: ❸
        - resource: pods
        - resource: builds
        reject: true ❹
        matchImageAnnotations: ❺
        - key: images.openshift.io/deny-execution
          value: "true"
        skipOnResolutionFailure: true ❻
        - name: allow-images-from-internal-registry
          # allows images from the internal registry and tries to resolve
          them
          onResources:
            - resource: pods
            - resource: builds
          matchIntegratedRegistry: true
        - name: allow-images-from-dockerhub
          onResources:
            - resource: pods
            - resource: builds
          matchRegistries:
            - docker.io
        resolutionRules: ❼
        - targetResource:
            resource: pods
            localNames: true
            policy: AttemptRewrite
        - targetResource: ❽
            group: batch
            resource: jobs
            localNames: true ❾
            policy: AttemptRewrite

```

❶ Try to resolve images to an immutable image digest and update the image pull specification in the pod.

❷ Array of rules to evaluate against incoming resources. If you only have **reject: true** rules, the default is **allow all**. If you have any accept rule, that is **reject: false** in any of the rules, the default behaviour of the ImagePolicy switches to **deny-all**.

- 3 Indicates which resources to enforce rules upon. If nothing is specified, the default is **pods**.
- 4 Indicates that if this rule matches, the pod should be rejected.
- 5 List of annotations to match on the image object's metadata.
- 6 If you are not able to resolve the image, do not fail the pod.
- 7 Array of rules allowing use of image streams in Kubernetes resources. The default configuration allows pods, replicationcontrollers, replicaset, statefulsets, daemonsets, deployments, and jobs to use same-project image stream tag references in their image fields.
- 8 Identifies the group and resource to which this rule applies. If resource is *****, this rule will apply to all resources in that group.
- 9 **LocalNames** will allow single segment names (for example, **ruby:2.4**) to be interpreted as namespace-local image stream tags, but only if the resource or target image stream has **local name resolution** enabled.



NOTE

If you normally rely on infrastructure images being pulled using a default registry prefix (such as **docker.io** or **registry.access.redhat.com**), those images will not match to any **matchRegistries** value since they will have no registry prefix. To ensure infrastructure images have a registry prefix that can match your image policy, set the **imageConfig.format** value in your **master-config.yaml** file.

11.3. TESTING THE IMAGEPOLICY ADMISSION PLUG-IN

1. Use the **openshift/image-policy-check** to test your configuration.
For example, use the information above, then test like this:

```
oc import-image openshift/image-policy-check:latest --confirm
```

2. Create a pod using this YAML. The pod should be created.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: docker.io/openshift/image-policy-check:latest
    name: first
```

3. Create another pod pointing to a different registry. The pod should be rejected.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
```

```
containers:
- image: different-registry/openshift/image-policy-check:latest
  name: first
```

4. Create a pod pointing to the internal registry using the imported image. The pod should be created and if you look at the image specification, you should see a digest in place of the tag.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:latest
    name: first
```

5. Create a pod pointing to the internal registry using the imported image. The pod should be created and if you look at the image specification, you should see the tag unmodified.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:v1
    name: first
```

6. Get the digest from **oc get istag/image-policy-check:latest** and use it for **oc annotate images/<digest> images.openshift.io/deny-execution=true**. For example:

```
$ oc annotate
images/sha256:09ce3d8b5b63595ffca6636c7daefb1a615a7c0e3f8ea68e5db044
a9340d6ba8 images.openshift.io/deny-execution=true
```

7. Create this pod again, and you should see the pod rejected:

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-check:latest
    name: first
```

CHAPTER 12. IMAGE SIGNATURES

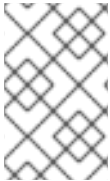
12.1. OVERVIEW

Container image signing on Red Hat Enterprise Linux (RHEL) systems provides a means of:

- Validating where a container image came from,
- Checking that the image has not been tampered with, and
- Setting policies to determine which validated images can be pulled to a host.

For a more complete understanding of the architecture of container image signing on RHEL systems, see the [Container Image Signing Integration Guide](#).

The OpenShift Container Registry allows the ability to store signatures via REST API. The **oc** CLI can be used to verify image signatures, with their validated displayed in the web console or CLI.



NOTE

Initial support for storing image signatures was added in OpenShift Container Platform 3.3. Initial support for verifying image signatures was added in OpenShift Container Platform 3.6.

12.2. SIGNING IMAGES USING ATOMIC CLI

OpenShift Container Platform does not automate image signing. Signing requires a developer's private GPG key, typically stored securely on a workstation. This document describes that workflow.

The **atomic** command line interface (CLI), version 1.12.5 or greater, provides commands for signing container images, which can be pushed to an OpenShift Container Registry. The **atomic** CLI is available on Red Hat-based distributions: RHEL, Centos, and Fedora. The **atomic** CLI is pre-installed on RHEL Atomic Host systems. For information on installing the **atomic** package on a RHEL host, see [Enabling Image Signature Support](#).



IMPORTANT

The **atomic** CLI uses the authenticated credentials from **oc login**. Be sure to use the same user on the same host for both **atomic** and **oc** commands. For example, if you execute **atomic** CLI as **sudo**, be sure to log in to OpenShift Container Platform using **sudo oc login**.

In order to attach the signature to the image, the user must have the **image-signer** cluster role. Cluster administrators can add this using:

```
$ oc adm policy add-cluster-role-to-user system:image-signer <user_name>
```

Images may be signed at push time:

```
$ atomic push [--sign-by <gpg_key_id>] --type atomic <image>
```


Signatures are stored in OpenShift Container Platform when the **atomic** transport type argument is specified. See [Signature Transports](#) for more information.

For full details on how to set up and perform image signing using the **atomic** CLI, see the [RHEL Atomic Host Managing Containers: Signing Container Images](#) documentation or the **atomic push --help** output for argument details.

A specific example workflow of working with the **atomic** CLI and an OpenShift Container Registry is documented in the [Container Image Signing Integration Guide](#).

12.3. VERIFYING IMAGE SIGNATURES USING OPENSIFT CLI

You can verify the signatures of an image imported to an OpenShift Container Registry using the **oc adm verify-image-signature** command. This command verifies if the image identity contained in the image signature can be trusted by using the public GPG key to verify the signature itself then match the provided expected identity with the identity (the pull spec) of the given image.

By default, this command uses the public GPG keyring located in **\$GNUPGHOME/pubring.gpg**, typically in path **~/.gnupg**. By default, this command does not save the result of the verification back to the image object. To do so, you must specify the **--save** flag, as shown below.



NOTE

In order to verify the signature of an image, the user must have the **image-auditor** cluster role. Cluster administrators can add this using:

```
$ oc adm policy add-cluster-role-to-user system:image-auditor
<user_name>
```



IMPORTANT

Using the **--save** flag on already verified image together with invalid GPG key or invalid expected identity causes the saved verification status and all signatures to be removed, and the image will become unverified.

In order to avoid deleting all signatures by mistake, you can run the command without the **--save** flag first and check the logs for potential issues.

To verify an image signature use the following format:

```
$ oc adm verify-image-signature <image> --expected-identity=<pull_spec> [-
-save] [options]
```

The **<pull_spec>** can be found by describing the image stream. The **<image>** may be found by describing the image stream tag. See the following example command output.

Example Image Signature Verification

```
$ oc describe is nodejs -n openshift
Name:                nodejs
Namespace:           openshift
Created:             2 weeks ago
Labels:              <none>
```

```

Annotations:      openshift.io/display-name=Node.js
                  openshift.io/image.dockerRepositoryCheck=2017-07-
05T18:24:01Z
Docker Pull Spec: 172.30.1.1:5000/openshift/nodejs
...

$ oc describe istag nodejs:latest -n openshift
Image Name:
sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288
...

$ oc adm verify-image-signature \

sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 \
--expected-identity 172.30.1.1:5000/openshift/nodejs:latest \
--public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release \
--save

```

12.4. ACCESSING IMAGE SIGNATURES USING REGISTRY API

The OpenShift Container Registry provides an **extensions** endpoint that allows you to write and read image signatures. The image signatures are stored in the OpenShift Container Platform key-value store via the Docker Registry API.



NOTE

This endpoint is experimental and not supported by the upstream Docker Registry project. See the [upstream API documentation](#) for general information about the Docker Registry API.

12.4.1. Writing Image Signatures via API

In order to add a new signature to the image, you can use the HTTP **PUT** method to send a JSON payload to the **extensions** endpoint:

```

PUT /extensions/v2/<namespace>/<name>/signatures/<digest>

$ curl -X PUT --data @signature.json http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatur
es/sha256:<digest>

```

The JSON payload with the signature content should have the following structure:

```

{
  "version": 2,
  "type":    "atomic",
  "name":
"sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@c
ddeb7006d914716e2728000746a0b23",
  "content": "<cryptographic_signature>"
}

```

The **name** field contains the name of the image signature, which must be unique and in the format

<digest>@<name>. The **<digest>** represents an image name and the **<name>** is the name of the signature. The signature name must be 32 characters long. The **<cryptographic_signature>** must follow the specification documented in the [containers/image](#) library.

12.4.2. Reading Image Signatures via API

Assuming a signed image has already been pushed into the OpenShift Container Registry, you can read the signatures using the following command:

```
GET /extensions/v2/<namespace>/<name>/signatures/<digest>
```

```
$ curl http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatur
es/sha256:<digest>
```

The **<namespace>** represents the OpenShift Container Platform project name or registry repository name and the **<name>** refers to the name of the image repository. The **digest** represents the SHA-256 checksum of the image.

If the given image contains the signature data, the output of the command above should produce following JSON response:

```
{
  "signatures": [
    {
      "version": 2,
      "type": "atomic",
      "name":
"sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@c
ddeb7006d914716e2728000746a0b23",
      "content": "<cryptographic_signature>"
    }
  ]
}
```

The **name** field contains the name of the image signature, which must be unique and in the format **<digest>@<name>**. The **<digest>** represents an image name and the **<name>** is the name of the signature. The signature name must be 32 characters long. The **<cryptographic_signature>** must follow the specification documented in the [containers/image](#) library.

12.4.3. Importing Image Signatures Automatically from Signature Stores

OpenShift Container Platform can automatically import image signatures if a signature store is configured on all OpenShift Container Platform master nodes through the registries configuration directory.

The registries configuration directory contains the configuration for various registries (servers storing remote container images) and for the content stored in them. The single directory ensures that the configuration does not have to be provided in command-line options for each command, so that it can be shared by all the users of the [containers/image](#).

The default registries configuration directory is located in the */etc/containers/registries.d/default.yaml* file.

A sample configuration that will cause image signatures to be imported automatically for all Red Hat images:

```
docker:
  registry.access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

1

1 Defines the URL of a signature store. This URL is used for reading existing signatures.



NOTE

Signatures imported automatically by OpenShift Container Platform will be *unverified* by default and will have to be verified by image administrators.

For more details about the registries configuration directory, see [Registries Configuration Directory](#).

CHAPTER 13. SCOPED TOKENS

13.1. OVERVIEW

A user may want to give another entity the power to act as they have, but only in a limited way. For example, a project administrator may want to delegate the power to create pods. One way to do this is to create a scoped token.

A scoped token is a token that identifies as a given user, but is limited to certain actions by its scope. Right now, only a **cluster-admin** can create scoped tokens.

13.2. EVALUATION

Scopes are evaluated by converting the set of scopes for a token into a set of **PolicyRules**. Then, the request is matched against those rules. The request attributes must match at least one of the scope rules to be passed to the "normal" authorizer for further authorization checks.

13.3. USER SCOPES

User scopes are focused on getting information about a given user. They are intent-based, so the rules are automatically created for you:

- **user:full** - Allows full read/write access to the API with all of the user's permissions.
- **user:info** - Allows read-only access to information about the user: name, groups, and so on.
- **user:check-access** - Allows access to **self-localsubjectaccessreviews** and **self-subjectaccessreviews**. These are the variables where you pass an empty user and groups in your request object.
- **user:list-projects** - Allows read-only access to list the projects the user has access to.

13.4. ROLE SCOPE

The role scope allows you to have the same level of access as a given role filtered by namespace.

- **role:<cluster-role name>:<namespace or * for all>** - Limits the scope to the rules specified by the cluster-role, but only in the specified namespace .



NOTE

Caveat: This prevents escalating access. Even if the role allows access to resources like secrets, rolebindings, and roles, this scope will deny access to those resources. This helps prevent unexpected escalations. Many people do not think of a role like **edit** as being an escalating role, but with access to a secret it is.

- **role:<cluster-role name>:<namespace or * for all>:!** - This is similar to the example above, except that including the bang causes this scope to allow escalating access.

CHAPTER 14. MONITORING IMAGES

14.1. OVERVIEW

You can monitor [images](#) and [nodes](#) in your instance using the [CLI](#).

14.2. VIEWING IMAGES STATISTICS

You can display usage statistics about all of the images that OpenShift Container Platform manages. In other words, all the images pushed to the internal registry either [directly](#) or through a [build](#).

To view the usage statistics:

```
$ oc adm top images
NAME                                IMAGESTREAMTAG      PARENTS
USAGE                                METADATA    STORAGE
sha256:80c985739a78b openshift/python (3.5)
yes                                303.12MiB
sha256:64461b5111fc7 openshift/ruby (2.2)
yes                                234.33MiB
sha256:0e19a0290ddc1 test/ruby-ex (latest)      sha256:64461b5111fc71ec
Deployment: ruby-ex-1/test      yes            150.65MiB
sha256:a968c61adad58 test/django-ex (latest)  sha256:80c985739a78b760
Deployment: django-ex-1/test    yes            186.07MiB
```

The command displays the following information:

- Image ID
- Project, name, and tag of the accompanying **ImageStreamTag**
- Potential parents of the image, listed by their IDs
- Information about where the image is used
- Flag informing whether the image contains proper Docker metadata information
- Size of the image

14.3. VIEWING IMAGESTREAMS STATISTICS

You can display usage statistics about **ImageStreams**.

To view the usage statistics:

```
$ oc adm top imagestreams
NAME                                STORAGE    IMAGES    LAYERS
openshift/python                    1.21GiB    4         36
openshift/ruby                      717.76MiB  3         27
test/ruby-ex                        150.65MiB  1         10
test/django-ex                      186.07MiB  1         10
```

The command displays the following information:

- Project and name of the **ImageStream**
- Size of the entire **ImageStream** stored in the internal [Red Hat Container Registry](#)
- Number of images this particular **ImageStream** is pointing to
- Number of layers **ImageStream** consists of

14.4. PRUNING IMAGES

The information returned from the previous commands is helpful when performing image pruning.

CHAPTER 15. MANAGING SECURITY CONTEXT CONSTRAINTS

15.1. OVERVIEW

Security context constraints allow administrators to control permissions for pods. To learn more about this API type, see the [security context constraints \(SCCs\)](#) architecture documentation. You can manage SCCs in your instance as normal API [objects](#) using [the CLI](#).



NOTE

You must have [cluster-admin privileges](#) to manage SCCs.



IMPORTANT

Do not modify the default SCCs. Customizing the default SCCs can lead to issues when upgrading. Instead, [create new SCCs](#).

15.2. LISTING SECURITY CONTEXT CONSTRAINTS

To get a current list of SCCs:

```
$ oc get scc
```

NAME	PRIV	CAPS	SELINUX	RUNASUSER
FSGROUP	SUPGROUP	PRIORITY	READONLYROOTFS	VOLUMES
anyuid	false	[]	MustRunAs	RunAsAny
RunAsAny	RunAsAny	10	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
hostaccess	false	[]	MustRunAs	MustRunAsRange
MustRunAs	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	hostPath	persistentVolumeClaim	secret]
hostmount-anyuid	false	[]	MustRunAs	RunAsAny
RunAsAny	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	hostPath	nfs	persistentVolumeClaim
secret]				
hostnetwork	false	[]	MustRunAs	MustRunAsRange
MustRunAs	MustRunAs	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
nonroot	false	[]	MustRunAs	MustRunAsNonRoot
RunAsAny	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
privileged	true	[*]	RunAsAny	RunAsAny
RunAsAny	RunAsAny	<none>	false	[*]
restricted	false	[]	MustRunAs	MustRunAsRange
MustRunAs	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	

15.3. EXAMINING A SECURITY CONTEXT CONSTRAINTS OBJECT

To examine a particular SCC, use **oc get**, **oc describe**, **oc export**, or **oc edit**. For example, to examine the **restricted** SCC:


```

$ oc describe scc restricted
Name:      restricted
Priority:   <none>
Access:
  Users:    <none>
  Groups:   system:authenticated
Settings:
  Allow Privileged:  false
  Default Add Capabilities:  <none>
  Required Drop Capabilities:  KILL,MKNOD,SYS_CHROOT,SETUID,SETGID
  Allowed Capabilities:  <none>
  Allowed Seccomp Profiles:  <none>
  Allowed Volume Types:
configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
  Allow Host Network:  false
  Allow Host Ports:    false
  Allow Host PID:      false
  Allow Host IPC:      false
  Read Only Root Filesystem:  false
  Run As User Strategy: MustRunAsRange
    UID:      <none>
    UID Range Min:  <none>
    UID Range Max:  <none>
  SELinux Context Strategy: MustRunAs
    User:      <none>
    Role:      <none>
    Type:      <none>
    Level:     <none>
  FSGroup Strategy: MustRunAs
    Ranges:    <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:    <none>

```



NOTE

In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs other than priority, users, groups, labels, and annotations.

15.4. CREATING NEW SECURITY CONTEXT CONSTRAINTS

To create a new SCC:

1. Define the SCC in a JSON or YAML file:

Security Context Constraint Object Definition

```

kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: scc-admin
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:

```

```

    type: RunAsAny
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  users:
  - my-admin-user
  groups:
  - my-admin-group

```

Optionally, you can add drop capabilities to an SCC by setting the **requiredDropCapabilities** field with the desired values. Any specified capabilities will be dropped from the container. For example, to create an SCC with the **KILL**, **MKNOD**, and **SYS_CHROOT** required drop capabilities, add the following to the SCC object:

```

requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT

```

You can see the list of possible values in the [Docker documentation](#).

TIP

Because capabilities are passed to the Docker, you can use a special **ALL** value to drop all possible capabilities.

- Then, run **oc create** passing the file to create it:

```

$ oc create -f scc_admin.yaml
securitycontextconstraints "scc-admin" created

```

- Verify that the SCC was created:

```

$ oc get scc scc-admin
NAME          PRIV          CAPS          SELINUX     RUNASUSER   FSGROUP
SUPGROUP     PRIORITY     READONLYROOTFS  VOLUMES
scc-admin    true         []             RunAsAny    RunAsAny    RunAsAny
RunAsAny     <none>       false          [awsElasticBlockStore
azureDisk azureFile cephFS cinder configMap downwardAPI emptyDir fc
flexVolume flocker gcePersistentDisk gitRepo glusterfs iscsi nfs
persistentVolumeClaim photonPersistentDisk quobyte rbd secret
vsphere]

```

15.5. DELETING SECURITY CONTEXT CONSTRAINTS

To delete an SCC:

```

$ oc delete scc <scc_name>

```

**NOTE**

If you delete a default SCC, it will be regenerated upon restart.

15.6. UPDATING SECURITY CONTEXT CONSTRAINTS

To update an existing SCC:

```
$ oc edit scc <scc_name>
```

**NOTE**

In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs other than priority, users, and groups.

15.6.1. Example Security Context Constraints Settings

Without Explicit runAsUser Setting

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext: ❶
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

- ❶ When a container or pod does not request a user ID under which it should be run, the effective UID depends on the SCC that emits this pod. Because restricted SCC is granted to all authenticated users by default, it will be available to all users and service accounts and used in most cases. The restricted SCC uses **MustRunAsRange** strategy for constraining and defaulting the possible values of the **securityContext.runAsUser** field. The admission plug-in will look for the **openshift.io/sa.scc.uid-range** annotation on the current project to populate range fields, as it does not provide this range. In the end, a container will have **runAsUser** equal to the first value of the range that is hard to predict because every project has different ranges. See [Understanding Pre-allocated Values and Security Context Constraints](#) for more information.

With Explicit runAsUser Setting

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000 ❶
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

- 1 A container or pod that requests a specific user ID will be accepted by OpenShift Container Platform only when a service account or a user is granted access to a SCC that allows such a user ID. The SCC can allow arbitrary IDs, an ID that falls into a range, or the exact user ID specific to the request.

This works with SELinux, fsGroup, and Supplemental Groups. See [Volume Security](#) for more information.

15.7. UPDATING THE DEFAULT SECURITY CONTEXT CONSTRAINTS

Default SCCs will be created when the master is started if they are missing. To reset SCCs to defaults, or update existing SCCs to new default definitions after an upgrade you may:

1. Delete any SCC you would like to be reset and let it be recreated by restarting the master
2. Use the `oc adm policy reconcile-sccs` command

The `oc adm policy reconcile-sccs` command will set all SCC policies to the default values but retain any additional users, groups, labels, and annotations as well as priorities you may have already set. To view which SCCs will be changed you may run the command with no options or by specifying your preferred output with the `-o <format>` option.

After reviewing it is recommended that you back up your existing SCCs and then use the `--confirm` option to persist the data.



NOTE

If you would like to reset priorities and grants, use the `--additive-only=false` option.



NOTE

If you have customized settings other than priority, users, groups, labels, or annotations in an SCC, you will lose those settings when you reconcile.

15.8. HOW DO I?

The following describe common scenarios and procedures using SCCs.

15.8.1. Grant Access to the Privileged SCC

In some cases, an administrator might want to allow users or groups outside the administrator group access to create more *privileged pods*. To do so, you can:

1. Determine the user or group you would like to have access to the SCC.

**WARNING**

Granting access to a user only works when the user directly creates a pod. For pods created on behalf of a user, **in most cases** by the system itself, **access should be given to a service account** under which related controller is operated upon. Examples of resources that create pods on behalf of a user are Deployments, StatefulSets, DaemonSets, etc.

2. Run:

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

For example, to allow the **e2e-user** access to the **privileged** SCC, run:

```
$ oc adm policy add-scc-to-user privileged e2e-user
```

3. Modify **SecurityContext** of a container to request a privileged mode.

15.8.2. Grant a Service Account Access to the Privileged SCC

First, create a [service account](#). For example, to create service account **mysvcacct** in project **myproject**:

```
$ oc create serviceaccount mysvcacct -n myproject
```

Then, add the service account to the **privileged** SCC.

```
$ oc adm policy add-scc-to-user privileged
system:serviceaccount:myproject:mysvcacct
```

Then, ensure that the resource is being created on behalf of the service account. To do so, set the **spec.serviceAccountName** field to a service account name. Leaving the service account name blank will result in the **default** service account being used.

Then, ensure that at least one of the pod's containers is requesting a privileged mode in the security context.

15.8.3. Enable Images to Run with USER in the Dockerfile

To relax the security in your cluster so that images are not forced to run as a pre-allocated UID, without granting everyone access to the **privileged** SCC:

1. Grant all authenticated users access to the **anyuid** SCC:

```
$ oc adm policy add-scc-to-group anyuid system:authenticated
```

**WARNING**

This allows images to run as the root UID if no **USER** is specified in the *Dockerfile*.

15.8.4. Enable Container Images that Require Root

Some container images (examples: **postgres** and **redis**) require root access and have certain expectations about how volumes are owned. For these images, add the service account to the **anyuid** SCC.

```
$ oc adm policy add-scc-to-user anyuid
system:serviceaccount:myproject:mysvcacct
```

15.8.5. Use --mount-host on the Registry

It is recommended that [persistent storage](#) using **PersistentVolume** and **PersistentVolumeClaim** objects be used for [registry deployments](#). If you are testing and would like to instead use the **oc adm registry** command with the **--mount-host** option, you must first create a new [service account](#) for the registry and add it to the **privileged** SCC. See the [Administrator Guide](#) for full instructions.

15.8.6. Provide Additional Capabilities

In some cases, an image may require capabilities that Docker does not provide out of the box. You can provide the ability to request additional capabilities in the pod specification which will be validated against an SCC.

**IMPORTANT**

This allows images to run with elevated capabilities and should be used only if necessary. You should not edit the default **restricted** SCC to enable additional capabilities.

When used in conjunction with a non-root user, you must also ensure that the file that requires the additional capability is granted the capabilities using the **setcap** command. For example, in the *Dockerfile* of the image:

```
setcap cap_net_raw,cap_net_admin+p /usr/bin/ping
```

Further, if a capability is provided by default in Docker, you do not need to modify the pod specification to request it. For example, **NET_RAW** is provided by default and capabilities should already be set on **ping**, therefore no special steps should be required to run **ping**.

To provide additional capabilities:

1. Create a new SCC
2. Add the allowed capability using the **allowedCapabilities** field.

3. When creating the pod, request the capability in the `securityContext.capabilities.add` field.

15.8.7. Modify Cluster Default Behavior

When you grant access to the **anyuid** SCC for everyone, your cluster:

- Does not pre-allocate UIDs
- Allows containers to run as any user
- Prevents privileged containers

```
$ oc adm policy add-scc-to-group anyuid system:authenticated
```

To modify your cluster so that it does not pre-allocate UIDs and does not allow containers to run as root, grant access to the **nonroot** SCC for everyone:

```
$ oc adm policy add-scc-to-group nonroot system:authenticated
```



WARNING

Be very careful with any modifications that have a cluster-wide impact. When you grant an SCC to all authenticated users, as in the previous example, or modify an SCC that applies to all users, such as the **restricted** SCC, it also affects Kubernetes and OpenShift Container Platform components, including the web console and integrated docker registry. Changes made with these SCCs can cause these components to stop functioning.

Instead, create a custom SCC and target it to only specific users or groups. This way potential issues are confined to the affected users or groups and do not impact critical cluster components.

15.8.8. Use the hostPath Volume Plug-in

To relax the security in your cluster so that pods are allowed to use the **hostPath** volume plug-in without granting everyone access to more privileged SCCs such as **privileged**, **hostaccess**, or **hostmount-anyuid**, perform the following actions:

1. [Create a new SCC](#) named **hostpath**
2. Set the `allowHostDirVolumePlugin` parameter to **true** for the new SCC:

```
$ oc patch scc hostpath -p '{"allowHostDirVolumePlugin": true}'
```

3. Grant access to this SCC to all users:

```
$ oc adm policy add-scc-to-group hostpath system:authenticated
```

Now, all the pods that request **hostPath** volumes are admitted by the **hostpath** SCC.

15.8.9. Ensure That Admission Attempts to Use a Specific SCC First

You may control the sort ordering of SCCs in admission by setting the **Priority** field of the SCCs. See the [SCC Prioritization](#) section for more information on sorting.

15.8.10. Add an SCC to a User, Group, or Project

Before adding an SCC to a user or group, you can first use the **scc-review** option to check if the user or group can create a pod. See the [Authorization](#) topic for more information.

SCCs are not granted directly to a project. Instead, you add a service account to an SCC and either specify the service account name on your pod or, when unspecified, run as the **default** service account.

To add an SCC to a user:

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
```

To add an SCC to a service account:

```
$ oc adm policy add-scc-to-user <scc_name> \
    system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>
```

If you are currently in the project to which the service account belongs, you can use the **-z** flag and just specify the **<serviceaccount_name>**.

```
$ oc adm policy add-scc-to-user <scc_name> -z <serviceaccount_name>
```



IMPORTANT

Usage of the **-z** flag as described above is highly recommended, as it helps prevent typos and ensures that access is granted only to the specified service account. If not in the project, use the **-n** option to indicate the project namespace it applies to.

To add an SCC to a group:

```
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

To add an SCC to all service accounts in a namespace:

```
$ oc adm policy add-scc-to-group <scc_name> \
    system:serviceaccounts:<serviceaccount_namespace>
```


CHAPTER 16. SCHEDULING

16.1. OVERVIEW

16.1.1. Overview

Pod scheduling is an internal process that determines placement of new pods onto nodes within the cluster.

The scheduler code has a clean separation that watches new pods as they get created and identifies the most suitable node to host them. It then creates bindings (pod to node bindings) for the pods using the master API.

16.1.2. Default scheduling

OpenShift Container Platform comes with a default scheduler that serves the needs of most users. The default scheduler uses both inherent and customizable tools to determine the best fit for a pod.

For information on how the default scheduler determines pod placement and available customizable parameters, see [Default Scheduling](#).

16.1.3. Advanced scheduling

In situations where you might want more control over where new pods are placed, the OpenShift Container Platform advanced scheduling features allow you to configure a pod so that the pod is required to (or has a preference to) run on a particular node or alongside a specific pod. Advanced scheduling also allows you to prevent pods from being placed on a node or with another pod.

For information about advanced scheduling, see [Advanced Scheduling](#).

16.1.4. Custom scheduling

OpenShift Container Platform also allows you to use your own or third-party schedulers by editing the pod specification.

For more information, see [Custom Schedulers](#).

16.2. DEFAULT SCHEDULING

16.2.1. Overview

The default OpenShift Container Platform pod scheduler is responsible for determining placement of new pods onto nodes within the cluster. It reads data from the pod and tries to find a node that is a good fit based on configured policies. It is completely independent and exists as a standalone/pluggable solution. It does not modify the pod and just creates a binding for the pod that ties the pod to the particular node.

16.2.2. Generic Scheduler

The existing generic scheduler is the default platform-provided scheduler *engine* that selects a node to host the pod in a three-step operation:

1. The scheduler [filters out inappropriate nodes using predicates](#).

2. The scheduler [prioritizes the filtered list of nodes](#).
3. The scheduler [selects the highest priority node](#) for the pod.

16.2.3. Filter the Nodes

The available nodes are filtered based on the constraints or requirements specified. This is done by running each node through the list of filter functions called [predicates](#).

16.2.3.1. Prioritize the Filtered List of Nodes

This is achieved by passing each node through a series of [priority functions](#) that assign it a score between 0 - 10, with 0 indicating a bad fit and 10 indicating a good fit to host the pod. The scheduler configuration can also take in a simple *weight* (positive numeric value) for each priority function. The node score provided by each priority function is multiplied by the weight (default weight for most priorities is 1) and then combined by adding the scores for each node provided by all the priorities. This weight attribute can be used by administrators to give higher importance to some priorities.

16.2.3.2. Select the Best Fit Node

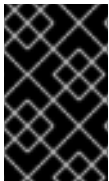
The nodes are sorted based on their scores and the node with the highest score is selected to host the pod. If multiple nodes have the same high score, then one of them is selected at random.

16.2.4. Scheduler Policy

The selection of the [predicate](#) and [priorities](#) defines the policy for the scheduler.

The scheduler configuration file is a JSON file that specifies the predicates and priorities the scheduler will consider.

In the absence of the scheduler policy file, the default configuration file, */etc/origin/master/scheduler.json*, gets applied.



IMPORTANT

The predicates and priorities defined in the scheduler configuration file completely override the default scheduler policy. If any of the default predicates and priorities are required, you must explicitly specify the functions in the scheduler configuration file.

Default scheduler configuration file

```
{
  "apiVersion": "v1",
  "kind": "Policy",
  "predicates": [
    {
      "name": "NoVolumeZoneConflict"
    },
    {
      "name": "MaxEBSVolumeCount"
    },
    {
      "name": "MaxGCEPDVolumeCount"
    }
  ],
}
```

```

    {
      "name": "MaxAzureDiskVolumeCount"
    },
    {
      "name": "MatchInterPodAffinity"
    },
    {
      "name": "NoDiskConflict"
    },
    {
      "name": "GeneralPredicates"
    },
    {
      "name": "PodToleratesNodeTaints"
    },
    {
      "name": "CheckNodeMemoryPressure"
    },
    {
      "name": "CheckNodeDiskPressure"
    },
    {
      "name": "NoVolumeNodeConflict"
    },
    {
      "argument": {
        "serviceAffinity": {
          "labels": [
            "region"
          ]
        }
      },
      "name": "Region"
    }
  ],
  "priorities": [
    {
      "name": "SelectorSpreadPriority",
      "weight": 1
    },
    {
      "name": "InterPodAffinityPriority",
      "weight": 1
    },
    {
      "name": "LeastRequestedPriority",
      "weight": 1
    },
    {
      "name": "BalancedResourceAllocation",
      "weight": 1
    },
    {
      "name": "NodePreferAvoidPodsPriority",
      "weight": 10000
    }
  ]
}

```

```

    },
    {
      "name": "NodeAffinityPriority",
      "weight": 1
    },
    {
      "name": "TaintTolerationPriority",
      "weight": 1
    },
    {
      "argument": {
        "serviceAntiAffinity": {
          "label": "zone"
        }
      },
      "name": "Zone",
      "weight": 2
    }
  ]
}

```

16.2.4.1. Modifying Scheduler Policy

The scheduler policy is defined in a file on the master, named */etc/origin/master/scheduler.json* by default, unless overridden by the `kubernetesMasterConfig.schedulerConfigFile` field in the [master configuration file](#).

Sample modified scheduler configuration file

```

kind: "Policy"
version: "v1"
"predicates": [
  {
    "name": "PodFitsResources"
  },
  {
    "name": "NoDiskConflict"
  },
  {
    "name": "MatchNodeSelector"
  },
  {
    "name": "HostName"
  },
  {
    "name": "NoVolumeNodeConflict"
  },
  {
    "argument": {
      "serviceAffinity": {
        "labels": [
          "region"
        ]
      }
    }
  },
]

```

```

        "name": "Region"
    },
    ],
    "priorities": [
        {
            "name": "LeastRequestedPriority",
            "weight": 1
        },
        {
            "name": "BalancedResourceAllocation",
            "weight": 1
        },
        {
            "name": "ServiceSpreadingPriority",
            "weight": 1
        },
        {
            "argument": {
                "serviceAntiAffinity": {
                    "label": "zone"
                }
            },
            "name": "Zone",
            "weight": 2
        }
    ]
}

```

To modify the scheduler policy:

1. Edit the scheduler configuration file to configure the desired [default predicates and priorities](#). You can create a custom configuration, or use and modify one of the [sample policy configurations](#).
2. Add any [configurable predicates](#) and [configurable priorities](#) you require.
3. Restart the OpenShift Container Platform for the changes to take effect.

```
# systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

16.2.5. Available Predicates

Predicates are rules that filter out unqualified nodes.

There are several predicates provided by default in OpenShift Container Platform. Some of these predicates can be customized by providing certain parameters. Multiple predicates can be combined to provide additional filtering of nodes.

16.2.5.1. Static Predicates

These predicates do not take any configuration parameters or inputs from the user. These are specified in the scheduler configuration using their exact name.

16.2.5.1.1. Default Predicates

The default scheduler policy includes the following predicates:

NoVolumeZoneConflict checks that the volumes a pod requests are available in the zone.

```
{"name" : "NoVolumeZoneConflict"}
```

MaxEBSVolumeCount checks the maximum number of volumes that can be attached to an AWS instance.

```
{"name" : "MaxEBSVolumeCount"}
```

MaxGCEPDVolumeCount checks the maximum number of Google Compute Engine (GCE) Persistent Disks (PD).

```
{"name" : "MaxGCEPDVolumeCount"}
```

MaxAzureDiskVolumeCount checks the maximum number of Azure Disk Volumes.

```
{"name" : "MaxAzureDiskVolumeCount"}
```

MatchInterPodAffinity checks if the pod affinity/antiaffinity rules permit the pod.

```
{"name" : "MatchInterPodAffinity"}
```

NoDiskConflict checks if the volume requested by a pod is available.

```
{"name" : "NoDiskConflict"}
```

GeneralPredicates check whether non-critical predicates and essential predicates pass. See [General Predicates](#) below.

PodToleratesNodeTaints checks if a pod can tolerate the node taints.

```
{"name" : "PodToleratesNodeTaints"}
```

CheckNodeMemoryPressure checks if a pod can be scheduled on a node with a memory pressure condition.

```
{"name" : "CheckNodeMemoryPressure"}
```

CheckNodeDiskPressure checks if a pod can be scheduled on a node with a disk pressure condition.

```
{"name" : "CheckNodeDiskPressure"}
```

NoVolumeNodeConflict

```
{"name" : "NoVolumeNodeConflict"}
```

16.2.5.1.2. Other Supported Predicates

OpenShift Container Platform also supports the following predicates:

CheckVolumeBinding evaluates if a pod can fit based on the volumes, it requests, for both bound and unbound PVCs. * For PVCs that are bound, the predicate checks that the corresponding PV's node affinity is satisfied by the given node. * For PVCs that are unbound, the predicate searched for available PVs that can satisfy the PVC requirements and that the PV node affinity is satisfied by the given node.

The predicate returns true if all bound PVCs have compatible PVs with the node, and if all unbound PVCs can be matched with an available and node-compatible PV.

```
{"name" : "CheckVolumeBinding"}
```

The **CheckVolumeBinding** predicate must be enabled in non-default schedulers.

CheckNodeCondition checks if a pod can be scheduled on a node reporting **out of disk**, **network unavailable**, or **not ready** conditions.

```
{"name" : "CheckNodeCondition"}
```

PodToleratesNodeNoExecuteTaints checks if a pod tolerations can tolerate a node **NoExecute** taints.

```
{"name" : "PodToleratesNodeNoExecuteTaints"}
```

CheckNodeLabelPresence checks if all of the specified labels exist on a node, regardless of their value.

```
{"name" : "CheckNodeLabelPresence"}
```

checkServiceAffinity checks that ServiceAffinity labels are homogeneous for pods that are scheduled on a node.

```
{"name" : "checkServiceAffinity"}
```

16.2.5.2. General Predicates

The following general predicates check whether non-critical predicates and essential predicates pass. Non-critical predicates are the predicates that only non-critical pods need to pass and essential predicates are the predicates that all pods need to pass.

Non-critical general predicates

PodFitsResources determines a fit based on resource availability (CPU, memory, GPU, and so forth). The nodes can declare their resource capacities and then pods can specify what resources they require. Fit is based on requested, rather than used resources.

```
{"name" : "PodFitsResources"}
```

Essential general predicates

PodFitsHostPorts determines if a node has free ports for the requested pod ports (absence of port conflicts).

```
{"name" : "PodFitsHostPorts"}
```

HostName determines fit based on the presence of the Host parameter and a string match with the name of the host.

```
{"name" : "HostName"}
```

■ **MatchNodeSelector** determines fit based on [node selector](#) (`nodeSelector`) queries defined in the pod.

```
{"name" : "MatchNodeSelector"}
```

16.2.5.3. Configurable Predicates

You can configure these predicates in the scheduler configuration, by default `/etc/origin/master/scheduler.json`, to add labels to affect how the predicate functions.

Since these are configurable, multiple predicates of the same type (but different configuration parameters) can be combined as long as their user-defined names are different.

For information on using these priorities, see [Modifying Scheduler Policy](#).

ServiceAffinity places pods on nodes based on the service running on that pod. Placing pods of the same service on the same or co-located nodes can lead to higher efficiency.

This predicate attempts to place pods with specific labels in its [node selector](#) on nodes that have the same label.

If the pod does not specify the labels in its node selector, then the first pod is placed on any node based on availability and all subsequent pods of the service are scheduled on nodes that have the same label values as that node.

```
"predicates":[
  {
    "name":"<name>", ❶
    "argument":{
      "serviceAffinity":{
        "labels":[
          "<label>" ❷
        ]
      }
    }
  },
],
```

❶ Specify a name for the predicate.

❷ Specify a label to match.

For example:

```
"name":"ZoneAffinity",
"argument":{
  "serviceAffinity":{
    "labels":[
      "rack"
    ]
  }
}
```


For example, if the first pod of a service had a node selector **rack** was scheduled to a node with label **region=rack**, all the other subsequent pods belonging to the same service will be scheduled on nodes with the same **region=rack** label. For more information, see [Controlling Pod Placement](#).

Multiple-level labels are also supported. Users can also specify all pods for a service to be scheduled on nodes within the same region and within the same zone (under the region).

The **labelsPresence** parameter checks whether a particular node has a specific label. The labels create node *groups* that the **LabelPreference** priority uses. Matching by label can be useful, for example, where nodes have their physical location or status defined by labels.

```
"predicates":[
  {
    "name": "<name>", ❶
    "argument":{
      "labelsPresence":{
        "labels":[
          "<label>" ❷
        ],
        "presence": true ❸
      }
    }
  }
],
```

❶ Specify a name for the predicate.

❷ Specify a label to match.

❸ Specify whether the labels are required, either **true** or **false**.

- For **presence: false**, if any of the requested labels are present in the node labels, the pod cannot be scheduled. If the labels are not present, the pod can be scheduled.
- For **presence: true**, if all of the requested labels are present in the node labels, the pod can be scheduled. If all of the labels are not present, the pod is not scheduled.

For example:

```
"name": "RackPreferred",
"argument":{
  "labelsPresence":{
    "labels":[
      "rack",
      "region"
    ],
    "presence": true
  }
}
```

16.2.6. Available Priorities

Priorities are rules that rank remaining nodes according to preferences.

A custom set of priorities can be specified to configure the scheduler. There are several priorities provided by default in OpenShift Container Platform. Other priorities can be customized by providing certain parameters. Multiple priorities can be combined and different weights can be given to each in order to impact the prioritization.

16.2.6.1. Static Priorities

Static priorities do not take any configuration parameters from the user, except weight. A weight is required to be specified and cannot be 0 or negative.

These are specified in the scheduler configuration, by default `/etc/origin/master/scheduler.json`.

16.2.6.1.1. Default Priorities

The default scheduler policy includes the following priorities:

The default scheduler policy includes the priorities noted in the list. Each of the priority function has a weight of **1** except **NodePreferAvoidPodsPriority**, which has a weight of **10000**.

SelectorSpreadPriority looks for services, replication controllers (RC), replication sets (RS), and stateful sets that match the pod, then finds existing pods that match those selectors. The scheduler favors nodes that have fewer existing matching pods. Then, it schedules the pod on a node with the smallest number of pods that match those selectors as the pod being scheduled.

```
{"name" : "SelectorSpreadPriority", "weight" : 1}
```

InterPodAffinityPriority computes a sum by iterating through the elements of **weightedPodAffinityTerm** and adding *weight* to the sum if the corresponding PodAffinityTerm is satisfied for that node. The node(s) with the highest sum are the most preferred.

```
{"name" : "InterPodAffinityPriority", "weight" : 1}
```

LeastRequestedPriority favors nodes with fewer requested resources. It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes nodes that have the highest available/remaining capacity.

```
{"name" : "LeastRequestedPriority", "weight" : 1}
```

BalancedResourceAllocation favors nodes with balanced resource usage rate. It calculates the difference between the consumed CPU and memory as a fraction of capacity, and prioritizes the nodes based on how close the two metrics are to each other. This should always be used together with **LeastRequestedPriority**.

```
{"name" : "BalancedResourceAllocation", "weight" : 1}
```

NodePreferAvoidPodsPriority ignores pods that are owned by a controller other than a replication controller.

```
{"name" : "NodePreferAvoidPodsPriority", "weight" : 10000}
```

NodeAffinityPriority prioritizes nodes according to node affinity scheduling preferences

```
{"name" : "NodeAffinityPriority", "weight" : 1}
```

TaintTolerationPriority prioritizes nodes that have a fewer number of *intolerable* taints on them for a pod. An intolerable taint is one which has key **PreferNoSchedule**.

```
{ "name" : "TaintTolerationPriority", "weight" : 1 }
```

16.2.6.1.2. Other Priorities

OpenShift Container Platform also supports the following priorities:

EqualPriority gives an equal weight of **1** to all nodes, if no priority configurations are provided. We recommend using this priority only for testing environments.

```
{ "name" : "EqualPriority", "weight" : 1 }
```

MostRequestedPriority prioritizes nodes with most requested resources. It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes based on the maximum of the average of the fraction of requested to capacity.

```
{ "name" : "MostRequestedPriority", "weight" : 1 }
```

ImageLocalityPriority prioritizes nodes that already have requested pod container's images.

```
{ "name" : "ImageLocalityPriority", "weight" : 1 }
```

ServiceSpreadingPriority spreads pods by minimizing the number of pods belonging to the same service onto the same machine.

```
{ "name" : "ServiceSpreadingPriority", "weight" : 1 }
```

16.2.6.2. Configurable Priorities

You can configure these priorities in the scheduler configuration, by default **/etc/origin/master/scheduler.json**, to add labels to affect how the priorities.

The type of the priority function is identified by the argument that they take. Since these are configurable, multiple priorities of the same type (but different configuration parameters) can be combined as long as their user-defined names are different.

For information on using these priorities, see [Modifying Scheduler Policy](#).

ServiceAntiAffinity takes a label and ensures a good spread of the pods belonging to the same service across the group of nodes based on the label values. It gives the same score to all nodes that have the same value for the specified label. It gives a higher score to nodes within a group with the least concentration of pods.

```
"priorities":[
  {
    "name":"<name>", 1
    "weight" : 1 2
    "argument":{
      "serviceAntiAffinity":{
        "label":[
          "<label>" 3
        ]
      }
    }
  }
]
```

```

    ]
  }
}
]

```

- 1 Specify a name for the priority.
- 2 Specify a weight. Enter a non-zero positive value.
- 3 Specify a label to match.

For example:

```

    "name": "RackSpread", 1
    "weight" : 1 2
    "argument": {
      "serviceAntiAffinity": {
        "label": "rack" 3
      }
    }
  }
}

```

- 1 Specify a name for the priority.
- 2 Specify a weight. Enter a non-zero positive value.
- 3 Specify a label to match.



NOTE

In some situations using **ServiceAntiAffinity** based on custom labels does not spread pod as expected. See [this Red Hat Solution](#).

*The **labelPreference** parameter gives priority based on the specified label. If the label is present on a node, that node is given priority. If no label is specified, priority is given to nodes that do not have a label.

```

    "priorities": [
      {
        "name": "<name>", 1
        "weight" : 1 2
        "argument": {
          "labelPreference": {
            "label": "<label>", 3
            "presence": true 4
          }
        }
      }
    ]
  }
}

```

- 1 Specify a name for the priority.

- 2 Specify a weight. Enter a non-zero positive value.
- 3 Specify a label to match.
- 4 Specify whether the label is required, either **true** or **false**.

16.2.7. Use Cases

One of the important use cases for scheduling within OpenShift Container Platform is to support flexible affinity and anti-affinity policies.

16.2.7.1. Infrastructure Topological Levels

Administrators can define multiple topological levels for their infrastructure (nodes) by specifying [labels on nodes](#) (e.g., **region=r1**, **zone=z1**, **rack=s1**).

These label names have no particular meaning and administrators are free to name their infrastructure levels anything (eg, city/building/room). Also, administrators can define any number of levels for their infrastructure topology, with three levels usually being adequate (such as: **regions** → **zones** → **racks**). Administrators can specify affinity and anti-affinity rules at each of these levels in any combination.

16.2.7.2. Affinity

Administrators should be able to configure the scheduler to specify affinity at any topological level, or even at multiple levels. Affinity at a particular level indicates that all pods that belong to the same service are scheduled onto nodes that belong to the same level. This handles any latency requirements of applications by allowing administrators to ensure that peer pods do not end up being too geographically separated. If no node is available within the same affinity group to host the pod, then the pod is not scheduled.

If you need greater control over where the pods are scheduled, see [Using Node Affinity](#) and [Using Pod Affinity and Anti-affinity](#). These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

16.2.7.3. Anti Affinity

Administrators should be able to configure the scheduler to specify anti-affinity at any topological level, or even at multiple levels. Anti-affinity (or 'spread') at a particular level indicates that all pods that belong to the same service are spread across nodes that belong to that level. This ensures that the application is well spread for high availability purposes. The scheduler tries to balance the service pods across all applicable nodes as evenly as possible.

If you need greater control over where the pods are scheduled, see [Using Node Affinity](#) and [Using Pod Affinity and Anti-affinity](#). These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

16.2.8. Sample Policy Configurations

The configuration below specifies the default scheduler configuration, if it were to be specified via the scheduler policy file.

```
kind: "Policy"
version: "v1"
```

```

predicates:
...
- name: "RegionZoneAffinity" ❶
  argument:
    serviceAffinity: ❷
      labels: ❸
        - "region"
        - "zone"
priorities:
...
- name: "RackSpread" ❹
  weight: 1
  argument:
    serviceAntiAffinity: ❺
      label: "rack" ❻

```

❶ The name for the predicate.

❷ The [type of predicate](#).

❸ The labels for the predicate.

❹ The name for the priority.

❺ The [type of priority](#).

❻ The labels for the priority.

In all of the sample configurations below, the list of predicates and priority functions is truncated to include only the ones that pertain to the use case specified. In practice, a complete/meaningful scheduler policy should include most, if not all, of the default predicates and priorities listed above.

The following example defines three topological levels, region (affinity) → zone (affinity) → rack (anti-affinity):

```

kind: "Policy"
version: "v1"
predicates:
...
- name: "RegionZoneAffinity"
  argument:
    serviceAffinity:
      labels:
        - "region"
        - "zone"
priorities:
...
- name: "RackSpread"
  weight: 1
  argument:
    serviceAntiAffinity:
      label: "rack"

```

The following example defines three topological levels, city (affinity) → building (anti-affinity) → room (anti-affinity):

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "CityAffinity"
  argument:
    serviceAffinity:
      labels:
        - "city"
priorities:
...
- name: "BuildingSpread"
  weight: 1
  argument:
    serviceAntiAffinity:
      label: "building"
- name: "RoomSpread"
  weight: 1
  argument:
    serviceAntiAffinity:
      label: "room"
```

The following example defines a policy to only use nodes with the 'region' label defined and prefer nodes with the 'zone' label defined:

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "RequireRegion"
  argument:
    labelsPresence:
      labels:
        - "region"
      presence: true
priorities:
...
- name: "ZonePreferred"
  weight: 1
  argument:
    labelPreference:
      label: "zone"
      presence: true
```

The following example combines both static and configurable predicates and also priorities:

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "RegionAffinity"
  argument:
```

```

      serviceAffinity:
        labels:
          - "region"
- name: "RequireRegion"
  argument:
    labelsPresence:
      labels:
        - "region"
      presence: true
- name: "BuildingNodesAvoid"
  argument:
    labelsPresence:
      labels:
        - "building"
      presence: false
- name: "PodFitsPorts"
- name: "MatchNodeSelector"
priorities:
...
- name: "ZoneSpread"
  weight: 2
  argument:
    serviceAntiAffinity:
      label: "zone"
- name: "ZonePreferred"
  weight: 1
  argument:
    labelPreference:
      label: "zone"
      presence: true
- name: "ServiceSpreadingPriority"
  weight: 1

```

16.3. CUSTOM SCHEDULING

16.3.1. Overview

You can run multiple, custom schedulers alongside the default scheduler and configure which scheduler to use for each pods.

To schedule a given pod using a specific scheduler, [specify the name of the scheduler in that pod specification](#).

16.3.2. Deploying the Scheduler

The steps below are the general process for deploying a scheduler into your cluster.



NOTE

Information on how to create/deploy a scheduler is outside the scope of this document. For an example, see [plugin/pkg/scheduler](#) in the Kubernetes source directory.

1. Create or edit a pod configuration and specify the name of the scheduler with the **`schedulerName`** parameter. The name must be unique.

Sample pod specification with scheduler

```
apiVersion: v1
kind: Pod
metadata:
  name: custom-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: custom-scheduler 1
  containers:
  - name: pod-with-second-annotation-container
    image: docker.io/ocpqe/hello-pod
```

- 1** The name of the scheduler to use. When no scheduler name is supplied, the pod is automatically scheduled using the default scheduler.

2. Run the following command to create the pod:

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

3. Run the following command to check that the pod was created with the custom scheduler:

```
$ oc get pod custom-scheduler -o yaml
```

4. Run the following command to check the status of the pod:

```
$ oc get pod
```

The pod should not be running.

NAME	READY	STATUS	RESTARTS	AGE
custom-scheduler	0/1	Pending	0	2m

5. Deploy the custom scheduler.

6. Run the following command to check the status of the pod:

```
$ oc get pod
```

The pod should be running.

NAME	READY	STATUS	RESTARTS	AGE
custom-scheduler	1/1	Running	0	4m

7. Run the following command to check that the scheduler was used:

```
$ oc describe pod custom-scheduler
```

The name of the scheduler is listed, as shown in the following truncated output:

```
[...]
```

```

Events:
  FirstSeen    LastSeen  Count    From              SubObjectPath  Type
Reason Message
  -----
  1m          1m         1      my-scheduler      Normal
Scheduled Successfully assigned custom-scheduler to <$node1>
[...]
```

16.4. CONTROLLING POD PLACEMENT

16.4.1. Overview

As a cluster administrator, you can set a policy to prevent application developers with certain roles from targeting specific nodes when scheduling pods.

The Pod Node Constraints admission controller ensures that pods are deployed onto only specified node hosts using labels] and prevents users without a specific role from using the **nodeSelector** field to schedule pods.

16.4.2. Constraining Pod Placement Using Node Name

Use the Pod Node Constraints admission controller to ensure a pod is deployed onto only a specified node host by assigning it a label and specifying this in the **nodeName** setting in a pod configuration.

1. Ensure you have the desired labels (see [Updating Labels on Nodes](#) for details) and **node selector** set up in your environment.
For example, make sure that your pod configuration features the **nodeName** value indicating the desired label:

```

apiVersion: v1
kind: Pod
spec:
  nodeName: <value>
```

2. Modify the master configuration file, */etc/origin/master/master-config.yaml*, to add **PodNodeConstraints** to the **admissionConfig** section:

```

...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
...

```

3. Restart OpenShift Container Platform for the changes to take effect.

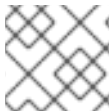
```
# systemctl restart atomic-openshift-master
```

16.4.3. Constraining Pod Placement Using a Node Selector

Using [node selectors](#), you can ensure that pods are only placed onto nodes with specific labels. As a cluster administrator, you can use the Pod Node Constraints admission controller to set a policy that prevents users without the **pods/binding** permission from using node selectors to schedule pods.

The **nodeSelectorLabelBlacklist** field of a master configuration file gives you control over the labels that certain roles can specify in a pod configuration's **nodeSelector** field. Users, service accounts, and groups that have the **pods/binding** permission [role](#) can specify any node selector. Those without the **pods/binding** permission are prohibited from setting a **nodeSelector** for any label that appears in **nodeSelectorLabelBlacklist**.

For example, an OpenShift Container Platform cluster might consist of five data centers spread across two regions. In the U.S., **us-east**, **us-central**, and **us-west**; and in the Asia-Pacific region (APAC), **apac-east** and **apac-west**. Each node in each geographical region is labeled accordingly. For example, **region: us-east**.



NOTE

See [Updating Labels on Nodes](#) for details on assigning labels.

As a cluster administrator, you can create an infrastructure where application developers should be deploying pods only onto the nodes closest to their geographical location. You can create a node selector, grouping the U.S. data centers into **superregion: us** and the APAC data centers into **superregion: apac**.

To maintain an even loading of resources per data center, you can add the desired **region** to the **nodeSelectorLabelBlacklist** section of a master configuration. Then, whenever a developer located in the U.S. creates a pod, it is deployed onto a node in one of the regions with the **superregion: us** label. If the developer tries to target a specific region for their pod (for example, **region: us-east**), they receive an error. If they try again, without the node selector on their pod, it can still be deployed onto the region they tried to target, because **superregion: us** is set as the project-level node selector, and nodes labeled **region: us-east** are also labeled **superregion: us**.

1. Ensure you have the desired labels (see [Updating Labels on Nodes](#) for details) and [node selector](#) set up in your environment.

For example, make sure that your pod configuration features the **nodeSelector** value indicating the desired label:

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
  ...
```

2. Modify the master configuration file, **/etc/origin/master/master-config.yaml**, to add **nodeSelectorLabelBlacklist** to the **admissionConfig** section with the labels that are assigned to the node hosts you want to deny pod placement:

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
```

```

kind: PodNodeConstraintsConfig
nodeSelectorLabelBlacklist:
  - kubernetes.io/hostname
  - <label>
...

```

- Restart OpenShift Container Platform for the changes to take effect.

```
# systemctl restart atomic-openshift-master
```

16.4.4. Control Pod Placement to Projects

The Pod Node Selector admission controller allows you to force pods onto nodes associated with a specific project and prevent pods from being scheduled in those nodes.

The Pod Node Selector admission controller determines where a pod can be placed using [labels on projects](#) and node selectors specified in pods. A new pod will be placed on a node associated with a project only if the node selectors in the pod match the labels in the project.

After the pod is created, the node selectors are merged into the pod so that the pod specification includes the labels originally included in the specification and any new labels from the node selectors. The example below illustrates the merging effect.

The Pod Node Selector admission controller also allows you to create a list of labels that are permitted in a specific project. This list acts as a *whitelist* that lets developers know what labels are acceptable to use in a project and gives administrators greater control over labeling in a cluster.

To activate the **Pod Node Selector** admission controller:

- Configure the **Pod Node Selector** admission controller and whitelist, using one of the following methods:

- Add the following to the master configuration file, */etc/origin/master/master-config.yaml*:

```

admissionConfig:
  pluginConfig:
    PodNodeSelector:
      configuration:
        podNodeSelectorPluginConfig: ❶
        clusterDefaultNodeSelector: "k3=v3" ❷
        ns1: region=west,env=test,infra=fedora,os=fedora ❸

```

- ❶ Adds the **Pod Node Selector** admission controller plug-in.
- ❷ Creates default labels for all nodes.
- ❸ Creates a whitelist of permitted labels in the specified project. Here, the project is **ns1** and the labels are the **key=value** pairs that follow.

- Create a file containing the admission controller information:

```

podNodeSelectorPluginConfig:
  clusterDefaultNodeSelector: "k3=v3"
  ns1: region=west,env=test,infra=fedora,os=fedora

```

Then, reference the file in the master configuration:

```
admissionConfig:
  pluginConfig:
    PodNodeSelector:
      location: <path-to-file>
```



NOTE

If a project does not have node selectors specified, the pods associated with that project will be merged using the default node selector (**clusterDefaultNodeSelector**).

- Restart OpenShift Container Platform for the changes to take effect.

```
# systemctl restart atomic-openshift-master
```

- Create a project object that includes the **scheduler.alpha.kubernetes.io/node-selector** annotation and labels.

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1
  annotations:
    scheduler.alpha.kubernetes.io/node-selector:
env=test,infra=fedora ❶
spec: {},
status: {}
```

- Annotation to create the labels to match the project label selector. Here, the key/value labels are **env=test** and **infra=fedora**.



NOTE

When using the **Pod Node Selector** admission controller, you cannot use **oc adm new-project <project-name>** for setting project node selector. When you set the project node selector using the **oc adm new-project myproject --node-selector='type=user-node,region=<region>'** command, OpenShift Container Platform sets the **openshift.io/node-selector** annotation, which is processed by **NodeEnv** admission plugin.

- Create a pod specification that includes the labels in the node selector, for example:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: hello-pod
spec:
  containers:
```

```

- image: "docker.io/ocpqe/hello-pod:latest"
  imagePullPolicy: IfNotPresent
  name: hello-pod
  ports:
    - containerPort: 8080
      protocol: TCP
  resources: {}
  securityContext:
    capabilities: {}
    privileged: false
  terminationMessagePath: /dev/termination-log
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  nodeSelector: ❶
    env: test
    os: fedora
  serviceAccount: ""
  status: {}

```

- ❶ Node selectors to match project labels.

5. Create the pod in the project:

```
# oc create -f pod.yaml --namespace=ns1
```

6. Check that the node selector labels were added to the pod configuration:

```

get pod pod1 --namespace=ns1 -o json

nodeSelector": {
  "env": "test",
  "infra": "fedora",
  "os": "fedora"
}

```

The node selectors are merged into the pod and the pod should be scheduled in the appropriate project.

If you create a pod with a label that is not specified in the project specification, the pod is not scheduled on the node.

For example, here the label **env: production** is not in any project specification:

```

nodeSelector:
  "env: production"
  "infra": "fedora",
  "os": "fedora"

```

If there is a node that does not have a node selector annotation, the pod will be scheduled there.

16.5. ADVANCED SCHEDULING

16.5.1. Overview

Advanced scheduling involves configuring a pod so that the pod is required to run on particular nodes or has a preference to run on particular nodes.

Generally, advanced scheduling is not necessary, as the OpenShift Container Platform automatically places pods in a reasonable manner. For example, the default scheduler attempts to distribute pods across the nodes evenly and considers the available resources in a node. However, you might want more control over where a pod is placed.

If a pod needs to be on a machine with a faster disk speed (or prevented from being placed on that machine) or pods from two different services need to be located so they can communicate, you can use advanced scheduling to make that happen.

To ensure that appropriate new pods are scheduled on a dedicated group of nodes and prevent other new pods from being scheduled on those nodes, you can combine these methods as needed.

16.5.2. Using Advanced Scheduling

There are several ways to invoke advanced scheduling in your cluster:

Pod Affinity and Anti-affinity

Pod affinity allows a **pod** to specify an affinity (or anti-affinity) towards a group of **pods** (for an application's latency requirements, due to security, and so forth) it can be placed with. The node does not have control over the placement.

Pod affinity uses labels on nodes and label selectors on pods to create rules for pod placement. Rules can be mandatory (required) or best-effort (preferred).

See [Using Pod Affinity and Anti-affinity](#).

Node Affinity

Node affinity allows a **pod** to specify an affinity (or anti-affinity) towards a group of **nodes** (due to their special hardware, location, requirements for high availability, and so forth) it can be placed on. The node does not have control over the placement.

Node affinity uses labels on nodes and label selectors on pods to create rules for pod placement. Rules can be mandatory (required) or best-effort (preferred).

See [Using Node Affinity](#).

Node Selectors

Node selectors are the simplest form of advanced scheduling. Like node affinity, node selectors also use labels on nodes and label selectors on pods to allow a **pod** to control the **nodes** on which it can be placed. However, node selectors do not have required and preferred rules that node affinities have.

See [Using Node Selectors](#).

Taints and Tolerations

Taints/Tolerations allow the **node** to control which **pods** should (or should not) be scheduled on them. Taints are labels on a node and tolerations are labels on a pod. The labels on the pod must match (or tolerate) the label (taint) on the node in order to be scheduled.

Taints/tolerations have one advantage over affinities. For example, if you add to a cluster a new group of nodes with different labels, you would need to update affinities on each of the pods you want to access the node and on any other pods you do not want to use the new nodes. With taints/tolerations, you would only need to update those pods that are required to land on those new nodes, because other pods would be repelled.

See [Using Taints and Tolerations](#).

16.6. ADVANCED SCHEDULING AND NODE AFFINITY

16.6.1. Overview

Node affinity is a set of rules used by the scheduler to determine where a pod can be placed. The rules are defined using custom [labels on nodes](#) and label selectors specified in pods. Node affinity allows a **pod** to specify an affinity (or anti-affinity) towards a group of **nodes** it can be placed on. The node does not have control over the placement.

For example, you could configure a pod to only run on a node with a specific CPU or in a specific availability zone.

There are two types of node affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.



NOTE

If labels on a node change at runtime that results in an node affinity rule on a pod no longer being met, the pod continues to run on the node.

16.6.2. Configuring Node Affinity

You configure node affinity through the pod specification file. You can specify a [required rule](#), a [preferred rule](#), or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example is a pod specification with a rule that requires the pod be placed on a node with a label whose key is **e2e-az-NorthSouth** and whose value is either **e2e-az-North** or **e2e-az-South**:

Sample pod configuration file with a node affinity required rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-NorthSouth ❸
                operator: In ❹
                values:
                  - e2e-az-North ❺
                  - e2e-az-South ❻
  containers:
    - name: with-node-affinity
```



```
image: docker.io/ocpqe/hello-pod
```

- ❶ The stanza to configure node affinity.
- ❷ Defines a required rule.
- ❸❹❺ The key/value pair (label) that must be matched to apply the rule.
- ❻ The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the pod specification. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

The following example is a node specification with a preferred rule that a node with a label whose key is **e2e-az-EastWest** and whose value is either **e2e-az-East** or **e2e-az-West** is preferred for the pod:

Sample pod configuration file with a node affinity preferred rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: ❶
      preferredDuringSchedulingIgnoredDuringExecution: ❷
        - weight: 1 ❸
          preference:
            matchExpressions:
              - key: e2e-az-EastWest ❹
                operator: In ❺
                values:
                  - e2e-az-East ❻
                  - e2e-az-West ❼
  containers:
    - name: with-node-affinity
      image: docker.io/ocpqe/hello-pod
```

- ❶ The stanza to configure node affinity.
- ❷ Defines a preferred rule.
- ❸ Specifies a weight for a preferred rule. The node with highest weight is preferred.
- ❹❺❻ The key/value pair (label) that must be matched to apply the rule.
- ❼ The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the pod specification. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

There is no explicit *node anti-affinity* concept, but using the **NotIn** or **DoesNotExist** operator replicates that behavior.



NOTE

If you are using node affinity and [node selectors](#) in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

16.6.2.1. Configuring a Required Node Affinity Rule

Required rules **must** be met before a pod can be scheduled on a node.

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler is required to place on the node.

1. Add a label to a node by editing the node configuration or by using the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az1
```

2. In the pod specification, use the **nodeAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter:
 - a. Specify the key and values that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **value** parameters as the label in the node.
 - b. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt**, or **Gt**. For example, use the operator **In** to require the label to be in the node:

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-name
                operator: In
              values:
                - e2e-az1
                - e2e-az2
```

3. Create the pod:

```
$ oc create -f e2e-az2.yaml
```

16.6.2.2. Configuring a Preferred Node Affinity Rule

Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler tries to place on the node.

1. Add a label to a node by editing the node configuration or by executing the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2. In the pod specification, use the **nodeAffinity** stanza to configure the **preferredDuringSchedulingIgnoredDuringExecution** parameter:
 - a. Specify a weight for the node, as a number 1-100. The node with highest weight is preferred.
 - b. Specify the key and values that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **value** parameters as the label in the node:

```
preferredDuringSchedulingIgnoredDuringExecution:
- weight: 1
  preference:
    matchExpressions:
    - key: e2e-az-name
      operator: In
      values:
      - e2e-az3
```

3. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt**, or **Gt**. For example, use the operator **In** to require the label to be in the node.
4. Create the pod.

```
$ oc create -f e2e-az3.yaml
```

16.6.3. Examples

The following examples demonstrate node affinity.

16.6.3.1. Node Affinity with Matching Labels

The following example demonstrates node affinity for a node and pod with matching labels:

- The **Node1** node has the label **zone:us**:

```
$ oc label node node1 zone=us
```

- The pod **pod-s1** has the **zone** and **us** key/value pair under a required node affinity rule:

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
```

```

    name: pod-s1
  spec:
    containers:
      - image: "docker.io/ocpqe/hello-pod"
        name: hello-pod
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: "zone"
                  operator: In
                  values:
                    - us

```

- Create the pod using the standard command:

```

$ oc create -f pod-s1.yaml
pod "pod-s1" created

```

- The pod **pod-s1** can be scheduled on **Node1**:

```

oc get pod -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s1	1/1	Running	0	4m	IP1	node1

16.6.3.2. Node Affinity with No Matching Labels

The following example demonstrates node affinity for a node and pod without matching labels:

- The **Node1** node has the label **zone: emea**:

```

$ oc label node node1 zone=emea

```

- The pod **pod-s1** has the **zone** and **us** key/value pair under a required node affinity rule:

```

$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
                operator: In
                values:
                  - us

```

- The pod **pod-s1** cannot be scheduled on **Node1**:

```
oc describe pod pod-s1
<---snip--->
Events:
  FirstSeen    LastSeen  Count    From              SubObjectPath  Type
Reason
-----
1m           33s       8        default-scheduler Warning
FailedScheduling    No nodes are available that match all of the
following predicates: MatchNodeSelector (1).
```

16.7. ADVANCED SCHEDULING AND POD AFFINITY AND ANTI-AFFINITY

16.7.1. Overview

Pod affinity and *pod anti-affinity* allow you to specify rules about how pods should be placed relative to other pods. The rules are defined using custom [labels on nodes](#) and label selectors specified in pods. Pod affinity/anti-affinity allows a **pod** to specify an affinity (or anti-affinity) towards a group of **pods** it can be placed with. The node does not have control over the placement.

For example, using affinity rules, you could spread or pack pods within a service or relative to pods in other services. Anti-affinity rules allow you to prevent pods of a particular service from scheduling on the same nodes as pods of another service that are known to interfere with the performance of the pods of the first service. Or, you could spread the pods of a service across nodes or availability zones to reduce correlated failures.

Pod affinity/anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on the labels on other pods. A [label](#) is a key/value pair.

- Pod affinity can tell the scheduler to locate a new pod on the same node as other pods if the label selector on the new pod matches the label on the current pod.
- Pod anti-affinity can prevent the scheduler from locating a new pod on the same node as pods with the same labels if the label selector on the new pod matches the label on the current pod.

There are two types of pod affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

16.7.2. Configuring Pod Affinity and Anti-affinity

You configure pod affinity/anti-affinity through the pod specification files. You can specify a [required rule](#), a [preferred rule](#), or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example shows a pod specification configured for pod affinity and anti-affinity.

In this example, the pod affinity rule indicates that the pod can schedule onto a node only if that node has at least one already-running pod with a label that has the key **security** and value **S1**. The pod anti-affinity rule says that the pod prefers to not schedule onto a node if that node is already running a pod

with label having key **security** and value **S2**.

Sample pod config file with pod affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        - labelSelector:
            matchExpressions:
              - key: security ❸
                operator: In ❹
                values:
                  - S1 ❺
            topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
```

❶ Stanza to configure pod affinity.

❷ Defines a required rule.

❸ ❺ The key and value (label) that must be matched to apply the rule.

❹ The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.

Sample pod config file with pod anti-affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: ❶
      preferredDuringSchedulingIgnoredDuringExecution: ❷
        - weight: 100 ❸
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: security ❹
                  operator: In ❺
                  values:
                    - S2
            topologyKey: kubernetes.io/hostname
```

```
containers:
- name: with-pod-affinity
  image: docker.io/ocpqe/hello-pod
```

- 1 Stanza to configure pod anti-affinity.
- 2 Defines a preferred rule.
- 3 Specifies a weight for a preferred rule. The node with the highest weight is preferred.
- 4 Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.
- 5 The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.



NOTE

If labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod continues to run on the node.

16.7.2.1. Configuring an Affinity Rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses affinity to allow scheduling with that pod.

1. Create a pod with a specific label in the pod specification:

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

2. When creating other pods, edit the pod specification as follows:

- a. Use the **podAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter or **preferredDuringSchedulingIgnoredDuringExecution** parameter:
- b. Specify the key and value that must be met. If you want the new pod to be scheduled with the other pod, use the same **key** and **value** parameters as the label on the first pod.

```
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
```

```

- key: security
  operator: In
  values:
  - S1
topologyKey: failure-domain.beta.kubernetes.io/zone

```

- c. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.
 - d. Specify a **topologyKey**, which is a prepopulated [Kubernetes label](#) that the system uses to denote such a topology domain.
3. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

16.7.2.2. Configuring an Anti-affinity Rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses an anti-affinity preferred rule to attempt to prevent scheduling with that pod.

1. Create a pod with a specific label in the pod specification:

```

$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s2
  labels:
    security: S2
spec:
  containers:
  - name: security-s2
    image: docker.io/ocpqe/hello-pod

```

2. When creating other pods, edit the pod specification to set the following parameters:
3. Use the **podAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter or **preferredDuringSchedulingIgnoredDuringExecution** parameter:
 - a. Specify a weight for the node, 1-100. The node that with highest weight is preferred.
 - b. Specify the key and values that must be met. If you want the new pod to not be scheduled with the other pod, use the same **key** and **value** parameters as the label on the first pod.

```

podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In

```



```

      values:
      - S2
    topologyKey: kubernetes.io/hostname

```

- c. For a preferred rule, specify a weight, 1-100.
- d. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.
4. Specify a **topologyKey**, which is a prepopulated [Kubernetes label](#) that the system uses to denote such a topology domain.
5. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

16.7.3. Examples

The following examples demonstrate pod affinity and pod anti-affinity.

16.7.3.1. Pod Affinity

The following example demonstrates pod affinity for pods with matching labels and label selectors.

- The pod **team4** has the label **team:4**.

```

$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod

```

- The pod **team4a** has the label selector **team:4** under **podAffinity**.

```

$ cat pod-team4a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4a
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: team
            operator: In
            values:

```

```

      - "4"
      topologyKey: kubernetes.io/hostname
    containers:
    - name: pod-affinity
      image: docker.io/ocpqe/hello-pod

```

- The **team4a** pod is scheduled on the same node as the **team4** pod.

16.7.3.2. Pod Anti-affinity

The following example demonstrates pod anti-affinity for pods with matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```

cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod

```

- The pod **pod-s2** has the label selector **security:s1** under **podAntiAffinity**.

```

cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s1
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-antiaffinity
    image: docker.io/ocpqe/hello-pod

```

- The pod **pod-s2** is not scheduled unless there is a node with a pod that has the **security:s2** label. If there is no other pod with that label, the new pod remains in a pending state:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s2	0/1	Pending	0	32s	<none>	

16.7.3.3. Pod Affinity with no Matching Labels

The following example demonstrates pod affinity for pods without matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** has the label selector **security:s2**.

```
$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** cannot be scheduled on the same node as **pod-s1**.

16.8. ADVANCED SCHEDULING AND NODE SELECTORS

16.8.1. Overview

A *node selector* specifies a map of key-value pairs. The rules are defined using custom [labels on nodes](#) and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the indicated key-value pairs as the label on the node.

If you are using node affinity and [node selectors](#) in the same pod configuration, see the [important considerations](#) below.

16.8.2. Configuring Node Selectors

Using **nodeSelector** in a pod configuration, you can ensure that pods are only placed onto nodes with specific labels.

1. Ensure you have the desired labels (see [Updating Labels on Nodes](#) for details) and **node selector** set up in your environment.

For example, make sure that your pod configuration features the **nodeSelector** value indicating the desired label:

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
  ...
```

2. Modify the master configuration file, `/etc/origin/master/master-config.yaml`, to add **nodeSelectorLabelBlacklist** to the **admissionConfig** section with the labels that are assigned to the node hosts you want to deny pod placement:

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiVersion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
  ...
```

3. Restart OpenShift Container Platform for the changes to take effect.

```
# systemctl restart atomic-openshift-master
```

NOTE

If you are using node selectors and **node affinity** in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

16.9. ADVANCED SCHEDULING AND TAINTS AND TOLERATIONS

16.9.1. Overview

Taints and tolerations allow the **node** to control which **pods** should (or should not) be scheduled on them.

16.9.2. Taints and Tolerations

A *taint* allows a node to refuse pod to be scheduled unless that pod has a matching *toleration*.

You apply taints to a node through the node specification (**NodeSpec**) and apply tolerations to a pod through the pod specification (**PodSpec**). A taint on a node instructs the node to repel all pods that do not tolerate the taint.

Taints and tolerations consist of a key, value, and effect. An operator allows you to leave one of these parameters empty.

Table 16.1. Taint and toleration components

Parameter	Description						
key	The key is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.						
value	The value is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.						
effect	<p>The effect is one of the following:</p> <table> <tr> <td>NoSchedule</td><td> <ul style="list-style-type: none"> New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain. </td></tr> <tr> <td>PreferNoSchedule</td><td> <ul style="list-style-type: none"> New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. Existing pods on the node remain. </td></tr> <tr> <td>NoExecute</td><td> <ul style="list-style-type: none"> New pods that do not match the taint cannot be scheduled onto that node. Existing pods on the node that do not have a matching toleration are removed. </td></tr> </table>	NoSchedule	<ul style="list-style-type: none"> New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain. 	PreferNoSchedule	<ul style="list-style-type: none"> New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. Existing pods on the node remain. 	NoExecute	<ul style="list-style-type: none"> New pods that do not match the taint cannot be scheduled onto that node. Existing pods on the node that do not have a matching toleration are removed.
NoSchedule	<ul style="list-style-type: none"> New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain. 						
PreferNoSchedule	<ul style="list-style-type: none"> New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. Existing pods on the node remain. 						
NoExecute	<ul style="list-style-type: none"> New pods that do not match the taint cannot be scheduled onto that node. Existing pods on the node that do not have a matching toleration are removed. 						

Parameter	Description	
operator	Equal	The key/value/effect parameters must match. This is the default.
	Exists	The key/effect parameters must match. You must leave a blank value parameter, which matches any.

A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:
 - the **key** parameters are the same;
 - the **value** parameters are the same;
 - the **effect** parameters are the same.
- If the **operator** parameter is set to **Exists**:
 - the **key** parameters are the same;
 - the **effect** parameters are the same.

16.9.2.1. Using Multiple Taints

You can put multiple taints on the same node and multiple tolerations on the same pod. OpenShift Container Platform processes multiple taints and tolerations as follows:

1. Process the taints for which the pod has a matching toleration.
2. The remaining unmatched taints have the indicated effects on the pod:
 - If there is at least one unmatched taint with effect **NoSchedule**, OpenShift Container Platform cannot schedule a pod onto that node.
 - If there is no unmatched taint with effect **NoSchedule** but there is at least one unmatched taint with effect **PreferNoSchedule**, OpenShift Container Platform tries to not schedule the pod onto the node.
 - If there is at least one unmatched taint with effect **NoExecute**, OpenShift Container Platform evicts the pod from the node (if it is already running on the node), or the pod is not scheduled onto the node (if it is not yet running on the node).
 - Pods that do not tolerate the taint are evicted immediately.
 - Pods that tolerate the taint without specifying **tolerationSeconds** in their toleration specification remain bound forever.

- Pods that tolerate the taint with a specified **tolerationSeconds** remain bound for the specified amount of time.

For example:

- The node has the following taints:

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
$ oc adm taint nodes node1 key1=value1:NoExecute
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- The pod has the following tolerations:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

In this case, the pod cannot be scheduled onto the node, because there is no toleration matching the third taint. The pod continues running if it is already running on the node when the taint is added, because the third taint is the only one of the three that is not tolerated by the pod.

16.9.3. Adding a Taint to an Existing Node

You add a taint to a node using the **oc adm taint** command with the parameters described in the [Taint and toleration components](#) table:

```
$ oc adm taint nodes <node-name> <key>=<value>:<effect>
```

For example:

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

The example places a taint on **node1** that has key **key1**, value **value1**, and taint effect **NoSchedule**.

16.9.4. Adding a Toleration to a Pod

To add a toleration to a pod, edit the pod specification to include a **tolerations** section:

Sample pod configuration file with Equal operator

```
tolerations:
- key: "key1" ①
  operator: "Equal" ②
  value: "value1" ③
  effect: "NoExecute" ④
  tolerationSeconds: 3600 ⑤
```

- 1 2 3 4** The toleration parameters, as described in the [Taint and toleration components](#) table.
- 5** The **tolerationSeconds** parameter specifies how long a pod can remain bound to a node before being evicted. See [Using Toleration Seconds to Delay Pod Evictions](#) below.

Sample pod configuration file with Exists operator

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

Both of these tolerations match the [taint created by the `oc adm taint` command above](#). A pod with either toleration would be able to schedule onto **node1**.

16.9.4.1. Using Toleration Seconds to Delay Pod Evictions

You can specify how long a pod can remain bound to a node before being evicted by specifying the **tolerationSeconds** parameter in the pod specification. If a taint with the **NoExecute** effect is added to a node, any pods that do not tolerate the taint are evicted immediately (pods that do tolerate the taint are not evicted). However, if a pod that to be evicted has the **tolerationSeconds** parameter, the pod is not evicted until that time period expires.

For example:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

Here, if this pod is running but does not have a matching taint, the pod stays bound to the node for 3,600 seconds and then be evicted. If the taint is removed before that time, the pod is not evicted.

16.9.4.1.1. Setting a Default Value for Toleration Seconds

This plug-in sets the default forgiveness toleration for pods, to tolerate the **node.alpha.kubernetes.io/not-ready:NoExecute** and **node.alpha.kubernetes.io/unreachable:NoExecute** taints for five minutes.

If the pod configuration provided by the user already has either toleration, the default is not added.

To enable Default Toleration Seconds:

1. Modify the master configuration file (*/etc/origin/master/master-config.yaml*) to Add **DefaultTolerationSeconds** to the admissionConfig section:

```
admissionConfig:
  pluginConfig:
    DefaultTolerationSeconds:
      configuration:
```



```
kind: DefaultAdmissionConfig
apiVersion: v1
disable: false
```

2. Restart OpenShift for the changes to take effect:

```
# systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

3. Verify that the default was added:

- a. Create a pod:

```
$ oc create -f </path/to/file>
```

For example:

```
$ oc create -f hello-pod.yaml
pod "hello-pod" created
```

- b. Check the pod tolerations:

```
$ oc describe pod <pod-name> |grep -i toleration
```

For example:

```
$ oc describe pod hello-pod |grep -i toleration
Tolerations:      node.alpha.kubernetes.io/not-
ready=:Exists:NoExecute for 300s
```

16.9.5. Preventing Pod Eviction for Node Problems

OpenShift Container Platform can be configured to represent **node unreachable** and **node not ready** conditions as taints. This allows per-pod specification of how long to remain bound to a node that becomes unreachable or not ready, rather than using the default of five minutes.

When the Taint Based Evictions feature is enabled, the taints are automatically added by the node controller and the normal logic for evicting pods from **Ready** nodes is disabled.

- If a node enters a not ready state, the **node.alpha.kubernetes.io/not-ready:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.
- If a node enters a not reachable state, the **node.alpha.kubernetes.io/unreachable:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.

To enable Taint Based Evictions:

1. Modify the master configuration file (*/etc/origin/master/master-config.yaml*) to add the following to the **kubernetesMasterConfig** section:

```
kubernetesMasterConfig:
  controllerArguments:
```

```
feature-gates:
- "TaintBasedEvictions=true"
```

2. Check that the taint is added to a node:

```
oc describe node $node | grep -i taint

Taints: node.alpha.kubernetes.io/not-ready:NoExecute
```

3. Restart OpenShift for the changes to take effect:

```
# systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

4. Add a toleration to pods:

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

or

```
tolerations:
- key: "node.alpha.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```



NOTE

To maintain the existing [rate limiting](#) behavior of pod evictions due to node problems, the system adds the taints in a rate-limited way. This prevents massive pod evictions in scenarios such as the master becoming partitioned from the nodes.

16.9.6. Daemonsets and Tolerations

[DaemonSet](#) pods are created with **NoExecute** tolerations for **node.alpha.kubernetes.io/unreachable** and **node.alpha.kubernetes.io/not-ready** with no **tolerationSeconds** to ensure that DaemonSet pods are never evicted due to these problems, even when the Default Toleration Seconds feature is disabled.

16.9.7. Examples

Taints and tolerations are a flexible way to steer pods away from nodes or evict pods that should not be running on a node. A few of typical scenarios are:

- [Dedicating a node for a user](#)
- [Binding a user to a node](#)
- [Dedicating nodes with special hardware](#)

16.9.7.1. Dedicating a Node for a User

You can specify a set of nodes for exclusive use by a particular set of users.

To specify dedicated nodes:

1. Add a taint to those nodes:

For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom [admission controller](#).
Only the pods with the tolerations are allowed to use the dedicated nodes.

16.9.7.2. Binding a User to a Node

You can configure a node so that particular users can use only the dedicated nodes.

To configure a node so that users can use only that node:

1. Add a taint to those nodes:

For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom [admission controller](#).
The admission controller should add a node affinity to require that the pods can only schedule onto nodes labeled with the **key:value** label (**dedicated=groupName**).
3. Add a label similar to the taint (such as the **key:value** label) to the dedicated nodes.

16.9.7.3. Nodes with Special Hardware

In a cluster where a small subset of nodes have specialized hardware (for example GPUs), you can use taints and tolerations to keep pods that do not need the specialized hardware off of those nodes, leaving the nodes for pods that do need the specialized hardware. You can also require pods that need specialized hardware to use specific nodes.

To ensure pods are blocked from the specialized hardware:

1. Taint the nodes that have the specialized hardware using one of the following commands:

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

2. Adding a corresponding toleration to pods that use the special hardware using an [admission controller](#).

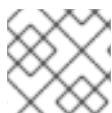
For example, the admission controller could use some characteristic(s) of the pod to determine that the pod should be allowed to use the special nodes by adding a toleration.

To ensure pods can only use the specialized hardware, you need some additional mechanism. For example, you could label the nodes that have the special hardware and use node affinity on the pods that need the hardware.

CHAPTER 17. SETTING QUOTAS

17.1. OVERVIEW

A resource quota, defined by a **ResourceQuota** object, provides constraints that limit aggregate resource consumption per project. It can limit the quantity of objects that can be created in a project by type, as well as the total amount of compute resources and storage that may be consumed by resources in that project.



NOTE

See the [Developer Guide](#) for more on compute resources.

17.2. RESOURCES MANAGED BY QUOTA

The following describes the set of compute resources and object types that may be managed by a quota.



NOTE

A pod is in a terminal state if **status.phase in (Failed, Succeeded)** is true.

Table 17.1. Compute Resources Managed by Quota

Resource Name	Description
cpu	The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. cpu and requests.cpu are the same value and can be used interchangeably.
memory	The sum of memory requests across all pods in a non-terminal state cannot exceed this value. memory and requests.memory are the same value and can be used interchangeably.
requests.cpu	The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. cpu and requests.cpu are the same value and can be used interchangeably.
requests.memory	The sum of memory requests across all pods in a non-terminal state cannot exceed this value. memory and requests.memory are the same value and can be used interchangeably.
limits.cpu	The sum of CPU limits across all pods in a non-terminal state cannot exceed this value.
limits.memory	The sum of memory limits across all pods in a non-terminal state cannot exceed this value.

Table 17.2. Storage Resources Managed by Quota

Resource Name	Description
requests.storage	The sum of storage requests across all persistent volume claims in any state cannot exceed this value.
persistentvolumeclaims	The total number of persistent volume claims that can exist in the project.
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	The sum of storage requests across all persistent volume claims in any state that have a matching storage class, cannot exceed this value.
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	The total number of persistent volume claims with a matching storage class that can exist in the project.

Table 17.3. Object Counts Managed by Quota

Resource Name	Description
pods	The total number of pods in a non-terminal state that can exist in the project.
replicationcontrollers	The total number of replication controllers that can exist in the project.
resourcequotas	The total number of resource quotas that can exist in the project.
services	The total number of services that can exist in the project.
secrets	The total number of secrets that can exist in the project.
configmaps	The total number of ConfigMap objects that can exist in the project.
persistentvolumeclaims	The total number of persistent volume claims that can exist in the project.
openshift.io/imagestreams	The total number of image streams that can exist in the project.

17.3. QUOTA SCOPES

Each quota can have an associated set of *scopes*. A quota will only measure usage for a resource if it matches the intersection of enumerated scopes.

Adding a scope to a quota restricts the set of resources to which that quota can apply. Specifying a resource outside of the allowed set results in a validation error.

Scope	Description
Terminating	Match pods where <code>spec.activeDeadlineSeconds >= 0</code> .
NotTerminating	Match pods where <code>spec.activeDeadlineSeconds</code> is <code>nil</code> .
BestEffort	Match pods that have best effort quality of service for either cpu or memory . See the Quality of Service Classes for more on committing compute resources.
NotBestEffort	Match pods that do not have best effort quality of service for cpu and memory .

A **BestEffort** scope restricts a quota to limiting the following resources:

- **pods**

A **Terminating**, **NotTerminating**, and **NotBestEffort** scope restricts a quota to tracking the following resources:

- **pods**
- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**
- **limits.cpu**

17.4. QUOTA ENFORCEMENT

After a resource quota for a project is first created, the project restricts the ability to create any new resources that may violate a quota constraint until it has calculated updated usage statistics.

After a quota is created and usage statistics are updated, the project accepts the creation of new content. When you create or modify resources, your quota usage is incremented immediately upon the request to create or modify the resource.

When you delete a resource, your quota use is decremented during the next full recalculation of quota statistics for the project. A configurable amount of time determines how long it takes to reduce quota usage statistics to their current observed system value.

If project modifications exceed a quota usage limit, the server denies the action, and an appropriate error message is returned to the user explaining the quota constraint violated, and what their currently observed usage stats are in the system.

17.5. REQUESTS VERSUS LIMITS

When allocating [compute resources](#), each container may specify a request and a limit value each for CPU and memory. Quotas can restrict any of these values.

If the quota has a value specified for **requests.cpu** or **requests.memory**, then it requires that every incoming container make an explicit request for those resources. If the quota has a value specified for **limits.cpu** or **limits.memory**, then it requires that every incoming container specify an explicit limit for those resources.

17.6. SAMPLE RESOURCE QUOTA DEFINITIONS

core-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" 1
    persistentvolumeclaims: "4" 2
    replicationcontrollers: "20" 3
    secrets: "10" 4
    services: "10" 5
```

- 1 The total number of **ConfigMap** objects that can exist in the project.
- 2 The total number of persistent volume claims (PVCs) that can exist in the project.
- 3 The total number of replication controllers that can exist in the project.
- 4 The total number of secrets that can exist in the project.
- 5 The total number of services that can exist in the project.

openshift-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" 1
```

- 1 The total number of image streams that can exist in the project.

compute-resources.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
```

```
spec:
  hard:
    pods: "4" ❶
    requests.cpu: "1" ❷
    requests.memory: 1Gi ❸
    limits.cpu: "2" ❹
    limits.memory: 2Gi ❺
```

- ❶ The total number of pods in a non-terminal state that can exist in the project.
- ❷ Across all pods in a non-terminal state, the sum of CPU requests cannot exceed 1 core.
- ❸ Across all pods in a non-terminal state, the sum of memory requests cannot exceed 1Gi.
- ❹ Across all pods in a non-terminal state, the sum of CPU limits cannot exceed 2 cores.
- ❺ Across all pods in a non-terminal state, the sum of memory limits cannot exceed 2Gi.

besteffort.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
  hard:
    pods: "1" ❶
  scopes:
    - BestEffort ❷
```

- ❶ The total number of pods in a non-terminal state with **BestEffort** quality of service that can exist in the project.
- ❷ Restricts the quota to only matching pods that have **BestEffort** quality of service for either memory or CPU.

compute-resources-long-running.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" ❶
    limits.cpu: "4" ❷
    limits.memory: "2Gi" ❸
  scopes:
    - NotTerminating ❹
```

- ❶ The total number of pods in a non-terminal state.

- 2 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- 3 Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
- 4 Restricts the quota to only matching pods where **spec.activeDeadlineSeconds** is set to **nil**. Build pods will fall under **NotTerminating** unless the **RestartNever** policy is applied.

compute-resources-time-bound.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" 1
    limits.cpu: "1" 2
    limits.memory: "1Gi" 3
  scopes:
    - Terminating 4

```

- 1 The total number of pods in a non-terminal state.
- 2 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- 3 Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
- 4 Restricts the quota to only matching pods where **spec.activeDeadlineSeconds** ≥ 0 . For example, this quota would charge for build or deployer pods, but not long running pods like a web server or database.

storage-consumption.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" 1
    requests.storage: "50Gi" 2
    gold.storageclass.storage.k8s.io/requests.storage: "10Gi" 3
    silver.storageclass.storage.k8s.io/requests.storage: "20Gi" 4
    silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" 5
    bronze.storageclass.storage.k8s.io/requests.storage: "0" 6
    bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" 7

```

- 1 The total number of persistent volume claims in a project
- 2 Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.

- 3 Across all persistent volume claims in a project, the sum of storage requested in the gold storage class cannot exceed this value.
- 4 Across all persistent volume claims in a project, the sum of storage requested in the silver storage class cannot exceed this value.
- 5 Across all persistent volume claims in a project, the total number of claims in the silver storage class cannot exceed this value.
- 6 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot request storage.
- 7 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot create claims.

17.7. CREATING A QUOTA

To create a quota, first define the quota to your specifications in a file, for example as seen in [Sample Resource Quota Definitions](#). Then, create using that file to apply it to a project:

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

For example:

```
$ oc create -f resource-quota.json -n demoproject
```

17.8. VIEWING A QUOTA

You can view usage statistics related to any hard limits defined in a project's quota by navigating in the web console to the project's **Quota** page.

You can also use the CLI to view quota details:

1. First, get the list of quotas defined in the project. For example, for a project called **demoproject**:

```
$ oc get quota -n demoproject
NAME                AGE
besteffort          11m
compute-resources   2m
core-object-counts  29m
```

2. Then, describe the quota you are interested in, for example the **core-object-counts** quota:

```
$ oc describe quota core-object-counts -n demoproject
Name:      core-object-counts
Namespace: demoproject
Resource   Used Hard
-----
configmaps 3 10
persistentvolumeclaims 0 4
```

```
replicationcontrollers 3 20
secrets      9 10
services     2 10
```

17.9. CONFIGURING QUOTA SYNCHRONIZATION PERIOD

When a set of resources are deleted, the synchronization time frame of resources is determined by the **resource-quota-sync-period** setting in the `/etc/origin/master/master-config.yaml` file.

Before quota usage is restored, a user may encounter problems when attempting to reuse the resources. You can change the **resource-quota-sync-period** setting to have the set of resources regenerate at the desired amount of time (in seconds) and for the resources to be available again:

```
kubernetesMasterConfig:
  apiLevels:
    - v1beta3
    - v1
  apiServerArguments: null
  controllerArguments:
    resource-quota-sync-period:
      - "10s"
```

After making any changes, restart the master services to apply them.

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-
controllers
```

Adjusting the regeneration time can be helpful for creating resources and determining resource usage when automation is used.



NOTE

The **resource-quota-sync-period** setting is designed to balance system performance. Reducing the sync period can result in a heavy load on the master.

17.10. ACCOUNTING FOR QUOTA IN DEPLOYMENT CONFIGURATIONS

If a quota has been defined for your project, see [Deployment Resources](#) for considerations on any deployment configurations.

17.11. REQUIRE EXPLICIT QUOTA TO CONSUME A RESOURCE



NOTE

This feature is tech preview and subject to change in future releases.

If a resource is not managed by quota, a user has no restriction on the amount of resource that can be consumed. For example, if there is no quota on storage related to the gold storage class, the amount of gold storage a project can create is unbounded.

For high-cost compute or storage resources, administrators may want to require an explicit quota be granted in order to consume a resource. For example, if a project was not explicitly given quota for

storage related to the gold storage class, users of that project would not be able to create any storage of that type.

In order to require explicit quota to consume a particular resource, the following stanza should be added to the master-config.yaml.

```
admissionConfig:
  pluginConfig:
    ResourceQuota:
      configuration:
        apiVersion: resourcequota.admission.k8s.io/v1alpha1
        kind: Configuration
        limitedResources:
          - resource: persistentvolumeclaims 1
        matchContains:
          - gold.storageclass.storage.k8s.io/requests.storage 2
```

- 1** The group/resource to whose consumption is limited by default.
- 2** The name of the resource tracked by quota associated with the group/resource to limit by default.

In the above example, the quota system will intercept every operation that creates or updates a **PersistentVolumeClaim**. It checks what resources understood by quota would be consumed, and if there is no covering quota for those resources in the project, the request is denied. In this example, if a user creates a **PersistentVolumeClaim** that uses storage associated with the gold storage class, and there is no matching quota in the project, the request is denied.

17.12. KNOWN ISSUES

- Invalid objects can cause quota resources for a project to become exhausted. Quota is incremented in admission prior to validation of the resource. As a result, quota can be incremented even if the pod is not ultimately persisted. This will be resolved in a future release. ([BZ1485375](#))

CHAPTER 18. SETTING MULTI-PROJECT QUOTAS

18.1. OVERVIEW

A multi-project quota, defined by a **ClusterResourceQuota** object, allows [quotas](#) to be shared across multiple projects. Resources used in each selected project will be aggregated and that aggregate will be used to limit resources across all the selected projects.

18.2. SELECTING PROJECTS

You can select projects based on annotation selection, label selection, or both. For example, to select projects based on annotations, run the following command:

```
$ oc create clusterquota for-user \
  --project-annotation-selector openshift.io/requester=<user-name> \
  --hard pods=10 \
  --hard secrets=20
```

It creates the following **ClusterResourceQuota** object:

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: 1
  hard:
    pods: "10"
    secrets: "20"
  selector:
    annotations: 2
    openshift.io/requester: <user-name>
    labels: null 3
status:
  namespaces: 4
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: 5
  hard:
    pods: "10"
    secrets: "20"
  used:
    pods: "1"
    secrets: "9"
```

1 The **ResourceQuotaSpec** object that will be enforced over the selected projects.

- 2 A simple key/value selector for annotations.
- 3 A label selector that can be used to select projects.
- 4 A per-namespace map that describes current quota usage in each selected project.
- 5 The aggregate usage across all selected projects.

This multi-project quota document controls all projects requested by **<user-name>** using the default project request endpoint. You are limited to 10 pods and 20 secrets.

Similarly, to select projects based on labels, run this command:

```
$ oc create clusterresourcequota for-name \ 1
  --project-label-selector=name=frontend \ 2
  --hard=pods=10 --hard=secrets=20
```

- 1 Both **clusterresourcequota** and **clusterquota** are aliases of the same command. **for-name** is the name of the **clusterresourcequota** object.
- 2 To select projects by label, provide a key-value pair by using the format **--project-label-selector=key=value**.

It creates the following **ClusterResourceQuota** object definition:

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend
```

18.3. VIEWING APPLICABLE CLUSTERRESOURCEQUOTAS

A project administrator is not allowed to create or modify the multi-project quota that limits his or her project, but the administrator is allowed to view the multi-project quota documents that are applied to his or her project. The project administrator can do this via the **AppliedClusterResourceQuota** resource.

```
$ oc describe AppliedClusterResourceQuota
```

produces:

```
Name:    for-user
Namespace:  <none>
Created:   19 hours ago
Labels:    <none>
Annotations:  <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource   Used   Hard
-----
pods        1    10
secrets     9    20
```

18.4. SELECTION GRANULARITY

Because of the locking consideration when claiming quota allocations, the number of active projects selected by a multi-project quota is an important consideration. Selecting more than 100 projects under a single multi-project quota may have detrimental effects on API server responsiveness in those projects.

CHAPTER 19. SETTING LIMIT RANGES

19.1. OVERVIEW

A limit range, defined by a **LimitRange** object, enumerates [compute resource constraints](#) in a [project](#) at the pod, container, image, image stream, and persistent volume claim level, and specifies the amount of resources that a pod, container, image, image stream, or persistent volume claim can consume.

All resource create and modification requests are evaluated against each **LimitRange** object in the project. If the resource violates any of the enumerated constraints, then the resource is rejected. If the resource does not set an explicit value, and if the constraint supports a default value, then the default value is applied to the resource.

Core Limit Range Object Definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits" ❶
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" ❷
        memory: "1Gi" ❸
      min:
        cpu: "200m" ❹
        memory: "6Mi" ❺
    - type: "Container"
      max:
        cpu: "2" ❻
        memory: "1Gi" ❼
      min:
        cpu: "100m" ❽
        memory: "4Mi" ❾
      default:
        cpu: "300m" ❿
        memory: "200Mi" ❾❶
      defaultRequest:
        cpu: "200m" ❿❷
        memory: "100Mi" ❿❸
      maxLimitRequestRatio:
        cpu: "10" ❿❹
```

- ❶ The name of the limit range object.
- ❷ The maximum amount of CPU that a pod can request on a node across all containers.
- ❸ The maximum amount of memory that a pod can request on a node across all containers.
- ❹ The minimum amount of CPU that a pod can request on a node across all containers.
- ❺ The minimum amount of memory that a pod can request on a node across all containers.

- 6 The maximum amount of CPU that a single container in a pod can request.
- 7 The maximum amount of memory that a single container in a pod can request.
- 8 The minimum amount of CPU that a single container in a pod can request.
- 9 The minimum amount of memory that a single container in a pod can request.
- 10 The default amount of CPU that a container will be limited to use if not specified.
- 11 The default amount of memory that a container will be limited to use if not specified.
- 12 The default amount of CPU that a container will request to use if not specified.
- 13 The default amount of memory that a container will request to use if not specified.
- 14 The maximum amount of CPU burst that a container can make as a ratio of its limit over request.

For more information on how CPU and memory are measured, see [Compute Resources](#).

OpenShift Container Platform Limit Range Object Definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "openshift-resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi 1
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 2
        openshift.io/images: 30 3
```

- 1 The maximum size of an image that can be pushed to an internal registry.
- 2 The maximum number of unique image tags per image stream's spec.
- 3 The maximum number of unique image references per image stream's status.

Both core and OpenShift Container Platform resources can be specified in just one limit range object. They are separated here into two examples for clarity.

19.1.1. Container Limits

Supported Resources:

- CPU
- Memory

Supported Constraints:

Per container, the following must hold true if specified:

Table 19.1. Container

Constraint	Behavior
Min	<p>Min[resource] less than or equal to container.resources.requests[resource] (required) less than or equal to container/resources.limits[resource] (optional)</p> <p>If the configuration defines a min CPU, then the request value must be greater than the CPU value. A limit value does not need to be specified.</p>
Max	<p>container.resources.limits[resource] (required) less than or equal to Max[resource]</p> <p>If the configuration defines a max CPU, then you do not need to define a request value, but a limit value does need to be set that satisfies the maximum CPU constraint.</p>
MaxLimitRequestRatio	<p>MaxLimitRequestRatio[resource] less than or equal to (container.resources.limits[resource] / container.resources.requests[resource])</p> <p>If a configuration defines a maxLimitRequestRatio value, then any new containers must have both a request and limit value. Additionally, OpenShift Container Platform calculates a limit to request ratio by dividing the limit by the request. This value should be a non-negative integer greater than 1.</p> <p>For example, if a container has cpu: 500 in the limit value, and cpu: 100 in the request value, then its limit to request ratio for cpu is 5. This ratio must be less than or equal to the maxLimitRequestRatio.</p>

Supported Defaults:

Default[resource]

Defaults **container.resources.limit[resource]** to specified value if none.

Default Requests[resource]

Defaults **container.resources.requests[resource]** to specified value if none.

19.1.2. Pod Limits

Supported Resources:

- CPU
- Memory

Supported Constraints:

Across all containers in a pod, the following must hold true:

Table 19.2. Pod

Constraint	Enforced Behavior
Min	Min[resource] less than or equal to container.resources.requests[resource] (required) less than or equal to container.resources.limits[resource] (optional)
Max	container.resources.limits[resource] (required) less than or equal to Max[resource]
MaxLimitRequestRatio	MaxLimitRequestRatio[resource] less than or equal to (container.resources.limits[resource] / container.resources.requests[resource])

19.1.3. Image Limits

Supported Resources:

- Storage

Resource type name:

- **openshift.io/Image**

Per image, the following must hold true if specified:

Table 19.3. Image

Constraint	Behavior
Max	image.dockerimagemetadata.size less than or equal to Max[resource]



NOTE

To prevent blobs exceeding the limit from being uploaded to the registry, the registry must be configured to enforce quota. An environment variable **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA** must be set to **true** which is done by default for new deployments. To update older deployment configuration, refer to [Enforcing quota in the Registry](#).

**WARNING**

The image size is not always available in the manifest of an uploaded image. This is especially the case for images built with Docker 1.10 or higher and pushed to a v2 registry. If such an image is pulled with an older Docker daemon, the image manifest will be converted by the registry to schema v1 lacking all the size information. No storage limit set on images will prevent it from being uploaded.

[The issue](#) is being addressed.

19.1.4. Image Stream Limits

Supported Resources:

- `openshift.io/image-tags`
- `openshift.io/images`

Resource type name:

- `openshift.io/ImageStream`

Per image stream, the following must hold true if specified:

Table 19.4. ImageStream

Constraint	Behavior
<code>Max[openshift.io/image-tags]</code>	<p><code>length(uniqueimagetags(imagestream.spec.tags))</code> less than or equal to <code>Max[openshift.io/image-tags]</code></p> <p><code>uniqueimagetags</code> returns unique references to images of given spec tags.</p>
<code>Max[openshift.io/images]</code>	<p><code>length(uniqueimages(imagestream.status.tags))</code> less than or equal to <code>Max[openshift.io/images]</code></p> <p><code>uniqueimages</code> returns unique image names found in status tags. The name equals image's digest.</p>

19.1.4.1. Counting of Image References

Resource `openshift.io/image-tags` represents unique [image references](#). Possible references are an `ImageStreamTag`, an `ImageStreamImage` and a `DockerImage`. They may be created using commands `oc tag` and `oc import-image` or by using [tag tracking](#). No distinction is made between internal and external references. However, each unique reference tagged in the image stream's specification is counted just once. It does not restrict pushes to an internal container registry in any way, but is useful for tag restriction.

Resource `openshift.io/images` represents unique image names recorded in image stream status. It allows for restriction of a number of images that can be pushed to the internal registry. Internal and external references are not distinguished.

19.1.5. PersistentVolumeClaim Limits

Supported Resources:

- Storage

Supported Constraints:

Across all persistent volume claims in a project, the following must hold true:

Table 19.5. Pod

Constraint	Enforced Behavior
Min	$\text{Min}[\text{resource}] \Leftarrow \text{claim.spec.resources.requests}[\text{resource}]$ (required)
Max	$\text{claim.spec.resources.requests}[\text{resource}]$ (required) $\Leftarrow \text{Max}[\text{resource}]$

Limit Range Object Definition

```
{
  "apiVersion": "v1",
  "kind": "LimitRange",
  "metadata": {
    "name": "pvcs" ❶
  },
  "spec": {
    "limits": [{
      "type": "PersistentVolumeClaim",
      "min": {
        "storage": "2Gi" ❷
      },
      "max": {
        "storage": "50Gi" ❸
      }
    }]
  }
}
```

❶ The name of the limit range object.

❷ The minimum amount of storage that can be requested in a persistent volume claim

❸ The maximum amount of storage that can be requested in a persistent volume claim

19.2. CREATING A LIMIT RANGE

CHAPTER 20. PRUNING OBJECTS

20.1. OVERVIEW

Over time, [API objects](#) created in OpenShift Container Platform can accumulate in the [etcd data store](#) through normal user operations, such as when building and deploying applications.

As an administrator, you can periodically prune older versions of objects from your OpenShift Container Platform instance that are no longer needed. For example, by pruning images you can delete older images and layers that are no longer in use, but are still taking up disk space.

20.2. BASIC PRUNE OPERATIONS

The CLI groups prune operations under a common parent command.

```
$ oc adm prune <object_type> <options>
```

This specifies:

- The **<object_type>** to perform the action on, such as **builds**, **deployments**, or **images**.
- The **<options>** supported to prune that object type.

20.3. PRUNING DEPLOYMENTS

In order to prune deployments that are no longer required by the system due to age and status, administrators may run the following command:

```
$ oc adm prune deployments [<options>]
```

Table 20.1. Prune Deployments CLI Configuration Options

Option	Description
--confirm	Indicate that pruning should occur, instead of performing a dry-run.
--orphans	Prune all deployments whose deployment config no longer exists, status is complete or failed, and replica count is zero.
--keep-complete=<N>	Per deployment config, keep the last N deployments whose status is complete and replica count is zero. (default 5)
--keep-failed=<N>	Per deployment config, keep the last N deployments whose status is failed and replica count is zero. (default 1)
--keep-younger-than=<duration>	Do not prune any object that is younger than <duration> relative to the current time. (default 60m) Valid units of measurement include nanoseconds (ns), microseconds (us), milliseconds (ms), seconds (s), minutes (m), and hours (h).

To see what a pruning operation would delete:

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m
```

To actually perform the prune operation:

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m --confirm
```

20.4. PRUNING BUILDS

In order to prune builds that are no longer required by the system due to age and status, administrators may run the following command:

```
$ oc adm prune builds [<options>]
```

Table 20.2. Prune Builds CLI Configuration Options

Option	Description
--confirm	Indicate that pruning should occur, instead of performing a dry-run.
--orphans	Prune all builds whose build config no longer exists, status is complete, failed, error, or canceled.
--keep-complete=<N>	Per build config, keep the last N builds whose status is complete. (default 5)
--keep-failed=<N>	Per build config, keep the last N builds whose status is failed, error, or canceled (default 1)
--keep-younger-than=<duration>	Do not prune any object that is younger than <duration> relative to the current time. (default 60m)

To see what a pruning operation would delete:

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m
```

To actually perform the prune operation:

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m --confirm
```



NOTE

Developers can enable [automatic build pruning](#) by modifying their build configuration.

20.5. PRUNING IMAGES

In order to prune images that are no longer required by the system due to age, status, or exceed limits, administrators may run the following command:

```
$ oc adm prune images [<options>]
```



NOTE

Currently, to prune images you must first [log in to the CLI](#) as a user with an [access token](#). The user must also have the [cluster role](#) `system:image-pruner` or greater (for example, `cluster-admin`).



NOTE

Pruning images removes data from the integrated registry. For this operation to work properly, ensure your [registry is configured](#) with `storage:delete:enabled` set to `true`.



NOTE

Pruning images with the `--namespace` flag does not remove images, only image streams. Images are non-namespaced resources. Therefore, limiting pruning to a particular namespace makes it impossible to calculate their current usage.

Table 20.3. Prune Images CLI Configuration Options

Option	Description
<code>--all</code>	Include images that were not pushed to the registry, but have been mirrored by pullthrough. This is on by default. To limit the pruning to images that were pushed to the integrated registry, pass <code>--all=false</code> .
<code>--certificate-authority</code>	The path to a certificate authority file to use when communicating with the OpenShift Container Platform-managed registries. Defaults to the certificate authority data from the current user's configuration file. If provided, secure connection will be initiated.
<code>--confirm</code>	Indicate that pruning should occur, instead of performing a dry-run. This requires a valid route to the integrated Docker registry. If this command is run outside of the cluster network, the route needs to be provided using <code>--registry-url</code> .
<code>--force-insecure</code>	Use caution with this option. Allow an insecure connection to the Docker registry that is hosted via HTTP or has an invalid HTTPS certificate. See Using Secure or Insecure Connections for more information.
<code>--keep-tag-revisions=<N></code>	For each image stream, keep up to at most N image revisions per tag. (default <code>3</code>)

Option	Description
--keep-younger-than=<duration>	Do not prune any image that is younger than <duration> relative to the current time. Do not prune any image that is referenced by any other object that is younger than <duration> relative to the current time. (default 60m)
--prune-over-size-limit	Prune each image that exceeds the smallest limit defined in the same project. This flag cannot be combined with --keep-tag-revisions nor --keep-younger-than .
--registry-url	The address to use when contacting the registry. The command will attempt to use a cluster-internal URL determined from managed images and image streams. In case it fails (the registry cannot be resolved or reached), an alternative route that works needs to be provided using this flag. The registry host name may be prefixed by https:// or http:// which will enforce particular connection protocol.

20.5.1. Image Prune Conditions

- Remove any image "managed by OpenShift Container Platform" (images with the annotation **openshift.io/image.managed**) that was created at least **--keep-younger-than** minutes ago and is not currently referenced by:
 - any pod created less than **--keep-younger-than** minutes ago.
 - any image stream created less than **--keep-younger-than** minutes ago.
 - any running pods.
 - any pending pods.
 - any replication controllers.
 - any deployment configurations.
 - any build configurations.
 - any builds.
 - the **--keep-tag-revisions** most recent items in **stream.status.tags[].items**.
- Remove any image "managed by OpenShift Container Platform" (images with the annotation **openshift.io/image.managed**) that is exceeding the smallest [limit](#) defined in the same project and is not currently referenced by:
 - any running pods.
 - any pending pods.
 - any replication controllers.
 - any deployment configurations.

- any build configurations.
- any builds.
- There is no support for pruning from external registries.
- When an image is pruned, all references to the image are removed from all image streams that have a reference to the image in **status.tags**.
- Image layers that are no longer referenced by any images are removed as well.



NOTE

--prune-over-size-limit cannot be combined with **--keep-tag-revisions** nor **--keep-younger-than** flags. Doing so will return an information that this operation is not allowed.

To see what a pruning operation would delete:

1. Keeping up to three tag revisions, and keeping resources (images, image streams and pods) younger than sixty minutes:

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

2. Pruning every image that exceeds defined limits:

```
$ oc adm prune images --prune-over-size-limit
```

To actually perform the prune operation for the previously mentioned options accordingly:

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm
```

```
$ oc adm prune images --prune-over-size-limit --confirm
```

20.5.2. Using Secure or Insecure Connections

The secure connection is the preferred and recommended approach. It is done over HTTPS protocol with a mandatory certificate verification. The **prune** command always attempts to use it if possible. If not possible, in some cases it can fall-back to insecure connection, which is dangerous. In this case, either certificate verification is skipped or plain HTTP protocol is used.

The fall-back to insecure connection is allowed in the following cases unless **--certificate-authority** is specified:

1. The **prune** command is run with the **--force-insecure** option.
2. The provided **registry-url** is prefixed with the **http://** scheme.
3. The provided **registry-url** is a local-link address or localhost.
4. The configuration of the current user allows for an insecure connection. This may be caused by the user either logging in using **--insecure-skip-tls-verify** or choosing the insecure connection when prompted.



IMPORTANT

If the registry is secured by a certificate authority different from the one used by OpenShift Container Platform, it needs to be specified using the **--certificate-authority** flag. Otherwise, the **prune** command will fail with an error similar to those listed in [Using the Wrong Certificate Authority](#) or [Using an Insecure Connection Against a Secured Registry](#).

20.5.3. Image Pruning Problems

Images Not Being Pruned

If your images keep accumulating and the **prune** command removes just a small portion of what you expect, ensure that you understand [the conditions](#) that must apply for an image to be considered a candidate for pruning.

Especially ensure that images you want removed occur at higher positions in each [tag history](#) than your chosen tag revisions threshold. For example, consider an old and obsolete image named **sha:abz**. By running the following command in namespace **N**, where the image is tagged, you will see the image is tagged three times in a single image stream named **myapp**:

```
$ image_name="sha:abz"
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range $ti,
$tag := $is.status.tags}}'\
  '{{range $ii, $item := $tag.items}}{{if eq $item.image
  ""$image_name}}\
    ${{is.metadata.name}}:{{tag.tag}} at position {{$ii}} out of {{len
$tag.items}}\n'\
  '{{end}}{{end}}{{end}}{{end}}'
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

When default options are used, the image will not ever be pruned because it occurs at position **0** in a history of **myapp:v2.1-may-2016** tag. For an image to be considered for pruning, the administrator must either:

1. Specify **--keep-tag-revisions=0** with the **oc adm prune images** command.

CAUTION

This action will effectively remove all the tags from all the namespaces with underlying images, unless they are younger or they are referenced by objects younger than the specified threshold.

2. Delete all the [istags](#) where the position is below the revision threshold, which means **myapp:v2.1** and **myapp:v2.1-may-2016**.
3. Move the image further in the history, either by running new builds pushing to the same *istag*, or by tagging other image. Unfortunately, this is not always desirable for old release tags.

Tags having a date or time of a particular image's build in their names should be avoided, unless the image needs to be preserved for undefined amount of time. Such tags tend to have just one image in its history, which effectively prevents them from ever being pruned. [Learn more about istag naming](#).

Using a Secure Connection Against Insecure Registry

If you see a message similar to the following in the output of the **oc adm prune images**, then your registry is not secured and the **oc adm prune images** client will attempt to use secure connection:

```
error: error communicating with registry: Get
https://172.30.30.30:5000/healthz: http: server gave HTTP response to
HTTPS client
```

1. The recommended solution is to [secure the registry](#). If that is not desired, you can force the client to use an insecure connection by appending **--force-insecure** to the command (**not recommended**).

20.5.3.1. Using an Insecure Connection Against a Secured Registry

If you see one of the following errors in the output of the **oc adm prune images** command, it means that your registry is secured using a certificate signed by a certificate authority other than the one used by **oc adm prune images** client for connection verification.

```
error: error communicating with registry: Get
http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get
https://172.30.30.30:5000/healthz: x509: certificate signed by unknown
authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"]
```

By default, the certificate authority data stored in user's configuration file are used — the same for communication with the master API.

Use the **--certificate-authority** option to provide the right certificate authority for the Docker registry server.

Using the Wrong Certificate Authority

The following error means that the certificate authority used to sign the certificate of the secured Docker registry is different than the authority used by the client.

```
error: error communicating with registry: Get https://172.30.30.30:5000/:
x509: certificate signed by unknown authority
```

Make sure to provide the right one with the flag **--certificate-authority**.

As a work-around, the **--force-insecure** flag can be added instead (**not recommended**).

20.6. HARD PRUNING THE REGISTRY

The OpenShift Container Registry can accumulate blobs that are not referenced by the OpenShift Container Platform cluster's etcd. The basic Pruning Images procedure, therefore, is unable to operate on them. These are called *orphaned blobs*.

Orphaned blobs can occur from the following scenarios:

- Manually deleting an image with **oc delete image <sha256:image-id>** command, which only removes the image from etcd, but not from the registry's storage.

- Pushing to the registry initiated by **docker** daemon failures, which causes some blobs to get uploaded, but the image manifest (which is uploaded as the very last component) does not. All unique image blobs become orphans.
- OpenShift Container Platform refusing an image because of quota restrictions.
- The standard image pruner deleting an image manifest, but is interrupted before it deletes the related blobs.
- A bug in the registry pruner, which fails to remove the intended blobs, causing the image objects referencing them to be removed and the blobs becoming orphans.

Hard pruning the registry, a separate procedure from basic image pruning, allows you to remove orphaned blobs. You should hard prune if you are running out of storage space in your OpenShift Container Registry and believe you have orphaned blobs.

This should be an infrequent operation and is necessary only when you have evidence that significant numbers of new orphans have been created. Otherwise, you can perform standard image pruning at regular intervals, for example, once a day (depending on the number of images being created).

To hard prune orphaned blobs from the registry:

1. **Log in:** Log in using [the CLI](#) as a user with an [access token](#).
2. **Run a basic image prune:** Basic image pruning removes additional images that are no longer needed. The hard prune does not remove images on its own. It only removes blobs stored in the registry storage. Therefore, you should run this just before the hard prune. See [Pruning Images](#) for steps.
3. **Switch the registry to read-only mode:** If the registry is not running in read-only mode, any pushes happening at the same time as the prune will either:
 - fail and cause new orphans, or
 - succeed although the images will not be pullable (because some of the referenced blobs were deleted).

Pushes will not succeed until the registry is switched back to read-write mode. Therefore, the hard prune must be carefully scheduled.

To switch the registry to read-only mode:

- a. Set the following environment variable:

```
$ oc env -n default \
    dc/docker-registry \
    'REGISTRY_STORAGE_MAINTENANCE_READONLY={"enabled":true}'
```

- b. By default, the registry should automatically redeploy when the previous step completes; wait for the redeployment to complete before continuing. However, if you have disabled these triggers, you must manually redeploy the registry so that the new environment variables are picked up:

```
$ oc rollout -n default \
    latest dc/docker-registry
```

4. **Add the `system:image-pruner` role:** The service account used to run the registry instances requires additional permissions in order to list some resources.

- a. Get the service account name:

```
$ service_account=$(oc get -n default \
  -o jsonpath='{$system:serviceaccount:{.metadata.namespace}:
  .spec.template.spec.serviceAccountName}\n' \
  dc/docker-registry)
```

- b. Add the **`system:image-pruner`** cluster role to the service account:

```
$ oc adm policy add-cluster-role-to-user \
  system:image-pruner \
  ${service_account}
```

5. **(Optional) Run the pruner in dry-run mode:** To see how many blobs would be removed, run the hard pruner in dry-run mode. No changes are actually made:

```
$ oc -n default \
  exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-
  registry \
  -o jsonpath='{$.items[0].metadata.name}\n')" \
  -- /usr/bin/dockerregistry -prune=check
```

Alternatively, to get the exact paths for the prune candidates, increase the logging level:

```
$ oc -n default \
  exec "$(oc -n default get pods -l deploymentconfig=docker-
  registry \
  -o jsonpath='{$.items[0].metadata.name}\n')" \
  -- /bin/sh \
  -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -
  prune=check'
```

Sample Output (Truncated)

```
$ oc exec docker-registry-3-vhndw \
  -- /bin/sh -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -
  prune=check'

time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune
(dry-run mode)" distribution_version="v2.4.1+unknown"
kubernetes_version=v1.6.1+$Format:%h$ openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete
blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663
fa5" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete
blob:
sha256:0022d49612807cb348cab562c072ef34d756adfe0100a61952cbcb87ee65
78a" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
```

```

time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete
blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a
37c" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete
blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7
663" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete
blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeec207e4fad2a5bafd28afbef
57e" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete
blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5
e06" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data

```

6. **Run the hard prune:** Execute the following command inside one running instance of **docker-registry** pod to run the hard prune:

```

$ oc -n default \
  exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-
registry -o jsonpath='{$.items[0].metadata.name}\n')" \
  -- /usr/bin/dockerregistry -prune=delete

```

Sample Output

```

$ oc exec docker-registry-3-vhndw \
  -- /usr/bin/dockerregistry -prune=delete

Deleted 13374 blobs
Freed up 2.835 GiB of disk space

```

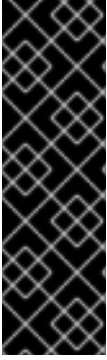
7. **Switch the registry back to read-write mode:** After the prune is finished, the registry can be switched back to read-write mode by executing:

```

$ oc env -n default dc/docker-registry
REGISTRY_STORAGE_MAINTENANCE_READONLY-

```

20.7. PRUNING CRON JOBS



IMPORTANT

Cron Jobs 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/>.

At this time, the results of cron jobs are not automatically pruned, unless a application developer explicitly configures that feature. Therefore, cluster administrator should manually perform a regular [Cron job cleanup](#). Red Hat recommends to [restrict the access](#) to cron jobs to a small group of trusted users and set appropriate [quota](#) to prevent the cron job from creating too many jobs and pods.

CHAPTER 21. EXTENDING THE KUBERNETES API WITH CUSTOM RESOURCES

In the Kubernetes API a resource is an endpoint that stores a collection of API objects of a certain kind. For example, the built-in pods resource contains a collection of Pod objects.

A *custom resource* is an object that extends the Kubernetes API or allows you to introduce your own API into a project or a cluster.

A *custom resource definition* (CRD) file defines your own object kinds and lets the API Server handle the entire lifecycle. Deploying a CRD into the cluster causes the Kubernetes API server to begin serving the specified custom resource.

When you create a new custom resource definition (CRD), the Kubernetes API Server reacts by creating a new RESTful resource path, that can be accessed by an entire cluster or a single project (namespace). As with existing built-in objects, deleting a project deletes all custom objects in that project.

21.1. CREATING CUSTOM RESOURCE DEFINITIONS

To create a CRD, open a YAML file and enter the fields in the following example.

Example YAML file for a Custom Resource Definition

```
apiVersion: apiextensions.k8s.io/v1beta1 ❶
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com ❷
spec:
  group: stable.example.com ❸
  version: v1 ❹
  scope: Namespaced ❺
  names:
    plural: crontabs ❻
    singular: crontab ❼
    kind: CronTab ❽
    shortNames:
      - ct ❾
```

- ❶ Use the `apiextensions.k8s.io/v1beta1` API.
- ❷ Specify a name for the definition. This must be in the `<plural-name><group>` format using the values from the `group` and `plural` fields.
- ❸ Specify a group name for the API. An API group is a collection of objects that are logically related. For example, all batch objects like **Job** or **ScheduledJob** could be in the batch API Group (such as `batch.api.example.com`). A good practice is to use a fully-qualified-domain name of your organization.
- ❹ Specify a version name to be used in the URL. Each API Group can exist in multiple versions. For example: **v1alpha**, **v1beta**, **v1**.
- ❺ Specify whether the custom objects are available to a project (**Namespaced**) or all projects in the cluster (**Cluster**).

- 6 Specify the plural name to be used in the URL. The **plural** field is the same as a resource in an API URL.
- 7 Specify a singular name to be used as an alias on the CLI and for display.
- 8 Specify the kind of objects that can be created. The type can be in CamelCase.
- 9 Specify a shorter string to match your resource on the CLI.

**NOTE**

By default, a custom resource definition is cluster-scoped and available to all projects.

After configuring the definition file, create the object:

```
oc create -f <file-name>.yaml
```

A new RESTful API endpoint is created at:

```
/apis/<spec:group>/<spec:version>/<scope>/*/<names-plural>/...
```

For example, using the example file, the following endpoint would be created:

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

This endpoint URL can then be used to create and manage custom objects. The kind of object is based on the **spec.kind** field of the Custom Resource Definition object you created.

21.2. CREATE CUSTOM OBJECTS

After the custom resource definition object has been created, you can create custom objects.

Custom objects can contain custom fields. These fields can contain arbitrary JSON.

In the following example, the **cronSpec** and **image** custom fields are set in a custom object of kind **CronTab**. The kind **CronTab** comes from the **spec.kind** field of the custom resource definition object you created above.

Example YAML file for a Custom Object

```
apiVersion: "stable.example.com/v1" 1
kind: CronTab 2
metadata:
  name: my-new-cron-object 3
spec: 4
  cronSpec: "* * * * /5"
  image: my-awesome-cron-image
```

- 1 Specify the group name and API version (name/version) from the custom resource definition.
- 2 Specify the type in the custom resource definition.

- 3 Specify a name for the object.
- 4 Specify conditions specific to the type of object.

After configuring the object file, create the object:

```
oc create -f <file-name>.yaml
```

21.3. MANAGE CUSTOM OBJECTS

You can then manage your custom resources.

To get information on a specific kind of custom resource, enter:

```
oc get <kind>
```

For example:

```
oc get crontab

NAME                                KIND
my-new-cron-object                 CronTab.v1.stable.example.com
```

Note that resource names are not case-sensitive, and you can use either the singular or plural forms defined in the CRD, as well as any short name. For example:

```
oc get crontabs
oc get crontab
oc get ct
```

You can also view the raw JSON data:

```
oc get <kind> -o yaml
```

You should see that it contains the custom <1> **cronSpec** and <2> **image** fields from the YAML you used to create it:

```
oc get ct -o yaml

apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-
```

```
new-cron-object
  uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

21.4. FINALIZERS

Custom objects support *finalizers*, which allow controllers to implement conditions that must be completed before the object can be deleted.

You can add a finalizer to a custom object like this:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  finalizers:
  - finalizer.stable.example.com
```

The first delete request on an object with finalizers sets a value for the **metadata.deletionTimestamp** field instead of deleting the object. This triggers controllers watching the object to execute any finalizers they handle.

Each controller then removes the finalizer from the list and issues the delete request again. This request deletes the object only if the list of finalizers is empty, meaning all finalizers are done.

CHAPTER 22. GARBAGE COLLECTION

22.1. OVERVIEW

The OpenShift Container Platform node performs two types of garbage collection:

- [Container garbage collection](#): Removes terminated containers.
- [Image garbage collection](#): Removes images not referenced by any running pods.

22.2. CONTAINER GARBAGE COLLECTION

Container garbage collection is enabled by default and happens automatically in response to eviction thresholds being reached. The node tries to keep any container for any pod accessible from the API. If the pod has been deleted, the containers will be as well. Containers are preserved as long the pod is not deleted and the eviction threshold is not reached. If the node is under disk pressure, it will remove containers and their logs will no longer be accessible via **oc logs**.

The policy for container garbage collection is based on three node settings:

Setting	Description
minimum-container-ttl-duration	The minimum age that a container is eligible for garbage collection. The default is 0 . Use 0 for no limit. Values for this setting can be specified using unit suffixes such as h for hour, m for minutes, s for seconds.
maximum-dead-containers-per-container	The number of instances to retain per pod container. The default is 1 .
maximum-dead-containers	The maximum number of total dead containers in the node. The default is -1 , which means unlimited.

The **maximum-dead-containers** setting takes precedence over the **maximum-dead-containers-per-container** setting when there is a conflict. For example, if retaining the number of **maximum-dead-containers-per-container** would result in a total number of containers that is greater than **maximum-dead-containers**, the oldest containers will be removed to satisfy the **maximum-dead-containers** limit.

When the node removes the dead containers, all files inside those containers are removed as well. Only containers created by the node will be garbage collected.

You can specify values for these settings in the **kubeletArguments** section of the **/etc/origin/node/node-config.yaml** file on node hosts. Add the section if it does not already exist:

Container Garbage Collection Settings

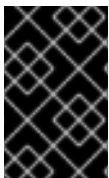
```
kubeletArguments:
  minimum-container-ttl-duration:
    - "10s"
  maximum-dead-containers-per-container:
    - "2"
```

```
maximum-dead-containers:
- "240"
```

22.2.1. Detecting Containers for Deletion

Each spin of the garbage collector loop goes through the following steps:

1. Retrieve a list of available containers.
2. Filter out all containers that are running or are not alive longer than the **minimum-container-ttl-duration** parameter.
3. Classify all remaining containers into equivalence classes based on pod and image name membership.
4. Remove all unidentified containers (containers that are managed by kubelet but their name is malformed).
5. For each class that contains more containers than the **maximum-dead-containers-per-container** parameter, sort containers in the class by creation time.
6. Start removing containers from the oldest first until the **maximum-dead-containers-per-container** parameter is met.
7. If there are still more containers in the list than the **maximum-dead-containers** parameter, the collector starts removing containers from each class so the number of containers in each one is not greater than the average number of containers per class, or **<all_remaining_containers>/<number_of_classes>**.
8. If this is still not enough, sort all containers in the list and start removing containers from the oldest first until the **maximum-dead-containers** criterion is met.



IMPORTANT

Update the default settings to meet your needs.

Garbage collection only removes the containers that do not have a pod associated with it.

22.3. IMAGE GARBAGE COLLECTION

Image garbage collection relies on disk usage as reported by **cAdvisor** on the node to decide which images to remove from the node. It takes the following settings into consideration:

Setting	Description
image-gc-high-threshold	The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is 85 .
image-gc-low-threshold	The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is 80 .

You can specify values for these settings in the **kubeletArguments** section of the */etc/origin/node/node-config.yaml* file on node hosts. Add the section if it does not already exist:

Image Garbage Collection Settings

```
kubeletArguments:
  image-gc-high-threshold:
    - "85"
  image-gc-low-threshold:
    - "80"
```

22.3.1. Detecting Images for Deletion

Two lists of images are retrieved in each garbage collector run:

1. A list of images currently running in at least one pod
2. A list of images available on a host

As new containers are run, new images appear. All images are marked with a time stamp. If the image is running (the first list above) or is newly detected (the second list above), it is marked with the current time. The remaining images are already marked from the previous spins. All images are then sorted by the time stamp.

Once the collection starts, the oldest images get deleted first until the stopping criterion is met.

CHAPTER 23. ALLOCATING NODE RESOURCES

23.1. OVERVIEW

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by all underlying [node components](#) (e.g., kubelet, kube-proxy, Docker) and the remaining system components (e.g., **sshd**, **NetworkManager**) on the host. Once specified, the scheduler has more information about the resources (e.g., memory, CPU) a node has allocated for pods.

23.2. CONFIGURING NODES FOR ALLOCATED RESOURCES

Resources reserved for node components are based on two node settings:

Setting	Description
kube-reserved	Resources reserved for node components. Default is none.
system-reserved	Resources reserved for the remaining system components. Default is none.

You can set these in the **kubeletArguments** section of the [node configuration file](#) (the `/etc/origin/node/node-config.yaml` file by default) using a set of **<resource_type>=<resource_quantity>** pairs (e.g., **cpu=200m,memory=512Mi**). Add the section if it does not already exist:

Example 23.1. Node Allocatable Resources Settings

```
kubeletArguments:
  kube-reserved:
    - "cpu=200m,memory=512Mi"
  system-reserved:
    - "cpu=200m,memory=512Mi"
```

Currently, the **cpu** and **memory** resource types are supported. For **cpu**, the resource quantity is specified in units of cores (e.g., 200m, 0.5, 1). For **memory**, it is specified in units of bytes (e.g., 200Ki, 50Mi, 5Gi).

See [Compute Resources](#) for more details.

If a flag is not set, it defaults to **0**. If none of the flags are set, the allocated resource is set to the node's capacity as it was before the introduction of allocatable resources.

23.3. COMPUTING ALLOCATED RESOURCES

An allocated amount of a resource is computed based on the following formula:

```
[Allocatable] = [Node Capacity] - [kube-reserved] - [system-reserved] -
[Hard-Eviction-Thresholds]
```

**NOTE**

The withholding of **Hard-Eviction-Thresholds** from allocatable is a change in behavior to improve system reliability now that allocatable is enforced for end-user pods at the node level. The **experimental-allocatable-ignore-eviction** setting is available to preserve legacy behavior, but it will be deprecated in a future release.

If **[Allocatable]** is negative, it is set to **0**.

23.4. VIEWING NODE ALLOCATABLE RESOURCES AND CAPACITY

To see a node's current capacity and allocatable resources, you can run:

```
$ oc get node/<node_name> -o yaml
...
status:
...
  allocatable:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
  capacity:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
...
```

23.5. SYSTEM RESOURCES REPORTED BY NODE

Starting with OpenShift Container Platform 3.3, each node reports system resources utilized by the container runtime and kubelet. To better aid your ability to configure **--system-reserved** and **--kube-reserved**, you can introspect corresponding node's resource usage using the node summary API, which is accessible at **<master>/api/v1/nodes/<node>/proxy/stats/summary**.

For instance, to access the resources from **cluster.node22** node, you can run:

```
$ curl <certificate details>
https://<master>/api/v1/nodes/cluster.node22/proxy/stats/summary
{
  "node": {
    "nodeName": "cluster.node22",
    "systemContainers": [
      {
        "cpu": {
          "usageCoreNanoSeconds": 929684480915,
          "usageNanoCores": 190998084
        },
        "memory": {
          "rssBytes": 176726016,
          "usageBytes": 1397895168,
          "workingSetBytes": 1050509312
        },
        "name": "kubelet"
      }
    ]
  },
}
```

```

    {
      "cpu": {
        "usageCoreNanoSeconds": 128521955903,
        "usageNanoCores": 5928600
      },
      "memory": {
        "rssBytes": 35958784,
        "usageBytes": 129671168,
        "workingSetBytes": 102416384
      },
      "name": "runtime"
    }
  ]
}

```

See REST API Overview for more details about certificate details.

23.6. NODE ENFORCEMENT

The node is able to limit the total amount of resources that pods may consume based on the configured allocatable value. This feature significantly improves the reliability of the node by preventing pods from starving system services (for example: container runtime, node agent, etc.) for resources. It is strongly encouraged that administrators reserve resources based on the desired node utilization target in order to improve node reliability.

The node enforces resource constraints using a new cgroup hierarchy that enforces quality of service. All pods are launched in a dedicated cgroup hierarchy separate from system daemons.

To configure this ability, the following kubelet arguments are provided.

Example 23.2. Node Cgroup Settings

```

kubeletArguments:
  cgroups-per-qos:
    - "true" ❶
  cgroup-driver:
    - "systemd" ❷
  enforce-node-allocatable:
    - "pods" ❸

```

- ❶ ❶ Enable or disable the new cgroup hierarchy managed by the node. Any change of this setting requires a full drain of the node. This flag must be true to allow the node to enforce node allocatable. We do not recommend users change this value.
- ❷ ❷ The cgroup driver used by the node when managing cgroup hierarchies. This value must match the driver associated with the container runtime. Valid values are **systemd** and **cgroupfs**. The default is **systemd**.
- ❸ ❸ A comma-delimited list of scopes for where the node should enforce node resource constraints. Valid values are **pods**, **system-reserved**, and **kube-reserved**. The default is **pods**. We do not recommend users change this value.

Optionally, the node can be made to enforce kube-reserved and system-reserved by specifying those tokens in the enforce-node-allocatable flag. If specified, the corresponding **--kube-reserved-cgroup** or **--system-reserved-cgroup** needs to be provided. In future releases, the node and container runtime will be packaged in a common cgroup separate from **system.slice**. Until that time, we do not recommend users change the default value of enforce-node-allocatable flag.

Administrators should treat system daemons similar to Guaranteed pods. System daemons can burst within their bounding control groups and this behavior needs to be managed as part of cluster deployments. Enforcing system-reserved limits can lead to critical system services being CPU starved or OOM killed on the node. The recommendation is to enforce system-reserved only if operators have profiled their nodes exhaustively to determine precise estimates and are confident in their ability to recover if any process in that group is OOM killed.

As a result, we strongly recommended that users only enforce node allocatable for **pods** by default, and set aside appropriate reservations for system daemons to maintain overall node reliability.

23.7. EVICTION THRESHOLDS

If a node is under memory pressure, it can impact the entire node and all pods running on it. If a system daemon is using more than its reserved amount of memory, an OOM event may occur that can impact the entire node and all pods running on it. To avoid (or reduce the probability of) system OOMs the node provides [Out Of Resource Handling](#).

By reserving some memory via the **--eviction-hard** flag, the node attempts to evict pods whenever memory availability on the node drops below the absolute value or percentage. If system daemons did not exist on a node, pods are limited to the memory **capacity - eviction-hard**. For this reason, resources set aside as a buffer for eviction before reaching out of memory conditions are not available for pods.

Here is an example to illustrate the impact of node allocatable for memory:

- Node capacity is **32Gi**
- **--kube-reserved** is **2Gi**
- **--system-reserved** is **1Gi**
- **--eviction-hard** is set to **<100Mi**.

For this node, the effective node allocatable value is **28.9Gi**. If the node and system components use up all their reservation, the memory available for pods is **28.9Gi**, and kubelet will evict pods when it exceeds this usage.

If we enforce node allocatable (**28.9Gi**) via top level cgroups, then pods can never exceed **28.9Gi**. Evictions would not be performed unless system daemons are consuming more than **3.1Gi** of memory.

If system daemons do not use up all their reservation, with the above example, pods would face memcg OOM kills from their bounding cgroup before node evictions kick in. To better enforce QoS under this situation, the node applies the hard eviction thresholds to the top-level cgroup for all pods to be **Node Allocatable + Eviction Hard Thresholds**.

If system daemons do not use up all their reservation, the node will evict pods whenever they consume more than **28.9Gi** of memory. If eviction does not occur in time, a pod will be OOM killed if pods consume **29Gi** of memory.

23.8. SCHEDULER

The scheduler now uses the value of **node.Status.Allocatable** instead of **node.Status.Capacity** to decide if a node will become a candidate for pod scheduling.

By default, the node will report its machine capacity as fully schedulable by the cluster.

CHAPTER 24. OPAQUE INTEGER RESOURCES

24.1. OVERVIEW

Opaque integer resources allow cluster operators to provide new node-level resources that would be otherwise unknown to the system. Users can consume these resources in pod specifications, similar to CPU and memory. The scheduler performs resource accounting so that no more than the available amount is simultaneously allocated to pods.



NOTE

Opaque integer resources are Alpha currently, and only resource accounting is implemented. There is no resource quota or limit range support for these resources, and they have no impact on QoS.

Opaque integer resources are called *opaque* because OpenShift Container Platform does not know what the resource is, but will schedule a pod on a node only if enough of that resource is available. They are called *integer resources* because they must be available, or *advertised*, in integer amounts. The API server restricts quantities of these resources to whole numbers. Examples of *valid* quantities are **3**, **3000m**, and **3Ki**.

Opaque integer resources can be used to allocate:

- Last-level cache (LLC)
- Graphics processing unit (GPU) devices
- Field-programmable gate array (FPGA) devices
- Slots for sharing bandwidth to a parallel file system.

For example, if a node has 800 GiB of a special kind of disk storage, you could create a name for the special storage, such as ***opaque-int-resource-special-storage***. You could advertise it in chunks of a certain size, such as 100 GiB. In that case, your node would advertise that it has eight resources of type ***opaque-int-resource-special-storage***.

Opaque integer resource names must begin with the prefix **`pod.alpha.kubernetes.io/opaque-int-resource-`**.

24.2. CREATING OPAQUE INTEGER RESOURCES

There are two steps required to use opaque integer resources. First, the cluster operator must name and advertise a per-node opaque resource on one or more nodes. Second, application developer must request the opaque resource in pods.

To make opaque integer resources available:

1. Allocate the resource and assign a name starting with **`pod.alpha.kubernetes.io/opaque-int-resource-`**
2. Advertise a new opaque integer resource by submitting a PATCH HTTP request to the API server that specifies the available quantity in the **`status.capacity`** for a node in the cluster. For example, the following HTTP request advertises five **`foo`** resources on the **`openshift-node-1`** node.

```

PATCH /api/v1/nodes/openshift-node-1/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: openshift-master:8080

[
  {
    "op": "add",
    "path": "/status/capacity/pod.alpha.kubernetes.io~1opaque-int-
resource-foo",
    "value": "5"
  }
]

```

**NOTE**

The **~1** in the **path** is the encoding for the character **/**. The operation path value in the JSON-Patch is interpreted as a JSON-Pointer. For more details, refer to [IETF RFC 6901, section 3](#).

After this operation, the node **status.capacity** includes a new resource. The **status.allocatable** field is updated automatically with the new resource asynchronously.

**NOTE**

Since the scheduler uses the node **status.allocatable** value when evaluating pod fitness, there might be a short delay between patching the node capacity with a new resource and the first pod that requests the resource to be scheduled on that node.

The application developer can then consume the opaque resources by editing the pod config to include the name of the opaque resource as a key in the **spec.containers[].resources.requests** field.

For example: The following pod requests two CPUs and one **foo** (an opaque resource).

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: myimage
    resources:
      requests:
        cpu: 2
        pod.alpha.kubernetes.io/opaque-int-resource-foo: 1

```

The pod will be scheduled only if all of the resource requests are satisfied (including CPU, memory, and any opaque resources). The pod will remain in the **PENDING** state while the resource request cannot be met by any node.

Conditions:

```

  Type      Status
PodScheduled False
...
Events:
  FirstSeen    LastSeen  Count  From              SubObjectPath Type      Reason
Message
-----
-----
  14s          0s        6  default-scheduler Warning    FailedScheduling  No nodes
are available that match all of the following predicates:: Insufficient
pod.alpha.kubernetes.io/opaque-int-resource-foo (1).
```

This information can also be found in the Developer Guide under [Quotas and Limit Ranges](#).

CHAPTER 25. OVERCOMMITTING

25.1. OVERVIEW

Containers can specify [compute resource requests and limits](#). Requests are used for scheduling your container and provide a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

The [scheduler](#) attempts to optimize the compute resource use across all nodes in your cluster. It places pods onto specific nodes, taking the pods' compute resource requests and nodes' available capacity into consideration.

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node, which may be desirable in development environments where a tradeoff of guaranteed performance for capacity is acceptable.

25.2. REQUESTS AND LIMITS

For each compute resource, a container may specify a resource request and limit. Scheduling decisions are made based on the request to ensure that a node has enough capacity available to meet the requested value. If a container specifies limits, but omits requests, the requests are defaulted to the limits. A container is not able to exceed the specified limit on the node.

The enforcement of limits is dependent upon the compute resource type. If a container makes no request or limit, the container is scheduled to a node with no resource guarantees. In practice, the container is able to consume as much of the specified resource as is available with the lowest local priority. In low resource situations, containers that specify no resource requests are given the lowest quality of service.

25.2.1. Tune Buffer Chunk Limit

If Fluentd logger is unable to keep up with a high number of logs, it will need to switch to file buffering to reduce memory usage and prevent data loss.

The Fluentd **buffer_chunk_limit** is determined by the environment variable **BUFFER_SIZE_LIMIT**, which has the default value **8m**. The file buffer size per output is determined by the environment variable **FILE_BUFFER_LIMIT**, which has the default value **256Mi**. The permanent volume size must be larger than **FILE_BUFFER_LIMIT** multiplied by the output.

On the Fluentd and Mux pods, permanent volume **/var/lib/fluentd** should be prepared by the PVC or hostmount, for example. That area is then used for the file buffers.

The **buffer_type** and **buffer_path** are configured in the Fluentd configuration files as follows:

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
    buffer_type file
    buffer_path `/var/lib/fluentd/buffer-output-es-config`
output-es-ops-config.conf:
    buffer_type file
    buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
filter-pre-mux-client.conf:
    buffer_type file
    buffer_path `/var/lib/fluentd/buffer-mux-client`
```

The Fluentd **buffer_queue_limit** is the value of the variable **BUFFER_QUEUE_LIMIT**. This value is **32** by default.

The environment variable **BUFFER_QUEUE_LIMIT** is calculated as **(FILE_BUFFER_LIMIT / (number_of_outputs * BUFFER_SIZE_LIMIT))**.

If the **BUFFER_QUEUE_LIMIT** variable has the default set of values:

- **FILE_BUFFER_LIMIT = 256Mi**
- **number_of_outputs = 1**
- **BUFFER_SIZE_LIMIT = 8Mi**

The value of **buffer_queue_limit** will be **32**. To change the **buffer_queue_limit**, you need to change the value of **FILE_BUFFER_LIMIT**.

In this formula, **number_of_outputs** is **1** if all the logs are sent to a single resource, and it is incremented by **1** for each additional resource. For example, the value of **number_of_outputs** is:

- **1** - if all logs are sent to a single Elasticsearch pod
- **2** - if application logs are sent to an Elasticsearch pod and ops logs are sent to another Elasticsearch pod
- **4** - if application logs are sent to an Elasticsearch pod, ops logs are sent to another Elasticsearch pod, and both of them are forwarded to other Fluentd instances

25.3. COMPUTE RESOURCES

The node-enforced behavior for compute resources is specific to the resource type.

25.3.1. CPU

A container is guaranteed the amount of CPU it requests and is additionally able to consume excess CPU available on the node, up to any limit specified by the container. If multiple containers are attempting to use excess CPU, CPU time is distributed based on the amount of CPU requested by each container.

For example, if one container requested 500m of CPU time and another container requested 250m of CPU time, then any extra CPU time available on the node is distributed among the containers in a 2:1 ratio. If a container specified a limit, it will be throttled not to use more CPU than the specified limit.

CPU requests are enforced using the CFS shares support in the Linux kernel. By default, CPU limits are enforced using the CFS quota support in the Linux kernel over a 100ms measuring interval, though [this can be disabled](#).

25.3.2. Memory

A container is guaranteed the amount of memory it requests. A container may use more memory than requested, but once it exceeds its requested amount, it could be killed in a low memory situation on the node.

If a container uses less memory than requested, it will not be killed unless system tasks or daemons need more memory than was accounted for in the node's resource reservation. If a container specifies a limit on memory, it is immediately killed if it exceeds the limit amount.

25.4. QUALITY OF SERVICE CLASSES

A node is *overcommitted* when it has a pod scheduled that makes no request, or when the sum of limits across all pods on that node exceeds available machine capacity.

In an overcommitted environment, it is possible that the pods on the node will attempt to use more compute resource than is available at any given point in time. When this occurs, the node must give priority to one pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) Class.

For each compute resource, a container is divided into one of three QoS classes with decreasing order of priority:

Table 25.1. Quality of Service Classes

Priority	Class Name	Description
1 (highest)	Guaranteed	If limits and optionally requests are set (not equal to 0) for all resources and they are equal, then the container is classified as Guaranteed .
2	Burstable	If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the container is classified as Burstable .
3 (lowest)	BestEffort	If requests and limits are not set for any of the resources, then the container is classified as BestEffort .

Memory is an incompressible resource, so in low memory situations, containers that have the lowest priority are killed first:

- **Guaranteed** containers are considered top priority, and are guaranteed to only be killed if they exceed their limits, or if the system is under memory pressure and there are no lower priority containers that can be evicted.
- **Burstable** containers under system memory pressure are more likely to be killed once they exceed their requests and no other **BestEffort** containers exist.
- **BestEffort** containers are treated with the lowest priority. Processes in these containers are first to be killed if the system runs out of memory.

25.5. CONFIGURING MASTERS FOR OVERCOMMITMENT

Scheduling is based on resources requested, while quota and hard limits refer to resource limits, which can be set higher than requested resources. The difference between request and limit determines the level of overcommit; for instance, if a container is given a memory request of 1Gi and a memory limit of 2Gi, it is scheduled based on the 1Gi request being available on the node, but could use up to 2Gi; so it is 200% overcommitted.

If OpenShift Container Platform administrators would like to control the level of overcommit and manage

container density on nodes, masters can be configured to override the ratio between request and limit set on developer containers. In conjunction with a [per-project LimitRange](#) specifying limits and defaults, this adjusts the container limit and request to achieve the desired level of overcommit.

This requires configuring the **ClusterResourceOverride** admission controller in the **master-config.yaml** as in the following example (reuse the existing configuration tree if it exists, or introduce absent elements as needed):

```
admissionConfig:
  pluginConfig:
    ClusterResourceOverride: ❶
    configuration:
      apiVersion: v1
      kind: ClusterResourceOverrideConfig
      memoryRequestToLimitPercent: 25 ❷
      cpuRequestToLimitPercent: 25 ❸
      limitCPUToMemoryPercent: 200 ❹
```

- ❶ This is the plug-in name; case matters and anything but an exact match for a plug-in name is ignored.
- ❷ (optional, 1-100) If a container memory limit has been specified or defaulted, the memory request is overridden to this percentage of the limit.
- ❸ (optional, 1-100) If a container CPU limit has been specified or defaulted, the CPU request is overridden to this percentage of the limit.
- ❹ (optional, positive integer) If a container memory limit has been specified or defaulted, the CPU limit is overridden to a percentage of the memory limit, with a 100 percentage scaling 1Gi of RAM to equal 1 CPU core. This is processed prior to overriding CPU request (if configured).

After changing the master configuration, a master restart is required.

Note that these overrides have no effect if no limits have been set on containers. [Create a LimitRange object](#) with default limits (per individual project, or in the [project template](#)) in order to ensure that the overrides apply.

Note also that after overrides, the container limits and requests must still be validated by any LimitRange objects in the project. It is possible, for example, for developers to specify a limit close to the minimum limit, and have the request then be overridden below the minimum limit, causing the pod to be forbidden. This unfortunate user experience should be addressed with future work, but for now, configure this capability and LimitRanges with caution.

When configured, overrides can be disabled per-project (for example, to allow infrastructure components to be configured independently of overrides) by editing the project and adding the following annotation:

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

25.6. CONFIGURING NODES FOR OVERCOMMITMENT

In an overcommitted environment, it is important to properly configure your node to provide best system behavior.

25.6.1. Reserving Memory Across Quality of Service Tiers

You can use the **experimental-qos-reserved** parameter to specify a percentage of memory to be reserved by a pod in a particular QoS level. This feature attempts to reserve requested resources to exclude pods from lower QoS classes from using resources requested by pods in higher QoS classes.

By reserving resources for higher QoS levels, pods that don't have resource limits are prevented from encroaching on the resources requested by pods at higher QoS levels.



IMPORTANT

The **experimental-qos-reserved** parameter 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/>.

To configure **experimental-qos-reserved**, edit the `/etc/origin/node/node-config.yaml` file for the node.

```
kubeletArguments:
  cgroups-per-qos:
    - true
  cgroup-driver:
    - 'systemd'
  cgroup-root:
    - '/'
  experimental-qos-reserved: 1
    - 'memory=50%'
```

- 1 Specifies how pod resource requests are reserved at the QoS level.

OpenShift Container Platform uses the **experimental-qos-reserved** parameter as follows:

- A value of **experimental-qos-reserved=memory=100%** will prevent the **Burstable** and **BestEffort** QoS classes from consuming memory that was requested by a higher QoS class. This increases the risk of inducing OOM on **BestEffort** and **Burstable** workloads in favor of increasing memory resource guarantees for **Guaranteed** and **Burstable** workloads.
- A value of **experimental-qos-reserved=memory=50%** will allow the **Burstable** and **BestEffort** QoS classes to consume half of the memory requested by a higher QoS class.
- A value of **experimental-qos-reserved=memory=0%** will allow a **Burstable** and **BestEffort** QoS classes to consume up to the full node allocatable amount if available, but increases the risk that a **Guaranteed** workload will not have access to requested memory. This condition effectively disables this feature.

25.6.2. Enforcing CPU Limits

Nodes by default enforce specified CPU limits using the CPU CFS quota support in the Linux kernel. If you do not want to enforce CPU limits on the node, you can disable its enforcement by modifying the [node configuration file](#) (the ***node-config.yaml*** file) to include the following:

```
kubeletArguments:
  cpu-cfs-quota:
    - "false"
```

If CPU limit enforcement is disabled, it is important to understand the impact that will have on your node:

- If a container makes a request for CPU, it will continue to be enforced by CFS shares in the Linux kernel.
- If a container makes no explicit request for CPU, but it does specify a limit, the request will default to the specified limit, and be enforced by CFS shares in the Linux kernel.
- If a container specifies both a request and a limit for CPU, the request will be enforced by CFS shares in the Linux kernel, and the limit will have no impact on the node.

25.6.3. Reserving Resources for System Processes

The [scheduler](#) ensures that there are enough resources for all pods on a node based on the pod requests. It verifies that the sum of requests of containers on the node is no greater than the node capacity. It includes all containers started by the node, but not containers or processes started outside the knowledge of the cluster.

It is recommended that you reserve some portion of the node capacity to allow for the system daemons that are required to run on your node for your cluster to function (**sshd**, **docker**, etc.). In particular, it is recommended that you reserve resources for incompressible resources such as memory.

If you want to explicitly reserve resources for non-pod processes, there are two ways to do so:

- The preferred method is to allocate node resources by specifying resources available for scheduling. See [Allocating Node Resources](#) for more details.
- Alternatively, you can create a **resource-reserver** pod that does nothing but reserve capacity from being scheduled on the node by the cluster. For example:

Example 25.1. resource-reserver Pod Definition

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-reserver
spec:
  containers:
    - name: sleep-forever
      image: gcr.io/google_containers/pause:0.8.0
      resources:
        limits:
          cpu: 100m 1
          memory: 150Mi 2
```

- 1 The amount of CPU to reserve on a node for host-level daemons unknown to the cluster.

- 2 The amount of memory to reserve on a node for host-level daemons unknown to the cluster.

You can save your definition to a file, for example **resource-reserver.yaml**, then place the file in the node configuration directory, for example **/etc/origin/node/** or the **--config=<dir>** location if otherwise specified.

Additionally, the node server needs to be configured to read the definition from the node configuration directory, by naming the directory in the **kubeletArguments.config** field of the [node configuration file](#) (usually named **node-config.yaml**):

```
kubeletArguments:
  config:
    - "/etc/origin/node" 1
```

- 1 If **--config=<dir>** is specified, use **<dir>** here.

With the **resource-reserver.yaml** file in place, starting the node server also launches the **sleep-forever** container. The scheduler takes into account the remaining capacity of the node, adjusting where to place cluster pods accordingly.

To remove the **resource-reserver** pod, you can delete or move the **resource-reserver.yaml** file from the node configuration directory.

25.6.4. Kernel Tunable Flags

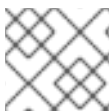
When the node starts, it ensures that the kernel tunable flags for memory management are set properly. The kernel should never fail memory allocations unless it runs out of physical memory.

To ensure this behavior, the node instructs the kernel to always overcommit memory:

```
$ sysctl -w vm.overcommit_memory=1
```

The node also instructs the kernel not to panic when it runs out of memory. Instead, the kernel OOM killer should kill processes based on priority:

```
$ sysctl -w vm.panic_on_oom=0
```



NOTE

The above flags should already be set on nodes, and no further action is required.

25.6.5. Disabling Swap Memory

You can disable swap by default on your nodes in order to preserve quality of service guarantees. Otherwise, physical resources on a node can oversubscribe, affecting the resource guarantees the Kubernetes scheduler makes during pod placement.

For example, if two guaranteed pods have reached their memory limit, each container could start using swap memory. Eventually, if there is not enough swap space, processes in the pods can be terminated due to the system being oversubscribed.

To disable swap:

```
$ swapoff -a
```

Failing to disable swap results in nodes not recognizing that they are experiencing **MemoryPressure**, resulting in pods not receiving the memory they made in their scheduling request. As a result, additional pods are placed on the node to further increase memory pressure, ultimately increasing your risk of experiencing a system out of memory (OOM) event.



IMPORTANT

If swap is enabled, any [out of resource handling](#) eviction thresholds for available memory will not work as expected. Take advantage of out of resource handling to allow pods to be evicted from a node when it is under memory pressure, and rescheduled on an alternative node that has no such pressure.

CHAPTER 26. ASSIGNING UNIQUE EXTERNAL IPS FOR INGRESS TRAFFIC

26.1. OVERVIEW

One approach to getting external traffic into the cluster is by using ExternalIP or IngressIP addresses.



IMPORTANT

This feature is only supported in non-cloud deployments. For cloud (GCE, AWS, and OpenStack) deployments, use the Load Balancer services for automatic deployment of a cloud load balancer to target the service's endpoints.

OpenShift Container Platform supports two pools of IP addresses:

- IngressIP uses by the Loadbalancer when choosing an external IP address for the service.
- ExternalIP is used when the user selects a specific IP from the configured pool.



NOTE

Both have to be configured to a device on an OpenShift Container Platform host to be used, whether with network interface controller (NIC) or virtual ethernet, as well as external routing. Ipfailover is recommended for this, because it selects the host and configures the NIC.

IngressIP and ExternalIP both allow external traffic access to the cluster, and, if routed correctly, external traffic can reach that service's endpoints via any TCP/UDP port the service exposes. This can be simpler than having to manage the port space of a limited number of shared IP addresses when manually assigning external IPs to services. Also, these addresses can be used as virtual IPs (VIPs) when configuring [high availability](#).

OpenShift Container Platform supports both the automatic and manual assignment of IP addresses, and each address is guaranteed to be assigned to a maximum of one service. This ensures that each service can expose its chosen ports regardless of the ports exposed by other services.

26.2. RESTRICTIONS

To use an **ExternalIP**, you can:

- Select an IP address from the [externalIPNetworkCIDRs](#) range.
- Have an IP address assigned from the [ingressIPNetworkCIDR](#) pool in the master configuration file. In this case, OpenShift Container Platform implements a non-cloud version of the load balancer service type and assigns IP addresses to the services.

CAUTION

You must ensure that the IP address pool you assign terminates at one or more nodes in your cluster. You can use the existing [oc adm ipfailover](#) to ensure that the external IPs are highly available.

For manually-configured external IPs, potential port clashes are handled on a first-come, first-served basis. If you request a port, it is only available if it has not yet been assigned for that IP address. For example:

Port clash example for manually-configured external IPs

Two services have been manually configured with the same external IP address of 172.7.7.7.

MongoDB service A requests port 27017, and then **MongoDB service B** requests the same port; the first request gets the port.

However, port clashes are not an issue for external IPs assigned by the ingress controller, because the controller assigns each service a unique address.

26.3. CONFIGURING THE CLUSTER TO USE UNIQUE EXTERNAL IPS

In non-cloud clusters, `ingressIPNetworkCIDR` is set by default to `172.29.0.0/16`. If your cluster environment is not already using this private range, you can use the default. However, if you want to use a different range, then you must set `ingressIPNetworkCIDR` in the `/etc/origin/master/master-config.yaml` file before you assign an ingress IP. Then, restart the master service.

CAUTION

External IPs assigned to services of type **LoadBalancer** will always be in the range of `ingressIPNetworkCIDR`. If `ingressIPNetworkCIDR` is changed such that the assigned external IPs are no longer in range, the affected services will be assigned new external IPs compatible with the new range.



NOTE

If you are using [high availability](#), then this range must be less than 255 IP addresses.

Sample `/etc/origin/master/master-config.yaml`

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

26.3.1. Configuring an Ingress IP for a Service

To assign an ingress IP:

1. Create a YAML file for a LoadBalancer service that requests a specific IP via the `loadBalancerIP` setting:

Sample LoadBalancer Configuration

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: db
```

```

    port: 3306
    loadBalancerIP: 172.29.0.1
    type: LoadBalancer
    selector:
      name: my-db-selector

```

2. Create a LoadBalancer service on your pod:

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

3. Check the service for an external IP. For example, for a service named **myservice**:

```
$ oc get svc myservice
```

When your LoadBalancer-type service has an external IP assigned, the output displays the IP:

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
myservice	172.30.74.106	172.29.0.1	3306/TCP	30s

26.4. ROUTING THE INGRESS CIDR FOR DEVELOPMENT OR TESTING

Add a static route directing traffic for the ingress CIDR to a node in the cluster. For example:

```
# route add -net 172.29.0.0/16 gw 10.66.140.17 eth0
```

In the example above, **172.29.0.0/16** is the **ingressIPNetworkCIDR**, and **10.66.140.17** is the node IP.

26.4.1. Service externalIPs

In addition to the cluster's internal IP addresses, the application developer can configure IP addresses that are external to the cluster. As the OpenShift Container Platform administrator, you are responsible for ensuring that traffic arrives at a node with this IP.

The externalIPs must be selected by the administrator from the **externalIPNetworkCIDRs** range configured in the **master-config.yaml** file. When **master-config.yaml** changes, the master services must be restarted.

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

Sample externalIPNetworkCIDR /etc/origin/master/master-config.yaml

```
networkConfig:
  externalIPNetworkCIDR: 172.47.0.0/24
```

Service externalIPs Definition (JSON)

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
```

```
    "name": "my-service"
  },
  "spec": {
    "selector": {
      "app": "MyApp"
    },
    "ports": [
      {
        "name": "http",
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376
      }
    ],
    "externalIPs" : [
      "80.11.12.10"
    ]
  }
}
```

1

- 1** List of External IP addresses on which the **port** is exposed. In addition to the internal IP addresses)

CHAPTER 27. HANDLING OUT OF RESOURCE ERRORS

27.1. OVERVIEW

This topic discusses best-effort attempts to prevent OpenShift Container Platform from experiencing out-of-memory (OOM) and out-of-disk-space conditions.

A node must maintain stability when available compute resources are low. This is especially important when dealing with incompressible resources such as memory or disk. If either resource is exhausted, the node becomes unstable.

Administrators can proactively monitor nodes for and prevent against situations where the node runs out of compute and memory resources using configurable [eviction policies](#).

This topic also provides information on how OpenShift Container Platform handles out-of-resource conditions and provides an [example scenario](#) and [recommended practices](#):

- [Resource reclaiming](#)
- [Pod eviction](#)
- [Pod scheduling](#)
- [Out of Resource and Out of Memory Killer](#)



WARNING

If swap memory is enabled for a node, that node cannot detect that it is under **MemoryPressure**.

To take advantage of memory based evictions, operators must [disable swap](#).

27.2. CONFIGURING EVICTION POLICIES

An *eviction policy* allows a node to fail one or more pods when the node is running low on available resources. Failing a pod allows the node to reclaim needed resources.

An eviction policy is a combination of an [eviction trigger signal](#) with a specific [eviction threshold value](#) that is set in the node configuration file or through the [command line](#). Evictions can be either [hard](#), where a node takes immediate action on a pod that exceeds a threshold, or [soft](#), where a node allows a grace period before taking action.

By using well-configured eviction policies, a node can proactively monitor for and prevent against total starvation of a compute resource.



NOTE

When the node fails a pod, it terminates all containers in the pod, and the **PodPhase** is transitioned to **Failed**.

When detecting disk pressure, the node supports the **nodefs** and **imagefs** file system partitions.

The **nodefs**, or **rootfs**, is the file system that the node uses for local disk volumes, daemon logs, emptyDir, and so on (for example, the file system that provides `/`). The **rootfs** contains **openshift.local.volumes**, by default `/var/lib/origin/openshift.local.volumes`.

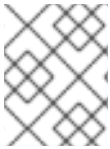
The **imagefs** is the file system that the container runtime uses for storing images and individual container-writable layers. Eviction thresholds are at 85% full for **imagefs**. The **imagefs** file system depends on the runtime and, in the case of Docker, which storage driver you are using.

- For Docker:
 - If you are using the **devicemapper** storage driver, the **imagefs** is thin pool. You can limit the read/write layer for the container by setting the `--storage-opt dm.basesize` flag in the Docker daemon.

```
$ sudo dockerd --storage-opt dm.basesize=50G
```

- If you are using the **overlay2** storage driver, the **imagefs** is the file system that contains `/var/lib/docker/overlay2`.

- For CRI-O, which uses the overlay driver, the **imagefs** is `/var/lib/containers/storage` by default.



NOTE

If you do not use local storage isolation (ephemeral storage) and not using XFS quota (volumeConfig), you cannot limit local disk usage by the pod.

27.2.1. Using the Node Configuration to Create a Policy

To configure an eviction policy, edit the node configuration file (the `/etc/origin/node/node-config.yaml` file) to specify the eviction thresholds under the **eviction-hard** or **eviction-soft** parameters.

For example:

Example 27.1. Sample Node Configuration file for a hard eviction

```
kubeletArguments:
  eviction-hard: 1
    - memory.available<100Mi 2
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
    - imagefs.inodesFree<10%
```

1 The type of eviction: Use this parameter for a [hard eviction](#).

2 Eviction thresholds based on a specific eviction trigger signal.



NOTE

You must provide percentage values for the **inodesFree** parameters. You can provide a percentage or a numerical value for the other parameters.

Example 27.2. Sample Node Configuration file for a soft eviction

```
kubeletArguments:
  eviction-soft: 1
    - memory.available<100Mi 2
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
    - imagefs.inodesFree<10%
  eviction-soft-grace-period: 3
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

- 1 The type of eviction: Use this parameter for a [soft eviction](#).
- 2 An eviction threshold based on a specific eviction trigger signal.
- 3 The grace period for the soft eviction. Leave the default values for optimal performance.

1. Restart the OpenShift Container Platform service for the changes to take effect:

```
# systemctl restart atomic-openshift-node
```

27.2.2. Understanding Eviction Signals

You can configure a node to trigger eviction decisions on any of the signals described in the table below. You add an eviction signal to an [eviction threshold](#) along with a threshold value.

The value of each signal is described in the **Description** column based on the node summary API.

To view the signals:

```
curl <certificate details> \
  https://<master>/api/v1/nodes/<node>/proxy/stats/summary
```

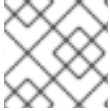
Table 27.1. Supported Eviction Signals

Node Condition	Eviction Signal	Value	Description
----------------	-----------------	-------	-------------

Node Condition	Eviction Signal	Value	Description
MemoryPressure	memory.available	memory.available = node.status.capacity[memory] - node.stats.memory.workingSet	Available memory on the node has exceeded an eviction threshold.
DiskPressure	nodefs.available	nodefs.available = node.stats.fs.available	Available disk space on either the node root file system or image file system has exceeded an eviction threshold.
	nodefs.inodesFree	nodefs.inodesFree = node.stats.fs.inodesFree	
	imagefs.available	imagefs.available = node.stats.runtime.imagefs.available	
	imagefs.inodesFree	imagefs.inodesFree = node.stats.runtime.imagefs.inodesFree	

Each of the above signals supports either a literal or percentage-based value. The percentage-based value is calculated relative to the total capacity associated with each signal.

A script derives the value for **memory.available** from your cgroup driver using the same set of steps that the kubelet performs. The script excludes inactive file memory (that is, the number of bytes of file-backed memory on inactive LRU list) from its calculation as it assumes that inactive file memory is reclaimable under pressure.



NOTE

Do not use tools like **free -m**, because **free -m** does not work in a container.

OpenShift Container Platform monitors these file systems every 10 seconds.

If you store volumes and logs in a dedicated file system, the node will not monitor that file system.



NOTE

As of OpenShift Container Platform 3.4, the node supports the ability to trigger eviction decisions based on disk pressure. Before evicting pods because of disk pressure, the node also performs [container and image garbage collection](#). In future releases, garbage collection will be deprecated in favor of a pure disk-eviction based configuration.

27.2.3. Understanding Eviction Thresholds

You can configure a node to specify eviction thresholds, which triggers the node to reclaim resources, by adding a threshold to the [node configuration file](#).

If an eviction threshold is met, independent of its associated grace period, the node reports a condition indicating that the node is under memory or disk pressure. This prevents the scheduler from scheduling any additional pods on the node while attempts to reclaim resources are made.

The node continues to report node status updates at the frequency specified by the **node-status-update-frequency** argument, which defaults to **10s** (ten seconds).

Eviction thresholds can be [hard](#), for when the node takes immediate action when a threshold is met, or [soft](#), for when you allow a grace period before reclaiming resources.



NOTE

Soft eviction usage is more common when you are targeting a certain level of utilization, but can tolerate temporary spikes. We recommended setting the soft eviction threshold lower than the hard eviction threshold, but the time period can be operator-specific. The system reservation should also cover the soft eviction threshold.

The soft eviction threshold is an advanced feature. You should configure a hard eviction threshold before attempting to use soft eviction thresholds.

Thresholds are configured in the following form:

```
<eviction_signal><operator><quantity>
```

- the **eviction-signal** value can be any [supported eviction signal](#).

- the **operator** value is `<`.
- the **quantity** value must match the [quantity representation](#) used by Kubernetes and can be expressed as a percentage if it ends with the `%` token.

For example, if an operator has a node with 10Gi of memory, and that operator wants to induce eviction if available memory falls below 1Gi, an eviction threshold for memory can be specified as either of the following:

```
memory.available<1Gi
memory.available<10%
```



NOTE

The node evaluates and monitors eviction thresholds every 10 seconds and the value can not be modified. This is the housekeeping interval.

27.2.3.1. Understanding Hard Eviction Thresholds

A hard eviction threshold has no grace period and, if observed, the node takes immediate action to reclaim the associated starved resource. If a hard eviction threshold is met, the node kills the pod immediately with no graceful termination.

To configure hard eviction thresholds, add eviction thresholds to the [node configuration file](#) under **eviction-hard**, as shown in [Using the Node Configuration to Create a Policy](#).

Sample Node Configuration file with hard eviction thresholds

```
kubeletArguments:
  eviction-hard:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
```

This example is a general guideline and not recommended settings.

27.2.3.1.1. Default Hard Eviction Thresholds

OpenShift Container Platform uses the following default configuration for **eviction-hard**.

```
...
kubeletArguments:
  eviction-hard:
    - memory.available<100Mi
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
...
```

27.2.3.2. Understanding Soft Eviction Thresholds

A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period. The node does not reclaim resources associated with the eviction signal until that grace period is exceeded. If no grace period is provided in the node configuration the node errors on startup.

In addition, if a soft eviction threshold is met, an operator can specify a maximum allowed pod termination grace period to use when evicting pods from the node. If **eviction-max-pod-grace-period** is specified, the node uses the lesser value among the **pod.Spec.TerminationGracePeriodSeconds** and the maximum-allowed grace period. If not specified, the node kills pods immediately with no graceful termination.

For soft eviction thresholds the following flags are supported:

- **eviction-soft**: a set of eviction thresholds (for example, **memory.available<1.5Gi**) that, if met over a corresponding grace period, triggers a pod eviction.
- **eviction-soft-grace-period**: a set of eviction grace periods (for example, **memory.available=1m30s**) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction.
- **eviction-max-pod-grace-period**: the maximum-allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met.

To configure soft eviction thresholds, add eviction thresholds to the [node configuration file](#) under **eviction-soft**, as shown in [Using the Node Configuration to Create a Policy](#).

Sample Node Configuration files with soft eviction thresholds

```
kubeletArguments:
  eviction-soft:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
  eviction-soft-grace-period:
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

This example is a general guideline and not recommended settings.

27.3. CONFIGURING THE AMOUNT OF RESOURCE FOR SCHEDULING

You can control how much of a node resource is made available for scheduling in order to allow the scheduler to fully allocate a node and to prevent evictions.

Set **system-reserved** equal to the amount of resource you want available to the scheduler for deploying pods and for system-daemons. Evictions should only occur if pods use more than their requested amount of an allocatable resource.

A node reports two values:

- **Capacity**: How much resource is on the machine

- **Allocatable:** How much resource is made available for scheduling.

To configure the amount of allocatable resources:

1. Edit the node configuration file (the `/etc/origin/node/node-config.yaml` file) to add or modify the **system-reserved** parameter for **eviction-hard** or **eviction-soft**.

```
kubeletArguments:
  eviction-hard: 1
    - "memory.available<500Mi"
  system-reserved:
    - "memory=1.5Gi"
```

- 1 This threshold can either be **eviction-hard** or **eviction-soft**.

2. Restart the OpenShift Container Platform service for the changes to take effect:

```
# systemctl restart atomic-openshift-node
```

27.4. CONTROLLING NODE CONDITION OSCILLATION

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node condition oscillates between **true** and **false**, which can cause problems for the scheduler.

To prevent this oscillation, set the **eviction-pressure-transition-period** parameter to control how long the node must wait before transitioning out of a pressure condition.

1. Edit or add the parameter to the **kubeletArguments** section of the node configuration file (the `/etc/origin/node/node-config.yaml`) using a set of **<resource_type>= <resource_quantity>** pairs.

```
kubeletArguments:
  eviction-pressure-transition-period="5m"
```

+ The node toggles the condition back to **false** when the node has not observed an eviction threshold being met for the specified pressure condition for the specified period.

+



NOTE

Use the default value (5 minutes) before doing any adjustments. The default choice is intended to allow the system to stabilize, and to prevent the scheduler from assigning new pods to the node before it has settled.

1. Restart the OpenShift Container Platform services for the changes to take effect:

```
# systemctl restart atomic-openshift-node
```

27.5. RECLAIMING NODE-LEVEL RESOURCES

If an eviction criteria is satisfied, the node initiates the process of reclaiming the pressured resource until the signal goes below the defined threshold. During this time, the node does not support scheduling any new pods.

The node attempts to reclaim node-level resources prior to evicting end-user pods, based on whether the host system has a dedicated **imagefs** configured for the container runtime.

With Imagefs

If the host system has **imagefs**:

- If the **nodefs** file system meets eviction thresholds, the node frees up disk space in the following order:
 - Delete dead pods/containers
- If the **imagefs** file system meets eviction thresholds, the node frees up disk space in the following order:
 - Delete all unused images

Without Imagefs

If the host system does not have **imagefs**:

- If the **nodefs** file system meets eviction thresholds, the node frees up disk space in the following order:
 - Delete dead pods/containers
 - Delete all unused images

27.6. UNDERSTANDING POD EVICTION

If an eviction threshold is met and the grace period is passed, the node initiates the process of evicting pods until the signal goes below the defined threshold.

The node ranks pods for eviction by their quality of service, and, among those with the same quality of service, by the consumption of the starved compute resource relative to the pod's scheduling request.

Each QOS level has an OOM score, which the Linux out-of-memory tool (OOM killer) uses to determine which pods to kill. See [Understanding Quality of Service and Out of Memory Killer](#) below.

The following table lists each QOS level and the associated OOM score.

Table 27.2. Quality of Service Levels

Quality of Service	Description
Guaranteed	Pods that consume the highest amount of the starved resource relative to their request are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.
Burstable	Pods that consume the highest amount of the starved resource relative to their request for that resource are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.

Quality of Service	Description
BestEffort	Pods that consume the highest amount of the starved resource are failed first.

A **Guaranteed** pod will never be evicted because of another pod's resource consumption unless a system daemon (such as node, **docker**, **journald**) is consuming more resources than were reserved using **system-reserved**, or **kube-reserved** allocations or if the node has only **Guaranteed** pods remaining.

If the node has only **Guaranteed** pods remaining, the node evicts a **Guaranteed** pod that least impacts node stability and limits the impact of the unexpected consumption to other **Guaranteed** pods.

Local disk is a **BestEffort** resource. If necessary, the node evicts pods one at a time to reclaim disk when **DiskPressure** is encountered. The node ranks pods by quality of service. If the node is responding to inode starvation, it will reclaim inodes by evicting pods with the lowest quality of service first. If the node is responding to lack of available disk, it will rank pods within a quality of service that consumes the largest amount of local disk, and evict those pods first.

27.6.1. Understanding Quality of Service and Out of Memory Killer

If the node experiences a system out of memory (OOM) event before it is able to reclaim memory, the node depends on the OOM killer to respond.

The node sets a **oom_score_adj** value for each container based on the quality of service for the pod.

Table 27.3. Quality of Service Levels

Quality of Service	oom_score_adj Value
Guaranteed	-998
Burstable	$\min(\max(2, 1000 - (1000 * \text{memoryRequestBytes}) / \text{machineMemoryCapacityBytes}), 999)$
BestEffort	1000

If the node is unable to reclaim memory prior to experiencing a system OOM event, the **oom_killer** calculates an **oom_score**:

```
% of node memory a container is using + `oom_score_adj` = `oom_score`
```

The node then kills the container with the highest score.

Containers with the lowest quality of service that are consuming the largest amount of memory relative to the scheduling request are failed first.

Unlike pod eviction, if a pod container is OOM failed, it can be restarted by the node based on the node restart policy.

27.7. UNDERSTANDING THE POD SCHEDULER AND OOR CONDITIONS

The scheduler views node conditions when placing additional pods on the node. For example, if the node has an eviction threshold like the following:

```
eviction-hard is "memory.available<500Mi"
```

and available memory falls below 500Mi, the node reports a value in **Node.Status.Conditions** as **MemoryPressure** as true.

Table 27.4. Node Conditions and Scheduler Behavior

Node Condition	Scheduler Behavior
MemoryPressure	If a node reports this condition, the scheduler will not place BestEffort pods on that node.
DiskPressure	If a node reports this condition, the scheduler will not place any additional pods on that node.

27.8. EXAMPLE SCENARIO

Consider the following scenario.

An operator:

- has a node with a memory capacity of **10Gi**;
- wants to reserve 10% of memory capacity for system daemons (kernel, node, etc.);
- wants to evict pods at 95% memory utilization to reduce thrashing and incidence of system OOM.

Implicit in this configuration is the understanding that **system-reserved** should include the amount of memory covered by the eviction threshold.

To reach that capacity, either some pod is using more than its request, or the system is using more than **1Gi**.

If a node has 10 Gi of capacity, and you want to reserve 10% of that capacity for the system daemons (**system-reserved**), perform the following calculation:

```
capacity = 10 Gi
system-reserved = 10 Gi * .1 = 1 Gi
```

The amount of allocatable resources becomes:

```
allocatable = capacity - system-reserved = 9 Gi
```

This means by default, the scheduler will schedule pods that request 9 Gi of memory to that node.

If you want to turn on eviction so that eviction is triggered when the node observes that available memory falls below 10% of capacity for 30 seconds, or immediately when it falls below 5% of capacity, you need the scheduler to see allocatable as 8Gi. Therefore, ensure your system reservation covers the greater of your eviction thresholds.

```
capacity = 10 Gi
eviction-threshold = 10 Gi * .1 = 1 Gi
system-reserved = (10Gi * .1) + eviction-threshold = 2 Gi
allocatable = capacity - system-reserved = 8 Gi
```

Enter the following in the *node-config.yaml*:

```
kubeletArguments:
  system-reserved:
    - "memory=2Gi"
  eviction-hard:
    - "memory.available<.5Gi"
  eviction-soft:
    - "memory.available<1Gi"
  eviction-soft-grace-period:
    - "memory.available=30s"
```

This configuration ensures that the scheduler does not place pods on a node that immediately induce memory pressure and trigger eviction assuming those pods use less than their configured request.

27.9. RECOMMENDED PRACTICE

27.9.1. DaemonSets and Out of Resource Handling

If a node evicts a pod that was created by a DaemonSet, the pod will immediately be recreated and rescheduled back to the same node, because the node has no ability to distinguish a pod created from a DaemonSet versus any other object.

In general, DaemonSets should not create **BestEffort** pods to avoid being identified as a candidate pod for eviction. Instead DaemonSets should ideally launch **Guaranteed** pods.

CHAPTER 28. MONITORING AND DEBUGGING ROUTERS

28.1. OVERVIEW

Depending on the underlying implementation, you can monitor a running [router](#) in multiple ways. This topic discusses the HAProxy template router and the components to check to ensure its health.

28.2. VIEWING STATISTICS

The HAProxy router exposes a web listener for the HAProxy statistics. Enter the router's public IP address and the correctly configured port (**1936** by default) to view the statistics page, and enter the administrator password when prompted. This password and port are configured during the router installation, but they can be found by viewing the *haproxy.config* file on the container.

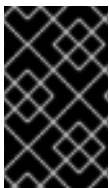
28.3. DISABLING STATISTICS VIEW

By default the HAProxy statistics are exposed on port **1936** (with a password protected account). To disable exposing the HAProxy statistics, specify **0** as the stats port number.

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

Note: HAProxy will still collect and store statistics, it would just *not* expose them via a web listener. You can still get access to the statistics by sending a request to the HAProxy AF_UNIX socket inside the HAProxy Router container.

```
$ cmd="echo 'show stat' | socat - UNIX-
CONNECT:/var/lib/haproxy/run/haproxy.sock"
$ routerPod=$(oc get pods --selector="router=router" \
--template="{{with index .items 0}}{{.metadata.name}}{{end}}")
$ oc exec $routerPod -- bash -c "$cmd"
```



IMPORTANT

For security purposes, the **oc exec** command does not work when accessing privileged containers. Instead, you can SSH into a node host, then use the **docker exec** command on the desired container.

28.4. VIEWING LOGS

To view a router log, run the **oc logs** command on the pod. Since the router is running as a plug-in process that manages the underlying implementation, the log is for the plug-in, not the actual HAProxy log.

To view the logs generated by HAProxy, start a syslog server and pass the location to a router pod using the following environment variables.

Table 28.1. Router Syslog Variables

Environment Variable	Description
ROUTER_SYSLOG_ADDRESS	The IP address of the syslog server. Port 514 is the default if no port is specified.
ROUTER_LOG_LEVEL	Optional. Set to change the HAProxy log level. If not set, the default log level is warning . This can be changed to any log level that HAProxy supports.
ROUTER_SYSLOG_FORMAT	Optional. Set to define customized HAProxy log format. This can be changed to any log format string that HAProxy accepts.

To set a running router pod to send messages to a syslog server:

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=<dest_ip:dest_port>
ROUTER_LOG_LEVEL=<level>
```

For example, the following sets HAProxy to send logs to 127.0.0.1 with the default port **514** and changes the log level to **debug**.

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1
ROUTER_LOG_LEVEL=debug
```

28.5. VIEWING THE ROUTER INTERNALS

routes.json

Routes are processed by the HAProxy router, and are stored both in memory, on disk, and in the HAProxy configuration file. The internal route representation, which is passed to the template to generate the HAProxy configuration file, is found in the `/var/lib/haproxy/router/routes.json` file. When troubleshooting a routing issue, view this file to see the data being used to drive configuration.

HAProxy configuration

You can find the HAProxy configuration and the backends that have been created for specific routes in the `/var/lib/haproxy/conf/haproxy.config` file. The mapping files are found in the same directory. The helper frontend and backends use mapping files when mapping incoming requests to a backend.

Certificates

Certificates are stored in two places:

- Certificates for edge terminated and re-encrypt terminated routes are stored in the `/var/lib/haproxy/router/certs` directory.
- Certificates that are used for connecting to backends for re-encrypt terminated routes are stored in the `/var/lib/haproxy/router/cacerts` directory.

The files are keyed by the namespace and name of the route. The key, certificate, and CA certificate are concatenated into a single file. You can use [OpenSSL](#) to view the contents of these files.

CHAPTER 29. HIGH AVAILABILITY

29.1. OVERVIEW

This topic describes setting up high availability for pods and services on your OpenShift Container Platform cluster.

IP failover manages a pool of Virtual IP (VIP) addresses on a set of nodes. Every VIP in the set will be serviced by a node selected from the set. As long as a single node is available, the VIPs will be served. There is no way to explicitly distribute the VIPs over the nodes, so there may be nodes with no VIPs and other nodes with many VIPs. If there is only one node, all VIPs will be on it.



NOTE

The VIPs must be routable from outside the cluster.

IP failover monitors a port on each VIP to determine whether the port is reachable on the node. If the port is not reachable, the VIP will not be assigned to the node. If the port is set to **0**, this check is suppressed. [The `check` script](#) does the needed testing.

IP failover uses **Keepalived** to host a set of externally accessible VIP addresses on a set of hosts. Each VIP is only serviced by a single host at a time. **Keepalived** uses the VRRP protocol to determine which host (from the set of hosts) will service which VIP. If a host becomes unavailable or if the service that **Keepalived** is watching does not respond, the VIP is switched to another host from the set. Thus, a VIP is always serviced as long as a host is available.

When a host running **Keepalived** passes the **check** script, the host can become in the **MASTER** state based on its priority and the priority of the current **MASTER**, as determined by the [preemption strategy](#).

The administrator can provide a script via the `--notify-script=` option, which is called whenever the state changes. **Keepalived** is in **MASTER** state when it is servicing the VIP, in **BACKUP** state when another node is servicing the VIP, or in **FAULT** state when the **check** script fails. The [notify script](#) is called with the new state whenever the state changes.

OpenShift Container Platform supports creation of IP failover deployment configuration, by running the `oc adm ipfailover` command. The IP failover deployment configuration specifies the set of VIP addresses, and the set of nodes on which to service them. A cluster can have multiple IP failover deployment configurations, with each managing its own set of unique VIP addresses. Each node in the IP failover configuration runs an IP failover pod, and this pod runs **Keepalived**.

When using VIPs to access a pod with host networking (e.g. a router), the application pod should be running on all nodes that are running the ipfailover pods. This enables any of the ipfailover nodes to become the master and service the VIPs when needed. If application pods are not running on all nodes with ipfailover, either some ipfailover nodes will never service the VIPs or some application pods will never receive any traffic. Use the same selector and replication count, for both ipfailover and the application pods, to avoid this mismatch.

While using VIPs to access a service, any of the nodes can be in the ipfailover set of nodes, since the service is reachable on all nodes (no matter where the application pod is running). Any of the ipfailover nodes can become master at any time. The service can either use external IPs and a service port or it can use a nodePort.

When using external IPs in the service definition the VIPs are set to the external IPs and the ipfailover monitoring port is set to the service port. A nodePort is open on every node in the cluster and the service will load balance traffic from whatever node currently supports the VIP. In this case, the ipfailover

monitoring port is set to the nodePort in the service definition.



IMPORTANT

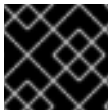
Setting up a nodePort is a privileged operation.



IMPORTANT

Even though a service VIP is highly available, performance can still be affected. **keepalived** makes sure that each of the VIPs is serviced by some node in the configuration, and several VIPs may end up on the same node even when other nodes have none. Strategies that externally load balance across a set of VIPs may be thwarted when ipfailover puts multiple VIPs on the same node.

When you use ingressIP, you can set up ipfailover to have the same VIP range as the ingressIP range. You can also disable the monitoring port. In this case, all the VIPs will appear on same node in the cluster. Any user can set up a service with an ingressIP and have it highly available.

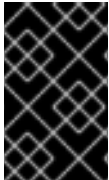


IMPORTANT

There are a maximum of 255 VIPs in the cluster.

29.2. CONFIGURING IP FAILOVER

Use the **oc adm ipfailover** command with suitable [options](#), to create ipfailover deployment configuration.



IMPORTANT

Currently, ipfailover is not compatible with cloud infrastructures. For AWS, an Elastic Load Balancer (ELB) can be used to make OpenShift Container Platform highly available, [using the AWS console](#).

As an administrator, you can configure ipfailover on an entire cluster, or on a subset of nodes, as defined by the label selector. You can also configure multiple IP failover deployment configurations in your cluster, where each one is independent of the others. The **oc adm ipfailover** command creates an ipfailover deployment configuration which ensures that a failover pod runs on each of the nodes matching the constraints or the label used. This pod runs **Keepalived** which uses VRRP (Virtual Router Redundancy Protocol) among all the **Keepalived** daemons to ensure that the service on the watched port is available, and if it is not, **Keepalived** will automatically float the VIPs.

For production use, make sure to use a **--selector=<label>** with at least two nodes to select the nodes. Also, set a **--replicas=<n>** value that matches the number of nodes for the given labeled selector.

The **oc adm ipfailover** command includes command line options that set environment variables that control **Keepalived**. The [environment variables](#) start with **OPENSIFT_HA_*** and they can be changed as needed.

For example, the command below will create an IP failover configuration on a selection of nodes labeled **router=us-west-ha** (on 4 nodes with 7 virtual IPs monitoring a service listening on port 80, such as the router process).

```
$ oc adm ipfailover --selector="router=us-west-ha" \
  --virtual-ips="1.2.3.4,10.1.1.100-104,5.6.7.8" \
  --watch-port=80 --replicas=4 --create
```

29.2.1. Virtual IP Addresses

Keepalived manages a set of virtual IP addresses. The administrator must make sure that all these addresses:

- Are accessible on the configured hosts from outside the cluster.
- Are not used for any other purpose within the cluster.

Keepalived on each node determines whether the needed service is running. If it is, VIPs are supported and **Keepalived** participates in the negotiation to determine which node will serve the VIP. For a node to participate, the service must be listening on the watch port on a VIP or the check must be disabled.



NOTE

Each VIP in the set may end up being served by a different node.

29.2.2. Check and Notify Scripts

Keepalived monitors the health of the application by periodically running an optional user supplied check script. For example, the script can test a web server by issuing a request and verifying the response.

The script is provided through the `--check-script=<script>` option to the `oc adm ipfailover` command. The script must exit with `0` for **PASS** or `1` for **FAIL**.

By default, the check is done every two seconds, but can be changed using the `--check-interval=<seconds>` option.

When a check script is not provided, a simple default script is run that tests the [TCP connection](#). This default test is suppressed when the monitor port is `0`.

For each VIP, **keepalived** keeps the state of the node. The VIP on the node may be in **MASTER**, **BACKUP**, or **FAULT** state. All VIPs on the node that are not in the **FAULT** state participate in the negotiation to decide which will be **MASTER** for the VIP. All of the losers enter the **BACKUP** state. When the **check** script on the **MASTER** fails, the VIP enters the **FAULT** state and triggers a renegotiation. When the **BACKUP** fails, the VIP enters the **FAULT** state. When the **check** script passes again on a VIP in the **FAULT** state, it exits **FAULT** and negotiates for **MASTER**. The resulting state is either **MASTER** or **BACKUP**.

The administrator can provide an optional **notify** script, which is called whenever the state changes. **Keepalived** passes the following three parameters to the script:

- **\$1** - "GROUP"|"INSTANCE"
- **\$2** - Name of the group or instance
- **\$3** - The new state ("MASTER"|"BACKUP"|"FAULT")

These scripts run in the IP failover pod and use the pod's file system, not the host file system. The options require the full path to the script. The administrator must make the script available in the pod to

extract the results from running the **notify** script. The recommended approach for providing the scripts is to use a [ConfigMap](#).

The full path names of the **check** and **notify** scripts are added to the **keepalived** configuration file, **/etc/keepalived/keepalived.conf**, which is loaded every time **keepalived** starts. The scripts can be added to the pod with a ConfigMap as follows.

1. Create the desired script and create a ConfigMap to hold it. The script has no input arguments and must return **0** for **OK** and **1** for **FAIL**.

The check script, **mycheckscript.sh**:

```
#!/bin/bash
# Whatever tests are needed
# E.g., send request and verify response
exit 0
```

2. Create the ConfigMap:

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. There are two approaches to adding the script to the pod: use **oc** commands or edit the deployment configuration. In both cases, the **defaultMode** for the mounted **configMap** files must allow execution. A value of **0755** (**493** decimal) is typical.

- a. Using **oc** commands:

```
$ oc env dc/ipf-ha-router \
    OPENSIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
$ oc volume dc/ipf-ha-router --add --overwrite \
    --name=config-volume \
    --mount-path=/etc/keepalive \
    --source='{ "configMap": { "name": "mycustomcheck",
    "defaultMode": 493}}'
```

- b. Editing the **ipf-ha-router** deployment configuration:

- i. Use **oc edit dc ipf-ha-router** to edit the router deployment configuration with a text editor.

```
...
spec:
  containers:
    - env:
      - name: OPENSIFT_HA_CHECK_SCRIPT ①
        value: /etc/keepalive/mycheckscript.sh
    ...
    volumeMounts: ②
      - mountPath: /etc/keepalive
        name: config-volume
    dnsPolicy: ClusterFirst
    ...
  volumes: ③
    - configMap:
        defaultMode: 0755 ④
```

```

    name: customrouter
    name: config-volume
  ...

```

- 1 In the `spec.container.env` field, add the `OPENSIFT_HA_CHECK_SCRIPT` environment variable to point to the mounted script file.
- 2 Add the `spec.container.volumeMounts` field to create the mount point.
- 3 Add a new `spec.volumes` field to mention the ConfigMap.
- 4 This sets execute permission on the files. When read back, it will be displayed in decimal (**493**).

ii. Save the changes and exit the editor. This restarts **ipf-ha-router**.

29.2.3. VRRP Preemption

When a host leaves the **FAULT** state by passing the check script, the host becomes a **BACKUP** if the new host has lower priority than the host currently in the **MASTER** state. However, if it has a higher priority, the preemption strategy determines its role in the cluster.

The **nopreempt** strategy does not move **MASTER** from the lower priority host to the higher priority host. With **preempt 300**, the default, **keepalived** waits the specified 300 seconds and moves **MASTER** to the higher priority host.

To specify preemption:

- a. When creating ipfailover using the **preemption-strategy**:

```

$ oc adm ipfailover --preempt-strategy=nopreempt \
  ...

```

- b. Setting the variable using the **oc set env** command:

```

$ oc set env dc/ipf-ha-router \
  --overwrite=true \
  OPENSIFT_HA_PREEMPTION=nopreempt

```

- c. Using **oc edit dc ipf-ha-router** to edit the router deployment configuration:

```

...
spec:
  containers:
  - env:
    - name: OPENSIFT_HA_PREEMPTION 1
      value: nopreempt
  ...

```

29.2.4. Keepalived Multicast

OpenShift Container Platform's IP failover internally uses **keepalived**.

**IMPORTANT**

Ensure that **multicast** is enabled on the nodes labeled above and they can accept network traffic for 224.0.0.18 (the VRRP multicast IP address).

Before starting the **keepalived** daemon, the startup script verifies the **iptables** rule that allows multicast traffic to flow. If there is no such rule, the startup script creates a new rule and adds it to the IP tables configuration. Where this new rule gets added to the IP tables configuration depends on the **--iptables-chain=** option. If there is an **--iptables-chain=** option specified, the rule gets added to the specified chain in the option. Otherwise, the rule is added to the **INPUT** chain.

**IMPORTANT**

The **iptables** rule must be present whenever there is one or more **keepalived** daemon running on the node.

The **iptables** rule can be removed after the last **keepalived** daemon terminates. The rule is not automatically removed.

You can manually manage the **iptables** rule on each of the nodes. It only gets created when none is present (as long as ipfailover is not created with the **--iptables-chain=""** option).

**IMPORTANT**

You must ensure that the manually added rules persist after a system restart.

Be careful since every **keepalived** daemon uses the VRRP protocol over multicast 224.0.0.18 to negotiate with its peers. There must be a different VRRP-id (in the range 0..255) for [each VIP](#).

```
$ for node in openshift-node-{5,6,7,8,9}; do    ssh $node <<EOF

export interface=${interface:-"eth0"}
echo "Check multicast enabled ... ";
ip addr show $interface | grep -i MULTICAST

echo "Check multicast groups ... "
ip maddr show $interface | grep 224.0.0

EOF
done;
```

29.2.5. Command Line Options and Environment Variables

Table 29.1. Command Line Options and Environment Variables

Option	Variable Name	Default	Notes
--------	---------------	---------	-------

Option	Variable Name	Default	Notes
-- watch-port	OPENSIFT_HA_MONITOR_PORT	80	The ipfailover pod tries to open a TCP connection to this port on each VIP. If connection is established, the service is considered to be running. If this port is set to 0, the test always passes.
-- interface	OPENSIFT_HA_NETWORK_INTERFACE		The interface name for ipfailover to use, to send VRRP traffic. By default, eth0 is used.
-- replicas	OPENSIFT_HA_REPLICA_COUNT	2	Number of replicas to create. This must match spec.replicas value in ipfailover deployment configuration.
-- virtual-ips	OPENSIFT_HA_VIRTUAL_IPS		The list of IP address ranges to replicate. This must be provided. (For example, 1.2.3.4-6,1.2.3.9.) See this discussion for more details.
-- vrrp-id-offset	OPENSIFT_HA_VRRP_ID_OFFSET	0	See VRRP ID Offset discussion for more details.
-- iptables-chain	OPENSIFT_HA_IPTABLES_CHAIN	INPUT	The name of the iptables chain, to automatically add an iptables rule to allow the VRRP traffic on. If the value is not set, an iptables rule will not be added. If the chain does not exist, it is not created.
-- check-script	OPENSIFT_HA_CHECK_SCRIPT		Full path name in the pod file system of a script that is periodically run to verify the application is operating. See this discussion for more details.
-- check-interval	OPENSIFT_HA_CHECK_INTERVAL	2	The period, in seconds, that the check script is run.
-- notify-script	OPENSIFT_HA_NOTIFY_SCRIPT		Full path name in the pod file system of a script that is run whenever the state changes. See this discussion for more details.
-- preemption-strategy	OPENSIFT_HA_PREEMPTION	preempt 300	Strategy for handling a new higher priority host. See the VRRP Preemption section for more details.

29.2.6. VRRP ID Offset

Each ipfailover pod managed by the ipfailover deployment configuration (1 pod per node/replica) runs a **keepalived** daemon. As more ipfailover deployment configurations are configured, more pods are created and more daemons join into the common VRRP negotiation. This negotiation is done by all the **keepalived** daemons and it determines which nodes will service which VIPs.

Internally, **keepalived** assigns a unique vrrp-id to each VIP. The negotiation uses this set of vrrp-ids, when a decision is made, the VIP corresponding to the winning vrrp-id is serviced on the winning node.

Therefore, for every VIP defined in the ipfailover deployment configuration, the ipfailover pod must assign a corresponding vrrp-id. This is done by starting at **--vrrp-id-offset** and sequentially assigning the vrrp-ids to the list of VIPs. The vrrp-ids may have values in the range 1..255.

When there are multiple ipfailover deployment configuration care must be taken to specify **--vrrp-id-offset** so that there is room to increase the number of VIPS in the deployment configuration and none of the vrrp-id ranges overlap.

29.2.7. Configuring a Highly-available Service

The following example describes how to set up highly-available **router** and **geo-cache** network services with IP failover on a set of nodes.

1. Label the nodes that will be used for the services. This step can be optional if you run the services on all the nodes in your OpenShift Container Platform cluster and will use VIPs that can float within all nodes in the cluster.

The following example defines a label for nodes that are servicing traffic in the US west geography **ha-svc-nodes=geo-us-west**:

```
$ oc label nodes openshift-node-{5,6,7,8,9} "ha-svc-nodes=geo-us-west"
```

2. Create the service account. You can use ipfailover or when using a router (depending on your environment policies), you can either reuse the **router** service account created previously or a new ipfailover service account.

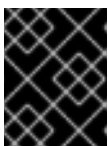
The following example creates a new service account with the name ipfailover in the **default** namespace:

```
$ oc create serviceaccount ipfailover -n default
```

3. Add the ipfailover service account in the **default** namespace to the **privileged** SCC:

```
$ oc adm policy add-scc-to-user privileged
system:serviceaccount:default:ipfailover
```

4. Start the **router** and the **geo-cache** services.



IMPORTANT

Since the ipfailover runs on all nodes from step 1, it is recommended to also run the router/service on all the step 1 nodes.

- a. Start the router with the nodes matching the labels used in the first step. The following example runs five instances using the `ipfailover` service account:

```
$ oc adm router ha-router-us-west --replicas=5 \
  --selector="ha-svc-nodes=geo-us-west" \
  --labels="ha-svc-nodes=geo-us-west" \
  --service-account=ipfailover
```

- b. Run the **geo-cache** service with a replica on each of the nodes. See an [example configuration](#) for running a **geo-cache** service.

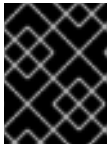


IMPORTANT

Make sure that you replace the **myimages/geo-cache** Docker image referenced in the file with your intended image. Change the number of replicas to the number of nodes in the **geo-cache** label. Check that the label matches the one used in the first step.

```
$ oc create -n <namespace> -f ./examples/geo-cache.json
```

5. Configure `ipfailover` for the **router** and **geo-cache** services. Each has its own VIPs and both use the same nodes labeled with **ha-svc-nodes=geo-us-west** in the first step. Ensure that the number of replicas match the number of nodes listed in the label setup, in the first step.



IMPORTANT

The **router**, **geo-cache**, and `ipfailover` all create deployment configuration and all must have different names.

6. Specify the VIPs and the port number that `ipfailover` should monitor on the desired instances. The `ipfailover` command for the **router**:

```
$ oc adm ipfailover ipf-ha-router-us-west \
  --replicas=5 --watch-port=80 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips="10.245.2.101-105" \
  --iptables-chain="INPUT" \
  --service-account=ipfailover --create
```

The following is the `oc adm ipfailover` command for the **geo-cache** service that is listening on port 9736. Since there are two **ipfailover** deployment configurations, the `--vrrp-id-offset` must be set so that each VIP gets its own offset. In this case, setting a value of **10** means that the **ipf-ha-router-us-west** can have a maximum of 10 VIPs (0-9) since **ipf-ha-geo-cache** is starting at 10.

```
$ oc adm ipfailover ipf-ha-geo-cache \
  --replicas=5 --watch-port=9736 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips=10.245.3.101-105 \
  --vrrp-id-offset=10 \
  --service-account=ipfailover --create
```

In the commands above, there are **ipfailover**, **router**, and **geo-cache** pods on each node. The set of VIPs for each ipfailover configuration must not overlap and they must not be used elsewhere in the external or cloud environments. The five VIP addresses in each example, **10.245.2.101-105** are served by the two ipfailover deployment configurations. IP failover dynamically selects which address is served on which node.

The administrator sets up external DNS to point to the VIP addresses knowing that all the **router** VIPs point to the same **router**, and all the **geo-cache** VIPs point to the same **geo-cache** service. As long as one node remains running, all the VIP addresses are served.

29.2.7.1. Deploy IP Failover Pod

Deploy the ipfailover router to monitor postgresql listening on node port 32439 and the external IP address, as defined in the **postgresql-ingress** service:

```
$ oc adm ipfailover ipf-ha-postgresql \
  --replicas=1 \ 1
  --selector="app-type=postgresql" \ 2
  --virtual-ips=10.9.54.100 \ 3
  --watch-port=32439 \ 4
  --service-account=ipfailover --create
```

- 1 1 Specifies the number of instances to deploy.
- 2 Restricts where the ipfailover is deployed.
- 3 Virtual IP address to monitor.
- 4 Port on which ipfailover will monitor on each node.

29.2.8. Dynamically Updating Virtual IPs for a Highly-available Service

The default deployment strategy for the IP failover service is to recreate the deployment. In order to dynamically update the VIPs for a highly available routing service with minimal or no downtime, you must:

- Update the IP failover service deployment configuration to use a rolling update strategy, and
- Update the **OPENSHIFT_HA_VIRTUAL_IPS** environment variable with the updated list or sets of virtual IP addresses.

The following example shows how to dynamically update the deployment strategy and the virtual IP addresses:

1. Consider an IP failover configuration that was created using the following:

```
$ oc adm ipfailover ipf-ha-router-us-west \
  --replicas=5 --watch-port=80 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips="10.245.2.101-105" \
  --service-account=ipfailover --create
```

2. Edit the deployment configuration:

■

```
$ oc edit dc/ipf-ha-router-us-west
```

- Update the `spec.strategy.type` field from **Recreate** to **Rolling**:

```
spec:
  replicas: 5
  selector:
    ha-svc-nodes: geo-us-west
  strategy:
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    type: Rolling ❶
```

- ❶ Set to **Rolling**.

- Update the `OPENSIFT_HA_VIRTUAL_IPS` environment variable to contain the additional virtual IP addresses:

```
- name: OPENSIFT_HA_VIRTUAL_IPS
  value: 10.245.2.101-105,10.245.2.110,10.245.2.201-205 ❶
```

- ❶ **10.245.2.110, 10.245.2.201-205** have been added to the list.

- Update the external DNS to match the set of VIPs.

29.3. CONFIGURING SERVICE EXTERNALIP AND NODEPORT

The user can assign VIPs as [ExternalIPs](#) in a service. **Keepalived** makes sure that each VIP is served on some node in the ipfailover configuration. When a request arrives on the node, the service that is running on all nodes in the cluster, load balances the request among the service's endpoints.

The [NodePorts](#) can be set to the ipfailover watch port so that **keepalived** can check the application is running. The NodePort is exposed on all nodes in the cluster, therefore it is available to **keepalived** on all ipfailover nodes.

29.4. HIGH AVAILABILITY FOR INGRESSIP

In non-cloud clusters, ipfailover and [ingressIP](#) to a service can be combined. The result is high availability services for users that create services using ingressIP.

The approach is to specify an `ingressIPNetworkCIDR` range and then use the same range in creating the ipfailover configuration.

Since, ipfailover can support up to a maximum of 255 VIPs for the entire cluster, the `ingressIPNetworkCIDR` needs to be `/24` or less.

CHAPTER 30. IPTABLES

30.1. OVERVIEW

There are many system components including OpenShift Container Platform, containers, and software that manage local firewall policies that rely on the kernel iptables configuration for proper network operation. In addition, the iptables configuration of all nodes in the cluster must be correct for networking to work.

All components independently work with iptables without knowledge of how other components are using them. This makes it very easy for one component to break another component's configuration. Further, OpenShift Container Platform and the Docker service assume that iptables remains set up exactly as they have set it up. They may not detect changes introduced by other components and if they do there may be some lag in implementing the fix. In particular, OpenShift Container Platform does monitor and fix problems. However, the Docker service does not.



IMPORTANT

Ensure that any changes you make to the iptables configuration on a node do not impact the operation of OpenShift Container Platform and the Docker service. Also, changes will often need to be made on all nodes in the cluster. Use caution, as iptables is not designed to have multiple concurrent users, and is very easy to break OpenShift Container Platform and Docker networking.

OpenShift Container Platform provides several chains, one of which is specifically intended for administrators to use for their own purposes: **OPENSIFT-ADMIN-OUTPUT-RULES**.

See the discussion of [using iptables rules to limit access to external resources](#) for more information.

The chains, order of the chains, and rules in the kernel iptables must be properly set up on each node in the cluster for OpenShift Container Platform and Docker networking to work properly. There are several tools and services that are commonly used in the system that interact with the kernel iptables and can accidentally impact OpenShift Container Platform and the Docker service.

30.2. IPTABLES

The iptables tool can be used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.

Independent of other use, such as a firewall, OpenShift Container Platform and the the Docker service manage chains in some of the tables. The chains are inserted in specific order and the rules are specific to their needs.

CAUTION

`iptables --flush [chain]` can remove key required configuration. Do not execute this command.

30.3. IPTABLES.SERVICE

The iptables service supports a local network firewall. It assumes total control of the iptables configuration. When it starts, it flushes and restores the complete iptables configuration. The restored

rules are from its configuration file, **/etc/sysconfig/iptables**. The configuration file is not kept up to date during operation, so the dynamically added rules are lost during every restart.



WARNING

Stopping and starting **iptables.service** will destroy configuration that is required by OpenShift Container Platform and Docker. OpenShift Container Platform and Docker are not notified of the change.

```
# systemctl disable iptables.service
# systemctl mask iptables.service
```

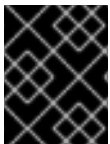
If you need to run **iptables.service**, keep a limited configuration in the configuration file and rely on OpenShift Container Platform and Docker to install their needed rules.

The **iptables.service** configuration is loaded from:

```
/etc/sysconfig/iptables
```

To make permanent rules changes, edit the changes into this file. Do not include Docker or OpenShift Container Platform rules.

After **iptables.service** is started or restarted on a node, the Docker service and **atomic-openshift-node.service** must be restarted to reconstruct the needed iptables configuration.



IMPORTANT

Restarting the Docker service will cause all containers running on the node to be stopped and restarted.

```
# systemctl restart iptables.service
# systemctl restart docker
# systemctl restart atomic-openshift-node.service
```

CHAPTER 31. SECURING BUILDS BY STRATEGY

31.1. OVERVIEW

Builds in OpenShift Container Platform are run in [privileged containers](#) that have access to the Docker daemon socket. As a security measure, it is recommended to limit who can run builds and the strategy that is used for those builds. [Custom builds](#) are inherently less safe than [Source builds](#), given that they can execute any code in the build with potentially full access to the node's Docker socket, and as such are disabled by default. [Docker build](#) permission should also be granted with caution as a vulnerability in the Docker build logic could result in a privileges being granted on the host node.

By default, all users that can create builds are granted permission to use the Docker and Source-to-Image build strategies. Users with [cluster-admin](#) privileges can enable the Custom build strategy, as referenced in the [Restricting Build Strategies to a User Globally](#) section of this page.

You can control who can build with what build strategy using an [authorization policy](#). Each build strategy has a corresponding build subresource. A user must have permission to create a build *and* permission to create on the build strategy subresource in order to create builds using that strategy. Default roles are provided which grant the **create** permission on the build strategy subresource.

Table 31.1. Build Strategy Subresources and Roles

Strategy	Subresource	Role
Docker	builds/docker	system:build-strategy-docker
Source-to-Image	builds/source	system:build-strategy-source
Custom	builds/custom	system:build-strategy-custom
JenkinsPipeline	builds/jenkinspipeline	system:build-strategy-jenkinspipeline

31.2. DISABLING A BUILD STRATEGY GLOBALLY

To prevent access to a particular build strategy globally, log in as a user with [cluster-admin](#) privileges, remove the corresponding role from the **system:authenticated** group, and apply the annotation **openshift.io/reconcile-protect: "true"** to protect them from changes between the API restarts. The following example shows disabling the docker build strategy.

1. Apply the **openshift.io/reconcile-protect** annotation

```
$ oc edit clusterrolebinding system:build-strategy-docker-binding

apiVersion: v1
groupNames:
- system:authenticated
kind: ClusterRoleBinding
metadata:
  annotations:
    openshift.io/reconcile-protect: "true" 1
  creationTimestamp: 2018-08-10T01:24:14Z
```



```

name: system:build-strategy-docker-binding
resourceVersion: "225"
selfLink: /oapi/v1/clusterrolebindings/system%3Abuild-strategy-
docker-binding
uid: 17b1f3d4-9c3c-11e8-be62-0800277d20bf
roleRef:
  name: system:build-strategy-docker
subjects:
- kind: SystemGroup
  name: system:authenticated
userNames:
- system:serviceaccount:management-infra:management-admin

```

- 1 Change the **openshift.io/reconcile-protect** annotation's value to **"true"**. By default, it is set to **"false"**.

2. Remove the role:

```

$ oc adm policy remove-cluster-role-from-group system:build-
strategy-docker system:authenticated

```

In versions prior to 3.2, the build strategy subresources were included in the **admin** and **edit** roles.

Ensure the build strategy subresources are also removed from these roles:

```

$ oc edit clusterrole admin
$ oc edit clusterrole edit

```

For each role, remove the line that corresponds to the resource of the strategy to disable.

Disable the Docker Build Strategy for admin

```

kind: ClusterRole
metadata:
  name: admin
...
rules:
- resources:
  - builds/custom
  - builds/docker 1
  - builds/source
...
...

```

- 1 Delete this line to disable Docker builds globally for users with the **admin** role.

31.3. RESTRICTING BUILD STRATEGIES TO A USER GLOBALLY

To allow only a set of specific users to create builds with a particular strategy:

1. [Disable global access to the build strategy.](#)

2. Assign the role corresponding to the build strategy to a specific user. For example, to add the **system:build-strategy-docker** cluster role to the user **devuser**:

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```



WARNING

Granting a user access at the cluster level to the **builds/docker** subresource means that the user will be able to create builds with the Docker strategy in any project in which they can create builds.

31.4. RESTRICTING BUILD STRATEGIES TO A USER WITHIN A PROJECT

Similar to granting the build strategy role to a user globally, to allow only a set of specific users within a project to create builds with a particular strategy:

1. [Disable global access to the build strategy.](#)
2. Assign the role corresponding to the build strategy to a specific user within a project. For example, to add the **system:build-strategy-docker** role within the project **devproject** to the user **devuser**:

```
$ oc adm policy add-role-to-user system:build-strategy-docker devuser -n devproject
```

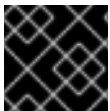
CHAPTER 32. RESTRICTING APPLICATION CAPABILITIES USING SECCOMP

32.1. OVERVIEW

Seccomp (secure computing mode) is used to restrict the set of system calls applications can make, allowing cluster administrators greater control over the security of workloads running in OpenShift Container Platform.

Seccomp support is achieved via two annotations in the pod configuration:

- **seccomp.security.alpha.kubernetes.io/pod**: profile applies to all containers in the pod that do not override
- **container.seccomp.security.alpha.kubernetes.io/<container_name>**: container-specific profile override



IMPORTANT

Containers are run with **unconfined** seccomp settings by default.

For detailed design information, refer to the [seccomp design document](#).

32.2. ENABLING SECCOMP

Seccomp is a feature of the Linux kernel. To ensure seccomp is enabled on your system, run:

```
$ cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

32.3. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR SECCOMP

A seccomp profile is a json file providing syscalls and the appropriate action to take when a syscall is invoked.

1. Create the seccomp profile.

The [default profile](#) is sufficient in many cases, but the cluster administrator must define the security constraints of an individual system.

To create your own custom profile, create a file on every node in the **seccomp-profile-root** directory.

If you are using the default **docker/default** profile, you do not need to create one.

2. Configure your nodes to use the **seccomp-profile-root** where your profiles will be stored. In the **node-config.yaml** via the **kubeletArguments**:

```
kubeletArguments:
  seccomp-profile-root:
    - "/your/path"
```

- Restart the node service to apply the changes:

```
# systemctl restart atomic-openshift-node
```

- In order to control which profiles may be used, and to set the default profile, [configure your SCC](#) via the **seccompProfiles** field. The first profile will be used as a default. The allowable formats of the **seccompProfiles** field include:

- **docker/default**: the default profile for the container runtime (no profile required)
- **unconfined**: unconfined profile, and disables seccomp
- **localhost/<profile-name>**: the profile installed to the node's local seccomp profile root
For example, if you are using the default **docker/default** profile, configure your SCC with:

```
seccompProfiles:  
- docker/default
```

32.4. CONFIGURING OPENSIFT CONTAINER PLATFORM FOR A CUSTOM SECCOMP PROFILE

To ensure pods in your cluster run with a custom profile:

- Create the seccomp profile in **seccomp-profile-root**.
- Configure **seccomp-profile-root**:

```
kubeletArguments:  
  seccomp-profile-root:  
    - "/your/path"
```

- Restart the node service to apply the changes:

```
# systemctl restart atomic-openshift-node
```

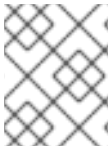
- Configure your SCC:

```
seccompProfiles:  
- localhost/<profile-name>
```

CHAPTER 33. SYSCTLs

33.1. OVERVIEW

Sysctl settings are exposed via Kubernetes, allowing users to modify certain kernel parameters at runtime for namespaces within a container. Only sysctls that are namespaced can be set independently on pods; if a sysctl is not namespaced (called *node-level*), it cannot be set within OpenShift Container Platform. Moreover, only those sysctls considered *safe* are whitelisted by default; other *unsafe* sysctls can be manually enabled on the node to be available to the user.



NOTE

As of OpenShift Container Platform 3.3.1, sysctl support is a feature in [Technology Preview](#).

33.2. UNDERSTANDING SYSCTLs

In Linux, the sysctl interface allows an administrator to modify kernel parameters at runtime. Parameters are available via the */proc/sys/* virtual process file system. The parameters cover various subsystems such as:

- kernel (common prefix: **kernel.**)
- networking (common prefix: **net.**)
- virtual memory (common prefix: **vm.**)
- MDADM (common prefix: **dev.**)

More subsystems are described in [Kernel documentation](#). To get a list of all parameters, you can run:

```
$ sudo sysctl -a
```

33.3. NAMESPACED VERSUS NODE-LEVEL SYSCTLs

A number of sysctls are *namespaced* in today's Linux kernels. This means that they can be set independently for each pod on a node. Being namespaced is a requirement for sysctls to be accessible in a pod context within Kubernetes.

The following sysctls are known to be namespaced:

- **kernel.shm***
- **kernel.msg***
- **kernel.sem**
- **fs.mqueue.***
- **net.***

Sysctls that are not namespaced are called *node-level* and must be set manually by the cluster administrator, either by means of the underlying Linux distribution of the nodes (e.g., via */etc/sysctls.conf*) or using a DaemonSet with privileged containers.

**NOTE**

Consider marking nodes with special sysctls as tainted. Only schedule pods onto them that need those sysctl settings. Use the [Kubernetes taints and toleration feature](#) to implement this.

33.4. SAFE VERSUS UNSAFE SYSCTLS

Sysctls are grouped into *safe* and *unsafe* sysctls. In addition to proper namespacing, a safe sysctl must be properly isolated between pods on the same node. This means that setting a safe sysctl for one pod:

- must not have any influence on any other pod on the node,
- must not allow to harm the node's health, and
- must not allow to gain CPU or memory resources outside of the resource limits of a pod.

By far, most of the namespaced sysctls are not necessarily considered safe.

Currently, OpenShift Container Platform supports, or whitelists, the following sysctls in the safe set:

- ***kernel.shm_rmid_forced***
- ***net.ipv4.ip_local_port_range***
- ***net.ipv4.tcp_syncookies***

This list will be extended in future versions when the kubelet supports better isolation mechanisms.

All safe sysctls are enabled by default. All unsafe sysctls are disabled by default and must be allowed manually by the cluster administrator on a per-node basis. Pods with disabled unsafe sysctls will be scheduled, but will fail to launch.

**WARNING**

Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems like wrong behavior of containers, resource shortage, or complete breakage of a node.

33.5. ENABLING UNSAFE SYSCTLS

With the warning above in mind, the cluster administrator can allow certain unsafe sysctls for very special situations, e.g., high-performance or real-time application tuning.

If you want to use unsafe sysctls, cluster administrators must enable them individually on nodes. Only namespaced sysctls can be enabled this way.

1. Use the **kubeletArguments** field in the **/etc/origin/node/node-config.yaml** file, as described in [Configuring Node Resources](#), to set the desired unsafe sysctls:

```
kubeletArguments:
  experimental-allowed-unsafe-sysctls:
    - "kernel.msg*,net.ipv4.route.min_pmtu"
```

2. Restart the node service to apply the changes:

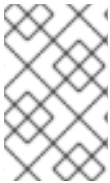
```
# systemctl restart atomic-openshift-node
```

33.6. SETTING SYSCTLS FOR A POD

Sysctls are set on pods using annotations. They apply to all containers in the same pod.

Here is an example, with different annotations for safe and unsafe sysctls:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  annotations:
    security.alpha.kubernetes.io/sysctls: kernel.shm_rmid_forced=1
    security.alpha.kubernetes.io/unsafe-sysctls:
      net.ipv4.route.min_pmtu=1000,kernel.msgmax=1 2 3
spec:
  ...
```



NOTE

A pod with the unsafe sysctls specified above will fail to launch on any node that has not enabled those two unsafe sysctls explicitly. As with node-level sysctls, use the [taints and toleration feature](#) or [labels on nodes](#) to schedule those pods onto the right nodes.

CHAPTER 34. ENCRYPTING DATA AT DATASTORE LAYER

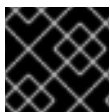
34.1. OVERVIEW

This topic reviews how to enable and configure encryption of secret data at the datastore layer. While the examples use the **secrets** resource, any resource can be encrypted, such as **configmaps**.



WARNING

This is an alpha feature and may change in future.



IMPORTANT

etcd v3 or later is required in order to use this feature.

34.2. CONFIGURATION AND DETERMINING WHETHER ENCRYPTION IS ALREADY ENABLED

To activate data encryption, pass the `--experimental-encryption-provider-config` argument to the Kubernetes API server:

Excerpt of *master-config.yaml*

```
kubernetesMasterConfig:
  apiServerArguments:
    experimental-encryption-provider-config:
      - /path/to/encryption-config.yaml
```

For more information about *master-config.yaml* and its format, see the [Master Configuration Files](#) topic.

34.3. UNDERSTANDING THE ENCRYPTION CONFIGURATION

Encryption configuration file with all available providers

```
kind: EncryptionConfig
apiVersion: v1
resources: ①
  - resources: ②
    - secrets
  providers: ③
    - aescbc: ④
      keys:
        - name: key1 ⑤
          secret: c2VjcmV0IGlzlIHNlY3VyZQ== ⑥
        - name: key2
          secret: dGhpcyBpcyBwYXNzd29yZA==
```



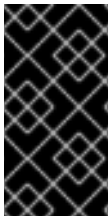
```

- secretbox:
  keys:
    - name: key1
      secret: YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY=
- aesgcm:
  keys:
    - name: key1
      secret: c2VjcmV0IGlzIHNlY3VyZQ==
    - name: key2
      secret: dGhpcyBpcyBwYXNzd29yZA==
- identity: {}

```

- 1 Each **resources** array item is a separate configuration and contains a complete configuration.
- 2 The **resources.resources** field is an array of Kubernetes resource names (**resource** or **resource.group**) that should be encrypted.
- 3 The **providers** array is an ordered [list of the possible encryption providers](#). Only one provider type can be specified per entry (**identity** or **aescbc** can be provided, but not both in the same item).
- 4 The first provider in the list is used to encrypt resources going into storage.
- 5 Arbitrary name of the secret.
- 6 Base64 encoded random key. Different providers have different key lengths. See instructions on [how to generate the key](#).

When reading resources from storage, each provider that matches the stored data attempts to decrypt the data in order. If no provider can read the stored data due to a mismatch in format or secret key, an error is returned, which prevents clients from accessing that resource.



IMPORTANT

If any resource is not readable via the encryption configuration (because keys were changed), the only recourse is to delete that key from the underlying etcd directly. Calls attempting to read that resource will fail until it is deleted or a valid decryption key is provided.

34.3.1. Available Providers

Name	Encryption	Strength	Speed	Key Length	Other Considerations
identity	None	N/A	N/A	N/A	Resources written as-is without encryption. When set as the first provider, the resource will be decrypted as new values are written.
aescbc	AES-CBC with PKCS#7 padding	Strongest	Fast	32-byte	The recommended choice for encryption, but may be slightly slower than secretbox .

Name	Encryption	Strength	Speed	Key Length	Other Considerations
secretbox	XSalsa20 and Poly1305	Strong	Faster	32-byte	A newer standard and may not be considered acceptable in environments that require high levels of review.
aesgcm	AES-GCM with a random initialization vector (IV)	Must be rotated every 200,000 writes	Fastest	16, 24, or 32-byte	Is not recommended for use except when an automated key rotation scheme is implemented.

Each provider supports multiple keys. The keys are tried in order for decryption. If the provider is the first provider, the first key is used for encryption.



NOTE

Kubernetes has no proper nonce generator and uses a random IV as nonce for AES-GCM. Since AES-GCM requires a proper nonce to be secure, AES-GCM is not recommended. The 200,000 write limit just limits the possibility of a fatal nonce misuse to a reasonable low margin.

34.4. ENCRYPTING DATA

Create a new encryption configuration file.

```
kind: EncryptionConfig
apiVersion: v1
resources:
- resources:
- secrets
providers:
- aescbc:
keys:
- name: key1
secret: <BASE 64 ENCODED SECRET>
- identity: {}
```

To create a new secret:

1. Generate a 32-byte random key and base64 encode it. For example, on Linux and macOS use:

```
$ head -c 32 /dev/urandom | base64
```

**IMPORTANT**

The encryption key must be generated with an appropriate cryptographically secure random number generator like `/dev/urandom`. For example, `math/random` from Golang or `random.random()` from Python are not suitable.

2. Place that value in the **secret** field.

3. Restart the API server:

```
# systemctl restart atomic-openshift-master-api
```

**IMPORTANT**

The encryption provider configuration file contains keys that can decrypt content in etcd, so you must properly restrict permissions on masters so only the user who runs the master API server can read it.

34.5. VERIFYING THAT DATA IS ENCRYPTED

Data is encrypted when written to etcd. After restarting the API server, any newly created or updated secrets should be encrypted when stored. To check, you can use the **etcdctl** command line program to retrieve the contents of your secret.

1. Create a new secret called **secret1** in the **default** namespace:

```
$ oc create secret generic secret1 -n default --from-literal=mykey=mydata
```

2. Using the **etcdctl** command line, read that secret out of etcd:

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields [...] | grep Value
```

[...] must be the additional arguments for connecting to the etcd server.

The final command will look similar to:

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields \
--cacert=/var/lib/origin/openshift.local.config/master/ca.crt \
--key=/var/lib/origin/openshift.local.config/master/master.etcd-client.key \
--cert=/var/lib/origin/openshift.local.config/master/master.etcd-client.crt \
--endpoints 'https://127.0.0.1:4001' | grep Value
```

3. Verify that the output of the command above is prefixed with **k8s:enc:aescbc:v1:** which indicates the **aescbc** provider has encrypted the resulting data.
4. Verify the secret is correctly decrypted when retrieved via the API:

```
$ oc get secret secret1 -n default -o yaml | grep mykey
```

This should match **mykey: bXIkYXRh**.

34.6. ENSURE ALL SECRETS ARE ENCRYPTED

Since secrets are encrypted when written, performing an update on a secret will encrypt that content.

```
$ oc adm migrate storage --include=secrets --confirm
```

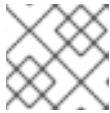
This command reads all secrets, then updates them to apply server-side encryption. If an error occurs due to a conflicting write, retry the command.

For larger clusters, you can subdivide the secrets by namespace or script an update.

34.7. ROTATING A DECRYPTION KEY

Changing the secret without incurring downtime requires a multi-step operation, especially in the presence of a highly available deployment where multiple API servers are running.

1. Generate a new key and add it as the second key entry for the current provider on all servers.
2. Restart all API servers to ensure each server can decrypt using the new key.



NOTE

If using a single API server, you can skip this step.

```
# systemctl restart atomic-openshift-master-api
```

3. Make the new key the first entry in the **keys** array so that it is used for encryption in the configuration.
4. Restart all API servers to ensure each server now encrypts using the new key.

```
# systemctl restart atomic-openshift-master-api
```

5. Run the following to encrypt all existing secrets with the new key:

```
$ oc adm migrate storage --include=secrets --confirm
```

6. After you back up etcd with the new key in use and update all secrets, remove the old decryption key from the configuration.

34.8. DECRYPTING DATA

To disable encryption at the datastore layer:

1. Place the **identity** provider as the first entry in the configuration:

```
kind: EncryptionConfig
apiVersion: v1
resources:
- resources:
```

```
- secrets
providers:
- identity: {}
- aescbc:
  keys:
  - name: key1
    secret: <BASE 64 ENCODED SECRET>
```

1. Restart all API servers:

```
# systemctl restart atomic-openshift-master-api
```

2. Run the following to force all secrets to be decrypted:

```
$ oc adm migrate storage --include=secrets --confirm
```

CHAPTER 35. ENCRYPTING HOSTS WITH IPSEC

35.1. OVERVIEW

IPsec protects traffic in an OpenShift Container Platform cluster by encrypting the communication between all master and node hosts that communicate using the Internet Protocol (IP).

This topic shows how to secure communication of an entire IP subnet from which the OpenShift Container Platform hosts receive their IP addresses, including all cluster management and pod data traffic.



NOTE

Because OpenShift Container Platform management traffic uses HTTPS, enabling IPsec encrypts management traffic a second time.



IMPORTANT

This procedure should be repeated on each master host, then node host, in your cluster. Hosts that do not have IPsec enabled will not be able to communicate with a host that does.

35.2. ENCRYPTING HOSTS

35.2.1. Step 1: Prerequisites

At this time, **libreswan** version 3.15 is the latest version supported on Red Hat Enterprise Linux 7. Ensure that **libreswan** 3.15 or later is installed on cluster hosts. If [opportunistic group functionality](#) is required, then **libreswan** version 3.19 or later is required.

[Configure the SDN MTU](#) to allow space for the IPsec header. In the configuration described here IPsec requires 62 bytes. If the cluster is operating on an ethernet network with an MTU of 1500 then the SDN MTU should be 1388, to allow for the overhead of IPsec and the SDN encapsulation.

After modifying the MTU in the OpenShift Container Platform configuration, the SDN must be made aware of the change by removing the SDN interface and restarting the OpenShift Container Platform node process.

```
# systemctl stop atomic-openshift-node
# ovs-vsctl del-br br0
# systemctl start atomic-openshift-node
```

35.2.2. Step 2: Certificates

By default, OpenShift Container Platform secures cluster management communication with mutually authenticated HTTPS communication. This means that both the client (for example, an OpenShift Container Platform node) and the server (for example, an OpenShift Container Platform api-server) send each other their certificates, which are checked against a known certificate authority (CA). These certificates are generated at cluster set up time and typically live on each host.

These certificates can also be used to secure pod communications with IPsec. You need three files on each host:

- Cluster CA file
 - Host client certificate file
 - Host private key file
1. Determine what the certificate's nickname will be after it has been imported into the **libreswan** certificate database. The nickname is taken directly from the certificate's subject's Common Name (CN):

```
# openssl x509 \
-in /path/to/client-certificate -subject -noout | \
sed -n 's/.*CN=\.*/\1/p'
```

2. Use **openssl** to combine the client certificate, CA certificate, and private key files into a **PKCS#12** file, which is a common file format for multiple certificates and keys:

```
# openssl pkcs12 -export \
-in /path/to/client-certificate \
-inkey /path/to/private-key \
-certfile /path/to/certificate-authority \
-passout pass: \
-out certs.p12
```

3. Import the **PKCS#12** file into the **libreswan** certificate database. The **-w** option is left empty because no password is assigned to the **PKCS#12** file, as it is only temporary.

```
# ipsec initnss
# pk12util -i certs.p12 -d sql:/etc/ipsec.d -w ""
# rm certs.p12
```

35.2.3. Step 3: libreswan IPsec Policy

Now that the necessary certificates are imported into the **libreswan** certificate database, create a policy that uses them to secure communication between hosts in your cluster.

If you are using **libreswan** 3.19 or later, then [opportunistic group configuration](#) is recommended. Otherwise, explicit connections are required.

35.2.3.1. Opportunistic Group Configuration

The following configuration creates two **libreswan** connections. The first encrypts traffic using the OpenShift Container Platform certificates, while the second creates exceptions to the encryption for cluster-external traffic.

1. Place the following into the **/etc/ipsec.d/openshift-cluster.conf** file:

```
conn private
left=%defaultroute
leftid=%fromcert
# our certificate
leftcert="NSS Certificate DB:<cert_nickname>"
right=%opportunisticgroup
rightid=%fromcert
```

1

```
# their certificate transmitted via IKE
rightca=%same
ikev2=insist
authby=rsasig
failureshunt=drop
negotiationshunt=hold
auto=ondemand

conn clear
left=%defaultroute
right=%group
authby=never
type=passthrough
auto=route
priority=100
```

1. Replace `<cert_nickname>` with the certificate nickname from step one.

2. Tell **libreswan** which IP subnets and hosts to apply each policy using policy files in `/etc/ipsec.d/policies/`, where each configured connection has a corresponding policy file. So, in the example above, the two connections, **private** and **clear**, each have a file in `/etc/ipsec.d/policies/`. `/etc/ipsec.d/policies/private` should contain the IP subnet of your cluster, which your hosts receive IP addresses from. By default, this causes all communication between hosts in the cluster subnet to be encrypted if the remote host's client certificate authenticates against the local host's Certificate Authority certificate. If the remote host's certificate does not authenticate, all traffic between the two hosts will be blocked.

For example, if all hosts are configured to use addresses in the **172.16.0.0/16** address space, your **private** policy file would contain **172.16.0.0/16**. Any number of additional subnets to encrypt may be added to this file, which results in all traffic to those subnets using IPsec as well.

3. Unencrypt the communication between all hosts and the subnet gateway to ensure that traffic can enter and exit the cluster. Add the gateway to the `/etc/ipsec.d/policies/clear` file:

```
172.16.0.1/32
```

Additional hosts and subnets may be added to this file, which will result in all traffic to these hosts and subnets being unencrypted.

35.2.3.2. Explicit Connection Configuration

In this configuration, each IPSec node configuration must explicitly list the configuration of every other node in the cluster. Using a configuration management tool such as Ansible to generate this file on each host is recommended.

1. This configuration also requires the full certificate subject of each node to be placed into the configuration for every other node. To read this subject from the node's certificate, use **openssl**:

```
# openssl x509 \
-in /path/to/client-certificate -text | \
grep "Subject:" | \
sed 's/[[:blank:]]*Subject: //'
```


- Place the following lines into the `/etc/ipsec.d/openshift-cluster.conf` file on each node for every other node in the cluster:

```
conn <other_node_hostname>
    left=<this_node_ip> ❶
    leftid="CN=<this_node_cert_nickname>" ❷
    lefttrsasigkey=%cert
    leftcert=<this_node_cert_nickname> ❸
    right=<other_node_ip> ❹
    rightid="<other_node_cert_full_subject>" ❺
    righttrsasigkey=%cert
    auto=start
    keyingtries=%forever
```

- ❶ Replace `<this_node_ip>` with the cluster IP address of this node.
- ❷ ❸ Replace `<this_node_cert_nickname>` with the node certificate nickname from step one.
- ❹ Replace `<other_node_ip>` with the cluster IP address of the other node.
- ❺ Replace `<other_node_cert_full_subject>` with the other node's certificate subject from just above. For example: `"O=system:nodes,CN=openshift-node-45.example.com"`.

- Place the following in the `/etc/ipsec.d/openshift-cluster.secrets` file on each node:

```
: RSA "<this_node_cert_nickname>" ❶
```

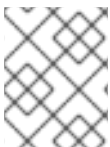
- ❶ Replace `<this_node_cert_nickname>` with the node certificate nickname from step one.

35.3. IPSEC FIREWALL CONFIGURATION

All nodes within the cluster need to allow IPsec related network traffic. This includes IP protocol numbers 50 and 51 as well as UDP port 500.

For example, if the cluster nodes communicate over interface `eth0`:

```
-A OS_FIREWALL_ALLOW -i eth0 -p 50 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p 51 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p udp --dport 500 -j ACCEPT
```



NOTE

IPsec also uses UDP port 4500 for NAT traversal, though this should not apply to normal cluster deployments.

35.4. STARTING AND ENABLING IPSEC

- Start the `ipsec` service to load the new configuration and policies, and begin encrypting:

```
# systemctl start ipsec
```

2. Enable the **ipsec** service to start on boot:

```
# systemctl enable ipsec
```

35.5. OPTIMIZING IPSEC

See the [Scaling and Performance Guide](#) for performance suggestions when encrypting with IPsec.

35.6. TROUBLESHOOTING

When authentication cannot be completed between two hosts, you will not be able to ping between them, because all IP traffic will be rejected. If the **clear** policy is not configured correctly, you will also not be able to SSH to the host from another host in the cluster.

You can use the **ipsec status** command to check that the **clear** and **private** policies have been loaded.

CHAPTER 36. BUILDING DEPENDENCY TREES

36.1. OVERVIEW

OpenShift Container Platform uses [image change triggers](#) in a **BuildConfig** to detect when an [image stream tag](#) has been updated. You can use the **oc adm build-chain** command to build a dependency tree that identifies which [images](#) would be affected by updating an image in a specified [image stream](#).

The **build-chain** tool can determine which [builds](#) to trigger; it analyzes the output of those builds to determine if they will in turn update another [image stream tag](#). If they do, the tool continues to follow the dependency tree. Lastly, it outputs a graph specifying the image stream tags that would be impacted by an update to the top-level tag. The default output syntax for this tool is set to a human-readable format; the DOT format is also supported.

36.2. USAGE

The following table describes common **build-chain** usage and general syntax:

Table 36.1. Common build-chain Operations

Description	Syntax
Build the dependency tree for the latest tag in <image-stream> .	<pre>\$ oc adm build-chain <image-stream></pre>
Build the dependency tree for the v2 tag in DOT format, and visualize it using the DOT utility.	<pre>\$ oc adm build-chain <image-stream>:v2 \ -o dot \ dot -T svg -o deps.svg</pre>
Build the dependency tree across all projects for the specified image stream tag found the test project.	<pre>\$ oc adm build-chain <image-stream>:v1 \ -n test --all</pre>



NOTE

You may need to install the **graphviz** package to use the **dot** command.

CHAPTER 37. REPLACING A FAILED ETCD MEMBER

If some etcd members fail, but you still have a quorum of etcd members, you can use the remaining etcd members and the data that they contain to add more etcd members without etcd or cluster downtime.

37.1. REMOVING A FAILED ETCD NODE

Before you add a new etcd node, remove the failed one.

Procedure

1. From an active etcd host, remove the failed etcd node:

```
# etcdctl -C https://<surviving host IP>:2379 \
--ca-file=/etc/etcd/ca.crt \
--cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key cluster-health

# etcdctl -C https://<surviving host IP>:2379 \
--ca-file=/etc/etcd/ca.crt \
--cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key member remove <failed member
identifier>
```

2. Stop the etcd service on the failed etcd member:

```
# systemctl stop etcd
```

37.2. ADDING AN ETCD MEMBER

You can add an etcd host either by using an Ansible playbook or by manual steps.

37.2.1. Adding a new etcd host using Ansible

Procedure

1. In the Ansible inventory file, create a new group named **[new_etcd]** and add the new host. Then, add the **new_etcd** group as a child of the **[OSEv3]** group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd 1

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
```

```
[new_etcd] 2
etcd0.example.com 3
```

1 2 3 Add these lines.

2. From the host that installed OpenShift Container Platform and hosts the Ansible inventory file, run the etcd **scaleup** playbook:

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-etcd/scaleup.yml
```

3. After the playbook runs, modify the inventory file to reflect the current status by moving the new etcd host from the **[new_etcd]** group to the **[etcd]** group:

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. If you use Flannel, modify the **flanneld** service configuration on every OpenShift Container Platform host, located at **/etc/sysconfig/flanneld**, to include the new etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379,https://etcd0.example.com:2379
```

5. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

37.2.2. Manually adding a new etcd host

Procedure

Modify the current etcd cluster

To create the etcd certificates, run the **openss1** command, replacing the values with those from your environment.

1. Create some environment variables:

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"
```

```
export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}"
export PREFIX="/etc/etcd/generated_certs/etcd-$CN/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```



NOTE

The custom **openssl** extensions used as **etcd_v3_ca_*** include the **\$SAN** environment variable as **subjectAltName**. See **/etc/etcd/ca/openssl.cnf** for more information.

2. Create the directory to store the configuration and certificates:

```
# mkdir -p ${PREFIX}
```

3. Create the server certificate request and sign it: (**server.csr** and **server.crt**)

```
# openssl req -new -config ${OPENSSLCFG} \
  -keyout ${PREFIX}server.key \
  -out ${PREFIX}server.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}server.crt \
  -in ${PREFIX}server.csr \
  -extensions etcd_v3_ca_server -batch
```

4. Create the peer certificate request and sign it: (**peer.csr** and **peer.crt**)

```
# openssl req -new -config ${OPENSSLCFG} \
  -keyout ${PREFIX}peer.key \
  -out ${PREFIX}peer.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. Copy the current etcd configuration and **ca.crt** files from the current node as examples to modify later:

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. While still on the surviving etcd host, add the new host to the cluster. To add additional etcd members to the cluster, you must first adjust the default **localhost** peer in the **peerURLs** value for the first member:

- a. Get the member ID for the first member using the **member list** command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
/172.18.0.75:2379" \ ❶
member list
```

- ❶ Ensure that you specify the URLs of only active etcd members in the **--peers** parameter value.

- b. Obtain the IP address where etcd listens for cluster peers:

```
$ ss -ltn | grep 2380
```

- c. Update the value of **peerURLs** using the **etcdctl member update** command by passing the member ID and IP address obtained from the previous steps:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://
/172.18.0.75:2379" \
member update 511b7fb6cc0001 https://172.18.1.18:2380
```

- d. Re-run the **member list** command and ensure the peer URLs no longer include **localhost**.

7. Add the new host to the etcd cluster. Note that the new host is not yet configured, so the status stays as **unstarted** until the you configure the new host.



WARNING

You must add each member and bring it online one at a time. When you add each additional member to the cluster, you must adjust the **peerURLs** list for the current peers. The **peerURLs** list grows by one for each member added. The **etcdctl member add** command outputs the values that you must set in the **etcd.conf** file as you add each member, as described in the following instructions.

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 ❶
```

```
Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster
```

```
ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https://<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https://<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

- 1 In this line, **10.3.9.222** is a label for the etcd member. You can specify the host name, IP address, or a simple name.

8. Update the sample **`${PREFIX}/etcd.conf`** file.

- a. Replace the following values with the values generated in the previous step:

- **ETCD_NAME**
- **ETCD_INITIAL_CLUSTER**
- **ETCD_INITIAL_CLUSTER_STATE**

- b. Modify the following variables with the new host IP from the output of the previous step. You can use **`${NEW_ETCD_IP}`** as the value.

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

- c. If you previously used the member system as an etcd node, you must overwrite the current values in the **`/etc/etcd/etcd.conf`** file.
- d. Check the file for syntax errors or missing IP addresses, otherwise the etcd service might fail:

```
# vi ${PREFIX}/etcd.conf
```

9. On the node that hosts the installation files, update the **`[etcd]`** hosts group in the **`/etc/ansible/hosts`** inventory file. Remove the old etcd hosts and add the new ones.

10. Create a **tgz** file that contains the certificates, the sample configuration file, and the **ca** and copy it to the new host:

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

Modify the new etcd host

1. Install **iptables-services** to provide iptables utilities to open the required ports for etcd:

```
# yum install -y iptables-services
```

2. Create the **OS_FIREWALL_ALLOW** firewall rules to allow etcd to communicate:

- Port 2379/tcp for clients
- Port 2380/tcp for peer communication

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2380 -j ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```



NOTE

In this example, a new chain **OS_FIREWALL_ALLOW** is created, which is the standard naming the OpenShift Container Platform installer uses for firewall rules.



WARNING

If the environment is hosted in an IaaS environment, modify the security groups for the instance to allow incoming traffic to those ports as well.

3. Install etcd:

```
# yum install -y etcd
```

Ensure version **etcd-2.3.7-4.el7.x86_64** or greater is installed,

4. Ensure the etcd service is not running:

```
# systemctl disable etcd --now
```

5. Remove any etcd configuration and data:

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. Extract the certificates and configuration files:

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. Modify the file ownership permissions:

```
# chown -R etcd:etcd /etc/etcd/*
# chown -R etcd:etcd /var/lib/etcd/
```

8. Start etcd on the new host:

```
# systemctl enable etcd --now
```

9. Verify that the host is part of the cluster and the current cluster health:

- If you use the v2 etcd api, run the following command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --peers="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

- If you use the v3 etcd api, run the following command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
  --key=/etc/etcd/peer.key \
  --cacert="/etc/etcd/ca.crt" \
  --endpoints="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379,\
https://*etcd0.example.com*:2379"\
  endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed
proposal: took = 1.498829ms
```

Modify each OpenShift Container Platform master

1. Modify the master configuration in the **etcdClientInfo** section of the **/etc/origin/master/master-config.yaml** file on every master. Add the new etcd host to the list of the etcd servers OpenShift Container Platform uses to store the data, and remove any failed etcd hosts:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
```

```
keyFile: master.etcd-client.key
urls:
- https://master-0.example.com:2379
- https://master-1.example.com:2379
- https://master-2.example.com:2379
- https://etcd0.example.com:2379
```

2. Restart the master API service:

- On every master:

```
# systemctl restart atomic-openshift-master-api
```

- Or, on a single master cluster installation:

```
# systemctl restart atomic-openshift-master
```



WARNING

The number of etcd nodes must be odd, so you must add at least two hosts.

3. If you use Flannel, modify the **flanneld** service configuration located at **/etc/sysconfig/flanneld** on every OpenShift Container Platform host to include the new etcd host:

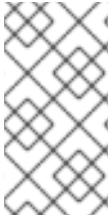
```
FLANNEL_ETCD_ENDPOINTS=https://master-0.example.com:2379,https://master-1.example.com:2379,https://master-2.example.com:2379,https://etcd0.example.com:2379
```

4. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

CHAPTER 38. RESTORING ETCD QUORUM

If you lose etcd quorum, you must back up etcd, take down your etcd cluster, and form a new one. You can use one healthy etcd node to form a new cluster, but you must remove all other healthy nodes.



NOTE

During etcd quorum loss, applications that run on OpenShift Container Platform are unaffected. However, the platform functionality is limited to read-only operations. You cannot take action such as scaling an application up or down, changing deployments, or running or modifying builds.

To confirm the loss of etcd quorum, run the following command and confirm that the cluster is unhealthy:

```
# ETCDCCTL_API=2 etcdctl --cert-file=/etc/origin/master/master.etcd-
client.crt \
    --key-file /etc/origin/master/master.etcd-client.key \
    --ca-file /etc/origin/master/master.etcd-ca.crt \
    --endpoints="https://*master-0.example.com*:2379,\
https://*master-1.example.com*:2379,\
https://*master-2.example.com*:2379"\
cluster-health

member 165201190bf7f217 is unhealthy: got unhealthy result from
https://master-0.example.com:2379
member b50b8a0acab2fa71 is unreachable: [https://master-
1.example.com:2379] are all unreachable
member d40307cbca7bc2df is unreachable: [https://master-
2.example.com:2379] are all unreachable
cluster is unhealthy
```

Note the member IDs and host names of the hosts. You use one of the nodes that can be reached to form a new cluster.

38.1. BACKING UP ETCD

When you back up etcd, you must back up both the etcd configuration files and the etcd data.

38.1.1. Backing up etcd configuration files

The etcd configuration files to be preserved are all stored in the `/etc/etcd` directory of the instances where etcd is running. This includes the etcd configuration file (`/etc/etcd/etcd.conf`) and the required certificates for cluster communication. All those files are generated at installation time by the Ansible installer.

Procedure

For each etcd member of the cluster, back up the etcd configuration.

```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

**NOTE**

The certificates and configuration files on each etcd cluster member are unique.

38.1.2. Backing up etcd data**Prerequisites****NOTE**

The OpenShift Container Platform installer creates aliases to avoid typing all the flags named **etcdctl2** for etcd v2 tasks and **etcdctl3** for etcd v3 tasks.

However, the **etcdctl3** alias does not provide the full endpoint list to the **etcdctl** command, so the **--endpoints** option with all the endpoints must be provided.

Before backing up etcd:

- **etcdctl** binaries should be available or, in containerized installations, the **rhel7/etcd** container should be available
- Ensure connectivity with the etcd cluster (port 2379/tcp)
- Ensure the proper certificates to connect to the etcd cluster

Procedure**NOTE**

While the **etcdctl backup** command is used to perform the backup, etcd v3 has no concept of a *backup*. Instead, you either take a *snapshot* from a live member with the **etcdctl snapshot save** command or copy the **member/snap/db** file from an etcd data directory.

The **etcdctl backup** command rewrites some of the metadata contained in the backup, specifically, the node ID and cluster ID, which means that in the backup, the node loses its former identity. To recreate a cluster from the backup, you create a new, single-node cluster, then add the rest of the nodes to the cluster. The metadata is rewritten to prevent the new node from joining an existing cluster.

Back up the etcd data:

- If you use the v2 API, take the following actions:

- a. Stop all etcd services:

```
# systemctl stop etcd.service
```

- b. Create the etcd data backup and copy the etcd **db** file:

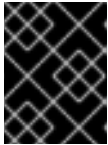
```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl2 backup \
  --data-dir /var/lib/etcd \
```

```
--backup-dir /backup/etcd-$(date +%Y%m%d)
# cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
```

c. Start all etcd services:

```
# systemctl start etcd.service
```

- If you use the v3 API, run the following commands:



IMPORTANT

Because clusters upgraded from previous versions of OpenShift Container Platform might contain v2 data stores, back up both v2 and v3 datastores.

a. Back up etcd v3 data:

```
# systemctl show etcd --property=ActiveState,SubState
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save */backup/etcd-$(date +%Y%m%d)*/db
Snapshot saved at /backup/etcd-<date>/db
```

b. Back up etcd v2 data:

```
# systemctl stop etcd.service
# etcdctl2 backup \
  --data-dir /var/lib/etcd \
  --backup-dir /backup/etcd-$(date +%Y%m%d)
# cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```



NOTE

The **etcdctl snapshot save** command requires the etcd service to be running.

In these commands, a **/backup/etcd-<date>/** directory is created, where **<date>** represents the current date, which must be an external NFS share, S3 bucket, or any external storage location.

In the case of an all-in-one cluster, the etcd data directory is located in the **/var/lib/origin/openshift.local.etcd** directory.

38.2. REMOVING AN ETCD HOST

If an etcd host fails beyond restoration, remove it from the cluster. To recover from an etcd quorum loss, you must also remove all healthy etcd nodes but one from your cluster.

Steps to be performed on all masters hosts

Procedure

1. Remove each other etcd host from the etcd cluster. Run the following command for each etcd node:

```
# etcdctl -C https://<surviving host IP address>:2379 \
--ca-file=/etc/etcd/ca.crt \
--cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key member remove <failed member ID>
```

2. Remove the other etcd hosts from the `/etc/origin/master/master-config.yaml` +master configuration file on every master:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
    - https://master-1.example.com:2379 ❶
    - https://master-2.example.com:2379 ❷
```

❶ ❷ The host to remove.

3. Restart the master API service on every master:

```
# systemctl restart atomic-openshift-master-api
```

Or, if using a single master cluster installation:

```
# systemctl restart atomic-openshift-master
```

Steps to be performed in the current etcd cluster

Procedure

1. Remove the failed host from the cluster:

```
# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
failed to check the health of member 8372784203e11288 on
https://192.168.55.21:2379: Get https://192.168.55.21:2379/health:
dial tcp 192.168.55.21:2379: getsockopt: connection refused
member 8372784203e11288 is unreachable: [https://192.168.55.21:2379]
are all unreachable
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy

# etcdctl2 member remove 8372784203e11288 ❶
Removed member 8372784203e11288 from cluster

# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
```

```
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

1. The **remove** command requires the etcd ID, not the hostname.
2. To ensure the etcd configuration does not use the failed host when the etcd service is restarted, modify the **/etc/etcd/etcd.conf** file on all remaining etcd hosts and remove the failed host in the value for the **ETCD_INITIAL_CLUSTER** variable:

```
# vi /etc/etcd/etcd.conf
```

For example:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380,master-
2.example.com=https://192.168.55.13:2380
```

becomes:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380
```



NOTE

Restarting the etcd services is not required, because the failed host is removed using **etcdctl**.

3. Modify the Ansible inventory file to reflect the current status of the cluster and to avoid issues when re-running a playbook:

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
```

4. If you are using Flannel, modify the **flanneld** service configuration located at **/etc/sysconfig/flanneld** on every host and remove the etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379
```


- Restart the **flannel** service:

```
# systemctl restart flannel.service
```

38.3. CREATING A SINGLE-NODE ETCD CLUSTER

To restore the full functionality of your OpenShift Container Platform instance, make a remaining etcd node a standalone etcd cluster.

Procedure

- On the etcd node that you did not remove from the cluster, stop all etcd services:

```
# systemctl stop etcd.service
```

- Run the etcd service on the host, forcing a new cluster.

These commands create a custom file for the etcd service, which adds the **--force-new-cluster** option to the etcd start command:

```
# mkdir -p /etc/systemd/system/etcd.service.d/
# echo "[Service]" > /etc/systemd/system/etcd.service.d/temp.conf
# echo "ExecStart=" >> /etc/systemd/system/etcd.service.d/temp.conf
# sed -n '/ExecStart/s/"$/ --force-new-cluster"/p' \
    /usr/lib/systemd/system/etcd.service \
    >> /etc/systemd/system/etcd.service.d/temp.conf

# systemctl daemon-reload
# systemctl restart etcd
```

- List the etcd member and confirm that the member list contains only your single etcd host:

```
# etcdctl member list
165201190bf7f217: name=192.168.34.20 peerURLs=http://localhost:2380
clientURLs=https://master-0.example.com:2379 isLeader=true
```

- After restoring the data and creating a new cluster, you must update the **peerURLs** parameter value to use the IP address where etcd listens for peer communication:

```
# etcdctl member update 165201190bf7f217 https://192.168.34.20:2380
```

1

- 1** **165201190bf7f217** is the member ID shown in the output of the previous command, and **https://192.168.34.20:2380** is its IP address.

- To verify, check that the IP is in the member list:

```
$ etcdctl member list
5ee217d17301: name=master-0.example.com
peerURLs=https://*192.168.55.8*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

CHAPTER 39. ADDING ETCD NODES AFTER RESTORING

After the first instance is running, you can add multiple etcd servers to your cluster.

Procedure

1. Get the etcd name for the instance in the **ETCD_NAME** variable:

```
# grep ETCD_NAME /etc/etcd/etcd.conf
```

2. Get the IP address where etcd listens for peer communication:

```
# grep ETCD_INITIAL_ADVERTISE_PEER_URLS /etc/etcd/etcd.conf
```

3. If the node was previously part of a etcd cluster, delete the previous etcd data:

```
# rm -Rf /var/lib/etcd/*
```

4. On the etcd host where etcd is properly running, add the new member:

- If you use the v2 etcd api, run this command:

```
$ etcdctl member add <name> <advertise_peer_urls>
```

- If you use the v3 etcd api, run this command:

```
# etcdctl member add *<name>* \
  --peer-urls="*<advertise_peer_urls>*"
```

The command outputs some variables. For example:

```
ETCD_NAME="master2"
ETCD_INITIAL_CLUSTER="master-
0.example.com=https://192.168.55.8:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

5. Add the values from the previous command to the **/etc/etcd/etcd.conf** file of the new host:

```
# vi /etc/etcd/etcd.conf
```

6. Start the etcd service in the node joining the cluster:

```
# systemctl start etcd.service
```

7. Check for error messages:

```
$ journalctl -fu etcd.service
```

8. Repeat the previous steps for every etcd node to be added.
9. Once you add all the nodes, verify the cluster status and cluster health:

- If you use the v2 etcd api, run the following commands:

```
# etcdctl member list
5cd050b4d701: name=master1 peerURLs=https://10.0.0.7:2380
clientURLs=https://10.0.0.7:2379 isLeader=true
d0c57659d8990cbd: name=master2 peerURLs=https://10.0.0.5:2380
clientURLs=https://10.0.0.5:2379 isLeader=false
e4696d637de3eb2d: name=master3 peerURLs=https://10.0.0.6:2380
clientURLs=https://10.0.0.6:2379 isLeader=false
```

```
# etcdctl cluster-health
member 5cd050b4d701 is healthy: got healthy result from
https://10.0.0.7:2379
member d0c57659d8990cbd is healthy: got healthy result from
https://10.0.0.5:2379
member e4696d637de3eb2d is healthy: got healthy result from
https://10.0.0.6:2379
cluster is healthy
```

- If you use the v3 etcd api, run the following commands:

```
# etcdctl endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 1.423459ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.767481ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.599694ms

# etcdctl endpoint status
https://master-0.example.com:2379, 40bef1f6c79b3163, 3.2.5, 28
MB, true, 9, 2878
https://master-1.example.com:2379, 1ea57201a3ff620a, 3.2.5, 28
MB, false, 9, 2878
https://master-2.example.com:2379, 59229711e4bc65c8, 3.2.5, 28
MB, false, 9, 2878
```

CHAPTER 40. TROUBLESHOOTING OPENSIFT SDN

40.1. OVERVIEW

As described in the [SDN documentation](#) there are multiple layers of interfaces that are created to correctly pass the traffic from one container to another. In order to debug connectivity issues, you have to test the different layers of the stack to work out where the problem arises. This guide will help you dig down through the layers to identify the problem and how to fix it.

Part of the problem is that OpenShift Container Platform can be set up many ways, and the networking can be wrong in a few different places. So this document will work through some scenarios that, hopefully, will cover the majority of cases. If your problem is not covered, the tools and concepts that are introduced should help guide debugging efforts.

40.2. NOMENCLATURE

Cluster

The set of machines in the cluster. *i.e.* the Masters and the Nodes.

Master

A controller of the OpenShift Container Platform cluster. Note that the master may not be a node in the cluster, and thus, may not have IP connectivity to the pods.

Node

Host in the cluster running OpenShift Container Platform that can host pods.

Pod

Group of containers running on a node, managed by OpenShift Container Platform.

Service

Abstraction that presents a unified network interface that is backed by one or more pods.

Router

A web proxy that can map various URLs and paths into OpenShift Container Platform services to allow external traffic to travel into the cluster.

Node Address

The IP address of a node. This is assigned and managed by the owner of the network to which the node is attached. Must be reachable from any node in the cluster (master and client).

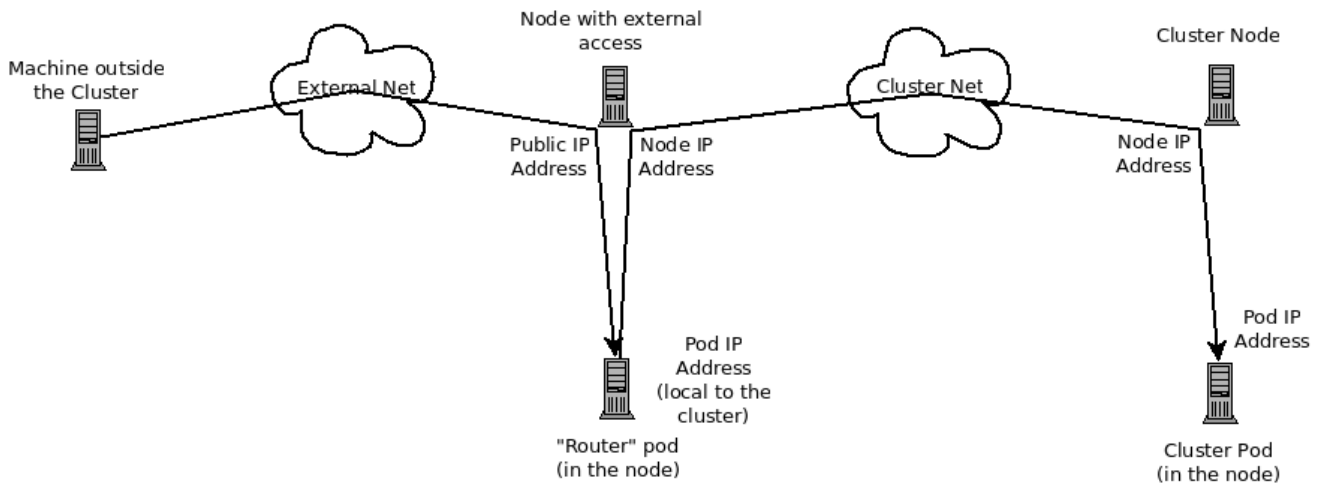
Pod Address

The IP address of a pod. These are assigned and managed by OpenShift Container Platform. By default they are assigned out of the 10.128.0.0/14 network (or, in older versions, 10.1.0.0/16). Only reachable from the client nodes.

Service Address

An IP address that represents the service, and is mapped to a pod address internally. These are assigned and managed by OpenShift Container Platform. By default they are assigned out of the 172.30.0.0/16 network. Only reachable from the client nodes.

The following diagram shows all of the pieces involved with external access.



40.3. DEBUGGING EXTERNAL ACCESS TO AN HTTP SERVICE

If you are on a machine outside the cluster and are trying to access a resource provided by the cluster there needs to be a process running in a pod that listens on a public IP address and "routes" that traffic inside the cluster. The [OpenShift Container Platform router](#) serves that purpose for HTTP, HTTPS (with SNI), WebSockets, or TLS (with SNI).

Assuming you can't access an HTTP service from the outside of the cluster, let's start by reproducing the problem on the command line of the machine where things are failing. Try:

```
curl -kv http://foo.example.com:8000/bar    # But replace the argument
with your URL
```

If that works, are you reproducing the bug from the right place? It is also possible that the service has some pods that work, and some that don't. So jump ahead to the [Section 40.4, "Debugging the Router"](#) section.

If that failed, then let's resolve the DNS name to an IP address (assuming it isn't already one):

```
dig +short foo.example.com                # But replace the hostname
with yours
```

If that doesn't give back an IP address, it's time to troubleshoot DNS, but that's outside the scope of this guide.



IMPORTANT

Make sure that the IP address that you got back is one that you expect to be running the router. If it's not, fix your DNS.

Next, use **ping -c address** and **tracpath address** to check that you can reach the router host. It is possible that they will not respond to ICMP packets, in which case those tests will fail, but the router machine may be reachable. In which case, try using the telnet command to access the port for the router directly:

```
telnet 1.2.3.4 8000
```

You may get:

■

```
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^['.
```

If so, there's something listening on the port on the IP address. That's good. Hit **ctrl-]** then hit the *enter* key and then type **close** to quit telnet. Move on to the [Section 40.4, "Debugging the Router"](#) section to check other things on the router.

Or you could get:

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection refused
```

Which tells us that the router is not listening on that port. See the [Section 40.4, "Debugging the Router"](#) section for more pointers on how to configure the router.

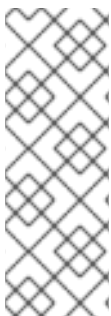
Or if you see:

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection timed out
```

Which tells us that you can't talk to anything on that IP address. Check your routing, firewalls, and that you have a router listening on that IP address. To debug the router, see the [Section 40.4, "Debugging the Router"](#) section. For IP routing and firewall issues, debugging that is beyond the purview of this guide.

40.4. DEBUGGING THE ROUTER

Now that you have an IP address, we need to **ssh** to that machine and check that the router software is running on that machine and configured correctly. So let's **ssh** there and get administrative OpenShift Container Platform credentials.



NOTE

If you have access to administrator credentials but are no longer logged in as the [default system user](#) **system:admin**, you can log back in as this user at any time as long as the credentials are still present in your [CLI configuration file](#). The following command logs in and switches to the **default** project:

```
$ oc login -u system:admin -n default
```

Check that the router is running:

```
# oc get endpoints --namespace=default --selector=router
NAMESPACE   NAME          ENDPOINTS
default     router        10.128.0.4:80
```

If that command fails, then your OpenShift Container Platform configuration is broken. Fixing that is outside the scope of this document.

You should see one or more router endpoints listed, but that won't tell you if they are running on the machine with the given external IP address, since the endpoint IP address will be one of the pod addresses that is internal to the cluster. To get the list of router host IP addresses, run:

```
# oc get pods --all-namespaces --selector=router --template='{{range
.items}}HostIP: {{.status.hostIP}} PodIP: {{.status.podIP}}{{end}}
{{"\n"}}'
HostIP: 192.168.122.202 PodIP: 10.128.0.4
```

You should see the host IP that corresponds to your external address. If you do not, refer to the [router documentation](#) to configure the router pod to run on the right node (by setting the affinity correctly) or update your DNS to match the IP addresses where the routers are running.

At this point in the guide, you should be on a node, running your router pod, but you still cannot get the HTTP request to work. First we need to make sure that the router is mapping the external URL to the correct service, and if that works, we need to dig into that service to make sure that all endpoints are reachable.

Let's list all of the routes that OpenShift Container Platform knows about:

```
# oc get route --all-namespaces
NAME                                HOST/PORT          PATH      SERVICE      LABELS
TLS TERMINATION
route-unsecured    www.example.com    /test     service-name
```

If the host name and path from your URL don't match anything in the list of returned routes, then you need to add a route. See the [router documentation](#).

If your route is present, then you need to debug access to the endpoints. That's the same as if you were debugging problems with a service, so continue on with the next [Section 40.5, "Debugging a Service"](#) section.

40.5. DEBUGGING A SERVICE

If you can't communicate with a service from inside the cluster (either because your services can't communicate directly, or because you are using the router and everything works until you get into the cluster) then you need to work out what endpoints are associated with a service and debug them.

First, let's get the services:

```
# oc get services --all-namespaces
NAMESPACE  NAME                LABELS
SELECTOR   IP(S)              PORT(S)
default    docker-registry    docker-registry=default
docker-registry=default  172.30.243.225    5000/TCP
default    kubernetes         component=apiserver,provider=kubernetes
<none>     172.30.0.1         443/TCP
default    router             router=router
router=router  172.30.213.8      80/TCP
```

You should see your service in the list. If not, then you need to define your [service](#).

The IP addresses listed in the service output are the Kubernetes service IP addresses that Kubernetes will map to one of the pods that backs that service. So you should be able to talk to that IP address. But, unfortunately, even if you can, it doesn't mean all pods are reachable; and if you can't, it doesn't mean all pods aren't reachable. It just tells you the status of the *one* that kubeproxy hooked you up to.

Let's test the service anyway. From one of your nodes:

■

```
curl -kv http://172.30.243.225:5000/bar # Replace the
argument with your service IP address and port
```

Then, let's work out what pods are backing our service (replace **docker-registry** with the name of the broken service):

```
# oc get endpoints --selector=docker-registry
NAME                ENDPOINTS
docker-registry     10.128.2.2:5000
```

From this, we can see that there's only one endpoint. So, if your service test succeeded, and the router test succeeded, then something really odd is going on. But if there's more than one endpoint, or the service test failed, try the following *for each* endpoint. Once you identify what endpoints aren't working, then proceed to the next section.

First, test each endpoint (change the URL to have the right endpoint IP, port, and path):

```
curl -kv http://10.128.2.2:5000/bar
```

If that works, great, try the next one. If it failed, make a note of it and we'll work out why, in the next section.

If all of them failed, then it is possible that the local node is not working, jump to the [Section 40.7, "Debugging Local Networking"](#) section.

If all of them worked, then jump to the [Section 40.11, "Debugging Kubernetes"](#) section to work out why the service IP address isn't working.

40.6. DEBUGGING NODE TO NODE NETWORKING

Using our list of non-working endpoints, we need to test connectivity to the node.

1. Make sure that all nodes have the expected IP addresses:

```
# oc get hostsubnet
NAME                HOST                HOST IP
SUBNET
rh71-os1.example.com rh71-os1.example.com 192.168.122.46
10.1.1.0/24
rh71-os2.example.com rh71-os2.example.com 192.168.122.18
10.1.2.0/24
rh71-os3.example.com rh71-os3.example.com 192.168.122.202
10.1.0.0/24
```

If you are using DHCP they could have changed. Ensure the host names, IP addresses, and subnets match what you expect. If any node details have changed, use **oc edit hostsubnet** to correct the entries.

2. After ensuring the node addresses and host names are correct, list the endpoint IPs and node IPs:

```
# oc get pods --selector=docker-registry \
--template='{range .items}HostIP: {{.status.hostIP}} PodIP:
{{.status.podIP}}{{end}}{"\n"}}'
```


HostIP: 192.168.122.202 PodIP: 10.128.0.4

- Find the endpoint IP address you made note of before and look for it in the **PodIP** entry, and find the corresponding **HostIP** address. Then test connectivity at the node host level using the address from **HostIP**:

- **ping -c 3 <IP_address>**: No response could mean that an intermediate router is eating the ICMP traffic.
- **tracpath <IP_address>**: Shows the IP route taken to the target, if ICMP packets are returned by all hops.
If both **tracpath** and **ping** fail, then look for connectivity issues with your local or virtual network.

- For local networking, check the following:

- Check the route the packet takes out of the box to the target address:

```
# ip route get 192.168.122.202
192.168.122.202 dev ens3 src 192.168.122.46
cache
```

In the above example, it will go out the interface named **ens3** with the source address of **192.168.122.46** and go directly to the target. If that is what you expected, use **ip a show dev ens3** to get the interface details and make sure that is the expected interface.

An alternate result may be the following:

```
# ip route get 192.168.122.202
1.2.3.4 via 192.168.122.1 dev ens3 src 192.168.122.46
```

It will pass through the **via** IP value to route appropriately. Ensure that the traffic is routing correctly. Debugging route traffic is beyond the scope of this guide.

Other debugging options for node to node networking can be solved with the following:

- Do you have ethernet link on both ends? Look for **Link detected: yes** in the output from **ethtool <network_interface>**.
- Are your duplex settings, and ethernet speeds right on both ends? Look through the rest of the **ethtool <network_interface>** information.
- Are the cables plugged in correctly? To the correct ports?
- Are the switches configured correctly?

Once you have ascertained that the node to node connectivity is fine, we need to look at the SDN configuration on both ends.

40.7. DEBUGGING LOCAL NETWORKING

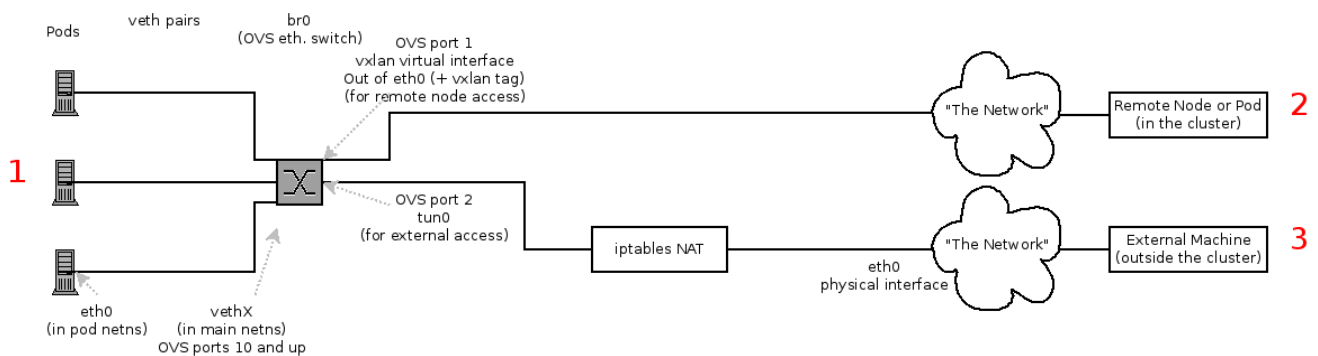
At this point we should have a list of one or more endpoints that you can't communicate with, but that have node to node connectivity. For each one, we need to work out what is wrong, but first you need to understand how the SDN sets up the networking on a node for the different pods.

40.7.1. The Interfaces on a Node

These are the interfaces that the OpenShift SDN creates:

- **br0**: The OVS bridge device that containers will be attached to. OpenShift SDN also configures a set of non-subnet-specific flow rules on this bridge.
- **tun0**: An OVS internal port (port 2 on **br0**). This gets assigned the cluster subnet gateway address, and is used for external network access. OpenShift SDN configures **netfilter** and routing rules to enable access from the cluster subnet to the external network via NAT.
- **vxlan_sys_4789**: The OVS VXLAN device (port 1 on **br0**), which provides access to containers on remote nodes. Referred to as **vxlan0** in the OVS rules.
- **vethX** (in the main netns): A Linux virtual ethernet peer of **eth0** in the Docker netns. It will be attached to the OVS bridge on one of the other ports.

40.7.2. SDN Flows Inside a Node



Depending on what you are trying to access (or be accessed from) the path will vary. There are four different places the SDN connects (inside a node). They are labeled in red on the diagram above.

- **Pod**: Traffic is going from one pod to another on the same machine (1 to a different 1)
- **Remote Node (or Pod)**: Traffic is going from a local pod to a remote node or pod in the same cluster (1 to 2)
- **External Machine**: Traffic is going from a local pod outside the cluster (1 to 3)

Of course the opposite traffic flows are also possible.

40.7.3. Debugging Steps

40.7.3.1. Is IP Forwarding Enabled?

Check that `sysctl net.ipv4.ip_forward` is set to 1 (and check the host if this is a VM)

40.7.3.2. Are your routes correct?

Check the route tables with `ip route`:

```
# ip route
default via 192.168.122.1 dev ens3
10.128.0.0/14 dev tun0 proto kernel scope link      #
This sends all pod traffic into OVS
10.128.2.0/23 dev tun0 proto kernel scope link src 10.128.2.1 #
This is traffic going to local pods, overriding the above
169.254.0.0/16 dev ens3 scope link metric 1002      #
This is for Zeroconf (may not be present)
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.42.1 #
Docker's private IPs... used only by things directly configured by docker;
not OpenShift
192.168.122.0/24 dev ens3 proto kernel scope link src 192.168.122.46 #
The physical interface on the local subnet
```

You should see the 10.128.x.x lines (assuming you have your pod network set to the default range in your configuration). If you do not, check the OpenShift Container Platform logs (see the [Section 40.10](#), “[Reading the Logs](#)” section)

40.7.4. Is the Open vSwitch configured correctly?

Check the Open vSwitch bridges on both sides:

```
# ovs-vsctl list-br
br0
```

This should be **br0**.

You can list all of the ports that ovs knows about:

```
# ovs-ofctl -O OpenFlow13 dump-ports-desc br0
OFPST_PORT_DESC reply (OF1.3) (xid=0x2):
  1(vxlan0): addr:9e:f1:7d:4d:19:4f
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
  2(tun0): addr:6a:ef:90:24:a3:11
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
  8(vethe19c6ea): addr:1e:79:f3:a0:e8:8c
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
  LOCAL(br0): addr:0a:7f:b4:33:c2:43
    config:      PORT_DOWN
    state:       LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
```

In particular, the **vethX** devices for all of the active pods should be listed as ports.

Next, list the flows that are configured on that bridge:

```
# ovs-ofctl -O OpenFlow13 dump-flows br0
```

The results will vary slightly depending on whether you are using the **ovs-subnet** or **ovs-multitenant** plug-in, but there are certain general things you can look for:

1. Every remote node should have a flow matching **tun_src=<node_IP_address>** (for incoming VXLAN traffic from that node) and another flow including the action **set_field: <node_IP_address>->tun_dst** (for outgoing VXLAN traffic to that node).
2. Every local pod should have flows matching **arp_spa=<pod_IP_address>** and **arp_tpa=<pod_IP_address>** (for incoming and outgoing ARP traffic for that pod), and flows matching **nw_src=<pod_IP_address>** and **nw_dst=<pod_IP_address>** (for incoming and outgoing IP traffic for that pod).

If there are flows missing, look in the [Section 40.10, “Reading the Logs”](#) section.

40.7.4.1. Is the iptables configuration correct?

Check the output from **iptables-save** to make sure you are not filtering traffic. However, OpenShift Container Platform sets up iptables rules during normal operation, so do not be surprised to see entries there.

40.7.4.2. Is your external network correct?

Check external firewalls, if any, allow traffic to the target address (this is site-dependent, and beyond the purview of this guide).

40.8. DEBUGGING VIRTUAL NETWORKING

40.8.1. Builds on a Virtual Network are Failing

If you are installing OpenShift Container Platform using a virtual network (for example, OpenStack), and a build is failing, the maximum transmission unit (MTU) of the target node host might not be compatible with the MTU of the primary network interface (for example, **eth0**).

For a build to complete successfully, the MTU of an SDN must be less than the eth0 network MTU in order to pass data to between node hosts.

1. Check the MTU of your network by running the **ip addr** command:

```
# ip addr
---
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether fa:16:3e:56:4c:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.0/24 brd 172.16.0.0 scope global dynamic eth0
        valid_lft 168sec preferred_lft 168sec
    inet6 fe80::f816:3eff:fe56:4c11/64 scope link
        valid_lft forever preferred_lft forever
---
```

The MTU of the above network is 1500.

2. The MTU in your node configuration must be lower than the network value. Check the **mtu** in the node configuration of the targeted node host:

■

```
# cat /etc/origin/node/node-config.yaml
...
networkConfig:
  mtu: 1450
  networkPluginName: company/openshift-ovs-subnet
...
```

In the above node configuration file, the **mtu** value is lower than the network MTU, so no configuration is needed. If the **mtu** value was higher, edit the file and lower the value to at least 50 units fewer than the MTU of the primary network interface, then restart the node service. This would allow larger packets of data to pass between nodes.

40.9. DEBUGGING POD EGRESS

If you are trying to access an external service from a pod, e.g.:

```
curl -kv github.com
```

Make sure that the DNS is resolving correctly:

```
dig +search +noall +answer github.com
```

That should return the IP address for the github server, but check that you got back the correct address. If you get back no address, or the address of one of your machines, then you may be matching the wildcard entry in your local DNS server.

To fix that, you either need to make sure that DNS server that has the wildcard entry is not listed as a **nameserver** in your **/etc/resolv.conf** or you need to make sure that the wildcard domain is not listed in the **search** list.

If the correct IP address was returned, then try the debugging advice listed above in [Section 40.7, “Debugging Local Networking”](#). Your traffic should leave the Open vSwitch on port 2 to pass through the **iptables** rules, then out the route table normally.

40.10. READING THE LOGS

Run: **journalctl -u atomic-openshift-node.service --boot | less**

Look for the **Output of setup script:** line. Everything starting with '+' below that are the script steps. Look through that for obvious errors.

Following the script you should see lines with **Output of adding table=0**. Those are the OVS rules, and there should be no errors.

40.11. DEBUGGING KUBERNETES

Check **iptables -t nat -L** to make sure that the service is being NAT'd to the right port on the local machine for the **kubeproxy**.

**WARNING**

This is all changing soon... Kubeproxy is being eliminated and replaced with an **iptables**-only solution.

40.12. FINDING NETWORK ISSUES USING THE DIAGNOSTICS TOOL

As a cluster administrator, run the diagnostics tool to diagnose common network issues:

```
# oc adm diagnostics NetworkCheck
```

The diagnostics tool runs a series of checks for error conditions for the specified component. See the [Diagnostics Tool section](#) for more information.

**NOTE**

Currently, the diagnostics tool cannot diagnose IP failover issues. As a workaround, you can run the script at <https://raw.githubusercontent.com/openshift/openshift-sdn/master/hack/ipf-debug.sh> on the master (or from another machine with access to the master) to generate useful debugging information. However, this script is unsupported.

By default, **oc adm diagnostics NetworkCheck** logs errors into */tmp/openshift/*. This can be configured with the **--network-logdir** option:

```
# oc adm diagnostics NetworkCheck --network-logdir=<path/to/directory>
```

40.13. MISCELLANEOUS NOTES

40.13.1. Other clarifications on ingress

- Kube - declare a service as NodePort and it will claim that port on all machines in the cluster (on what interface?) and then route into kube-proxy and then to a backing pod. See <https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport> (some node must be accessible from outside)
- Kube - declare as a LoadBalancer and something *you* have to write does the rest
- OS/AE - Both use the router

40.13.2. TLS Handshake Timeout

When a pod fails to deploy, check its docker log for a TLS handshake timeout:

```
$ docker log <container_id>
...
[...] couldn't get deployment [...] TLS handshake timeout
...
```

This condition, and generally, errors in establishing a secure connection, may be caused by a large difference in the MTU values between tun0 and the primary interface (e.g., eth0), such as when tun0 MTU is 1500 and eth0 MTU is 9000 (jumbo frames).

40.13.3. Other debugging notes

- Peer interfaces (of a Linux virtual ethernet pair) can be determined with **ethtool -S *ifname***
- Driver type: **ethtool -i *ifname***

CHAPTER 41. DIAGNOSTICS TOOL

41.1. OVERVIEW

The **oc adm diagnostics** command runs a series of checks for error conditions in the host or cluster. Specifically, it:

- Verifies that the default registry and router are running and correctly configured.
- Checks **ClusterRoleBindings** and **ClusterRoles** for consistency with base policy.
- Checks that all of the client configuration contexts are valid and can be connected to.
- Checks that SkyDNS is working properly and the pods have SDN connectivity.
- Validates master and node configuration on the host.
- Checks that nodes are running and available.
- Analyzes host logs for known errors.
- Checks that systemd units are configured as expected for the host.

41.2. USING THE DIAGNOSTICS TOOL

You can deploy OpenShift Container Platform in several ways. These include:

- Built from source
- Included within a VM image
- As a container image
- Using enterprise RPMs

Each method is suited for a different configuration and environment. To minimize environment assumptions, the diagnostics tool is included with the **openshift** binary to provide diagnostics within an OpenShift Container Platform server or client.

To use the diagnostics tool, preferably on a master host and as cluster administrator, run:

```
# oc adm diagnostics
```

This runs all available diagnostics and skips any that do not apply to the environment.

You can run a specific diagnostics by name or run specific diagnostics by name as you work to address issues. For example:

```
$ oc adm diagnostics
```

The options for the diagnostics tool require working configuration files. For example, the **NodeConfigCheck** does not run unless a node configuration is available.

The diagnostics tool uses the standard configuration file locations by default:

- Client:
 - As indicated by the **\$KUBECONFIG** environment variable
 - *~/.kube/config file*
- Master:
 - */etc/origin/master/master-config.yaml*
- Node:
 - */etc/origin/node/node-config.yaml*

You can specify non-standard locations with the **--config**, **--master-config**, and **--node-config** options. If a configuration file is not specified, related diagnostics are skipped.

Available diagnostics include:

Diagnostic Name	Purpose
AggregatedLogging	Check the aggregated logging integration for proper configuration and operation.
AnalyzeLogs	Check systemd service logs for problems. Does not require a configuration file to check against.
ClusterRegistry	Check that the cluster has a working Docker registry for builds and image streams.
ClusterRoleBindings	Check that the default cluster role bindings are present and contain the expected subjects according to base policy.
ClusterRoles	Check that cluster roles are present and contain the expected permissions according to base policy.
ClusterRouter	Check for a working default router in the cluster.
ConfigContexts	Check that each context in the client configuration is complete and has connectivity to its API server.
DiagnosticPod	Creates a pod that runs diagnostics from an application standpoint, which checks that DNS within the pod is working as expected and the credentials for the default service account authenticate correctly to the master API.
EtcdWriteVolume	Check the volume of writes against etcd for a time period and classify them by operation and key. This diagnostic only runs if specifically requested, because it does not run as quickly as other diagnostics and can increase load on etcd.

Diagnostic Name	Purpose
MasterConfigCheck	Check this host's master configuration file for problems.
MasterNode	Check that the master running on this host is also running a node to verify that it is a member of the cluster SDN.
MetricsApiProxy	Check that the integrated Heapster metrics can be reached via the cluster API proxy.
NetworkCheck	<p>Create diagnostic pods on multiple nodes to diagnose common network issues from an application standpoint. For example, this checks that pods can connect to services, other pods, and the external network.</p> <p>If there are any errors, this diagnostic stores results and retrieved files in a local directory (<i>/tmp/openshift/</i>, by default) for further analysis. The directory can be specified with the -network-logdir flag.</p>
NodeConfigCheck	Checks this host's node configuration file for problems.
NodeDefinitions	Check that the nodes defined in the master API are ready and can schedule pods.
RouteCertificateValidation	Check all route certificates for those that might be rejected by extended validation.
ServiceExternalIPs	Check for existing services that specify external IPs, which are disallowed according to master configuration.
UnitStatus	Check systemd status for units on this host related to OpenShift Container Platform. Does not require a configuration file to check against.

41.3. RUNNING DIAGNOSTICS IN A SERVER ENVIRONMENT

An Ansible-deployed cluster provides additional diagnostic benefits for nodes within an OpenShift Container Platform cluster. These include:

- Master and node configuration is based on a configuration file in a standard location.
- Systemd units are configured to manage the server(s).
- Both master and node configuration files are in standard locations.
- Systemd units are created and configured for managing the nodes in a cluster.
- All components log to journald.

Keeping to the default location of the configuration files placed by an Ansible-deployed cluster ensures that running **oc adm diagnostics** works without any flags. If you are not using the default location for the configuration files, you must use the **--master-config** and **--node-config** options:

```
# oc adm diagnostics --master-config=<file_path> --node-config=<file_path>
```

Systemd units and logs entries in journald are necessary for the current log diagnostic logic. For other deployment types, logs can be stored in single files, stored in files that combine node and master logs, or printed to stdout. If log entries do not use journald, the log diagnostics cannot work and do not run.

41.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT

You can run the diagnostics tool as an ordinary user or a **cluster-admin**, and it runs using the level of permissions granted to the account from which you run it.

A client with ordinary access can diagnose its connection to the master and run a diagnostic pod. If multiple users or masters are configured, connections are tested for all, but the diagnostic pod only runs against the current user, server, or project.

A client with **cluster-admin** access can diagnose the status of infrastructure such as nodes, registry, and router. In each case, running **oc adm diagnostics** searches for the standard client configuration file in its standard location and uses it if available.

41.5. ANSIBLE-BASED HEALTH CHECKS

Additional diagnostic health checks are available through the [Ansible-based tooling](#) used to install and manage OpenShift Container Platform clusters. They can report common deployment problems for the current OpenShift Container Platform installation.

These checks can be run either using the **ansible-playbook** command (the same method used during [Advanced Installation](#)) or as a [containerized version](#) of **openshift-ansible**. For the **ansible-playbook** method, the checks are provided by the **atomic-openshift-utils** RPM package. For the containerized method, the **openshift3/ose-ansible** container image is distributed via the [Red Hat Container Registry](#). Example usage for each method are provided in subsequent sections.

The following health checks are a set of diagnostic tasks that are meant to be run against the Ansible inventory file for a deployed OpenShift Container Platform cluster using the provided **health.yml** playbook.

**WARNING**

Due to potential changes the health check playbooks can make to the environment, you must run the playbooks against only Ansible-deployed clusters and using the same inventory file used for deployment. The changes consist of installing dependencies so that the checks can gather the required information. In some circumstances, additional system components, such as **docker** or networking configurations, can change if their current state differs from the configuration in the inventory file. You should run these health checks only if you do not expect the inventory file to make any changes to the existing cluster configuration.

Table 41.1. Diagnostic Health Checks

Check Name	Purpose
etcd_imagedata_size	<p>This check measures the total size of OpenShift Container Platform image data in an etcd cluster. The check fails if the calculated size exceeds a user-defined limit. If no limit is specified, this check fails if the size of image data amounts to 50% or more of the currently used space in the etcd cluster.</p> <p>A failure from this check indicates that a significant amount of space in etcd is being taken up by OpenShift Container Platform image data, which can eventually result in the etcd cluster crashing.</p> <p>A user-defined limit may be set by passing the etcd_max_image_data_size_bytes variable. For example, setting etcd_max_image_data_size_bytes=40000000000 causes the check to fail if the total size of image data stored in etcd exceeds 40 GB.</p>
etcd_traffic	<p>This check detects higher-than-normal traffic on an etcd host. It fails if a journalctl log entry with an etcd sync duration warning is found.</p> <p>For further information on improving etcd performance, see Recommended Practices for OpenShift Container Platform etcd Hosts and the Red Hat Knowledgebase.</p>

Check Name	Purpose
etcd_volume	<p>This check ensures that the volume usage for an etcd cluster is below a maximum user-specified threshold. If no maximum threshold value is specified, it is defaulted to 90% of the total volume size.</p> <p>A user-defined limit may be set by passing the etcd_device_usage_threshold_percent variable.</p>
docker_storage	<p>Only runs on hosts that depend on the docker daemon (nodes and containerized installations). Checks that docker's total usage does not exceed a user-defined limit. If no user-defined limit is set, docker's maximum usage threshold defaults to 90% of the total size available.</p> <p>You can set the threshold limit for total percent usage with a variable in the inventory file, for example max_thinpool_data_usage_percent=90.</p> <p>This also checks that docker's storage is using a supported configuration.</p>
curator, elasticsearch, fluentd, kibana	<p>This set of checks verifies that Curator, Kibana, Elasticsearch, and Fluentd pods have been deployed and are in a running state, and that a connection can be established between the control host and the exposed Kibana URL. These checks run only if the openshift_logging_install_logging inventory variable is set to true to ensure that they are executed in a deployment where cluster logging is enabled.</p>
logging_index_time	<p>This check detects higher than normal time delays between log creation and log aggregation by Elasticsearch in a logging stack deployment. It fails if a new log entry cannot be queried through Elasticsearch within a timeout (by default, 30 seconds). The check only runs if logging is enabled.</p> <p>A user-defined timeout may be set by passing the openshift_check_logging_index_timeout_seconds variable. For example, setting openshift_check_logging_index_timeout_seconds=45 causes the check to fail if a newly-created log entry is not able to be queried via Elasticsearch after 45 seconds.</p>

**NOTE**

A similar set of checks meant to run as part of the installation process can be found in [Configuring Cluster Pre-install Checks](#). Another set of checks for checking certificate expiration can be found in [Redeploying Certificates](#).

41.5.1. Running Health Checks via ansible-playbook

To run the **openshift-ansible** health checks using the **ansible-playbook** command, specify your cluster's inventory file and run the **health.yml** playbook:

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
    checks/health.yml
```

To set variables in the command line, include the **-e** flag with any desired variables in **key=value** format. For example:

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-
    checks/health.yml
    -e openshift_check_logging_index_timeout_seconds=45
    -e etcd_max_image_data_size_bytes=40000000000
```

To disable specific checks, include the variable **openshift_disable_check** with a comma-delimited list of check names in your inventory file before running the playbook. For example:

```
openshift_disable_check=etcd_traffic,etcd_volume
```

Alternatively, set any checks to disable as variables with **-e openshift_disable_check=<check1>,<check2>** when running the **ansible-playbook** command.

41.5.2. Running Health Checks via Docker CLI

You can run the **openshift-ansible** playbooks in a Docker container, avoiding the need for installing and configuring Ansible, on any host that can run the **ose-ansible** image via the Docker CLI.

Run the following as a non-root user that has privileges to run containers:

```
# docker run -u `id -u` \ 1
    -v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \ 2
    -v /etc/ansible/hosts:/tmp/inventory:ro \ 3
    -e INVENTORY_FILE=/tmp/inventory \
    -e PLAYBOOK_FILE=playbooks/byo/openshift-checks/health.yml \ 4
    -e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 -e
    etcd_max_image_data_size_bytes=40000000000" \ 5
    openshift3/ose-ansible
```

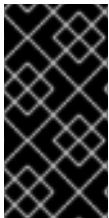
1 These options make the container run with the same UID as the current user, which is required for permissions so that the SSH key can be read inside the container (SSH private keys are expected to be readable only by their owner).

2

Mount SSH keys as a volume under **`/opt/app-root/src/.ssh`** under normal usage when running the container as a non-root user.

- 3 Change **`/etc/ansible/hosts`** to the location of the cluster's inventory file, if different. This file is bind-mounted to **`/tmp/inventory`**, which is used according to the **`INVENTORY_FILE`** environment variable in the container.
- 4 The **`PLAYBOOK_FILE`** environment variable is set to the location of the **`health.yml`** playbook relative to **`/usr/share/ansible/openshift-ansible`** inside the container.
- 5 Set any variables desired for a single run with the **`-e key=value`** format.

In the previous command, the SSH key is mounted with the **`:Z`** option so that the container can read the SSH key from its restricted SELinux context. Adding this option means that your original SSH key file is relabeled similarly to **`system_u:object_r:container_file_t:s0:c113,c247`**. For more details about **`:Z`**, see the **`docker-run(1)`** man page.



IMPORTANT

These volume mount specifications can have unexpected consequences. For example, if you mount, and therefore relabel, the **`$HOME/.ssh`** directory, **`sshd`** becomes unable to access the public keys to allow remote login. To avoid altering the original file labels, mount a copy of the SSH key or directory.

Mounting an entire **`.ssh`** directory can be helpful for:

- Allowing you to use an SSH configuration to match keys with hosts or modify other connection parameters.
- Allowing a user to provide a **`known_hosts`** file and have SSH validate host keys. This is disabled by the default configuration and can be re-enabled with an environment variable by adding **`-e ANSIBLE_HOST_KEY_CHECKING=True`** to the **`docker`** command line.

CHAPTER 42. IDLING APPLICATIONS

42.1. OVERVIEW

As an OpenShift Container Platform administrator, you can idle applications to reduce resource consumption. This is useful when deployed on a public cloud where cost is related to resource consumption.

If any scalable resources are not in use, OpenShift Container Platform discovers, then idles them, by scaling them to 0 replicas. When network traffic is directed to the resources, they are unidled by scaling up the replicas, then operation continues.

Applications are made of services, as well as other scalable resources, such as deployment configurations. The action of idling an application involves idling all associated resources.

42.2. IDLING APPLICATIONS

Idling an application involves finding the scalable resources (deployment configurations, replication controllers, and others) associated with a service. Idling an application finds the service and marks it as idled, scaling down the resources to zero replicas.

You can use the **oc idle** command to [idle a single service](#), or use the **--resource-names-file** option to [idle multiple services](#).

42.2.1. Idling Single Services

Idle a single service with the following command:

```
$ oc idle <service>
```

42.2.2. Idling Multiple Services

Idle multiple services by creating a list of the desired services, then using the **--resource-names-file** option with the **oc idle** command.

This is helpful if an application spans across a set of services within a project, or when idling multiple services in conjunction with a script in order to idle multiple applications in bulk within the same project.

1. Create a text file containing a list of the services, each on their own line.
2. Idle the services using the **--resource-names-file** option:

```
$ oc idle --resource-names-file <filename>
```



NOTE

The idle command is limited to a single project. For idling applications across a cluster, run the idle command for each project individually.

42.3. UNIDLING APPLICATIONS

Application services become active again when they receive network traffic and will be scaled back up their previous state. This includes both traffic to the services and traffic passing through routes.

Applications may be manually unidled by scaling up the resources. For example, to scale up a deploymentconfig, run the command:

```
$ oc scale --replicas=1 dc <deploymentconfig>
```

**NOTE**

Automatic unidling by a router is currently only supported by the default HAProxy router.

CHAPTER 43. ANALYZING CLUSTER CAPACITY

43.1. OVERVIEW

As a cluster administrator, you can use the cluster capacity tool to view the number of pods that can be scheduled to increase the current resources before they become exhausted, and to ensure any future pods can be scheduled. This capacity comes from an individual node host in a cluster, and includes CPU, memory, disk space, and others.

The cluster capacity tool simulates a sequence of scheduling decisions to determine how many instances of an input pod can be scheduled on the cluster before it is exhausted of resources to provide a more accurate estimation.



NOTE

The remaining allocatable capacity is a rough estimation, because it does not count all of the resources being distributed among nodes. It analyzes only the remaining resources and estimates the available capacity that is still consumable in terms of a number of instances of a pod with given requirements that can be scheduled in a cluster.

Also, pods might only have scheduling support on particular sets of nodes based on its selection and affinity criteria. As a result, the estimation of which remaining pods a cluster can schedule can be difficult.

You can run the cluster capacity analysis tool as a stand-alone utility from the command line, or [as a job](#) in a pod inside an OpenShift Container Platform cluster. Running it as job inside of a pod enables you to run it multiple times without intervention.

43.2. RUNNING CLUSTER CAPACITY ANALYSIS ON THE COMMAND LINE

To run the tool on the command line:

```
$ cluster-capacity --kubeconfig <path-to-kubeconfig> \
  --podspec <path-to-pod-spec>
```

The **--kubeconfig** option indicates your Kubernetes configuration file, and the **--podspec** option indicates a sample pod specification file, which the tool uses for estimating resource usage. The **podspec** specifies its resource requirements as **limits** or **requests**. The cluster capacity tool takes the pod's resource requirements into account for its estimation analysis.

An example of the pod specification input is:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
```

```

image: gcr.io/google-samples/gb-frontend:v4
imagePullPolicy: Always
resources:
  limits:
    cpu: 150m
    memory: 100Mi
  requests:
    cpu: 150m
    memory: 100Mi

```

You can also add the **--verbose** option to output a detailed description of how many pods can be scheduled on each node in the cluster:

```

$ cluster-capacity --kubeconfig <path-to-kubeconfig> \
  --podspec <path-to-pod-spec> --verbose

```

The output will look similar to the following:

```

small-pod pod requirements:
- CPU: 150m
- Memory: 100Mi

```

The cluster can schedule 52 instance(s) of the pod small-pod.

```

Termination reason: Unschedulable: No nodes are available that match all
of the
following predicates:: Insufficient cpu (2).

```

```

Pod distribution among nodes:
small-pod
- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)

```

In the above example, the number of estimated pods that can be scheduled onto the cluster is 52.

43.3. RUNNING CLUSTER CAPACITY AS A JOB INSIDE OF A POD

Running the cluster capacity tool as a job inside of a pod has the advantage of being able to be run multiple times without needing user intervention. Running the cluster capacity tool as a job involves using a **ConfigMap**.

1. Create the cluster role:

```

$ cat << EOF | oc create -f -
kind: ClusterRole
apiVersion: v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims",
    "persistentvolumes", "services"]
  verbs: ["get", "watch", "list"]
EOF

```

2. Create the service account:

```
$ oc create sa cluster-capacity-sa
```

3. Add the role to the service account:

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

4. Define and create the pod specification:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

5. The cluster capacity analysis is mounted in a volume using a **ConfigMap** named **cluster-capacity-configmap** to mount input pod spec file **pod.yaml** into a volume **test-volume** at the path **/test-pod**.

If you haven't created a **ConfigMap**, create one before creating the job:

```
$ oc create configmap cluster-capacity-configmap \
  --from-file=pod.yaml=pod.yaml
```

6. Create the job using the below example of a job specification file:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
      containers:
      - name: cluster-capacity
```

```

        image: openshift/origin-cluster-capacity
        imagePullPolicy: "Always"
        volumeMounts:
        - mountPath: /test-pod
          name: test-volume
        env:
        - name: CC_INCLUSTER 1
          value: "true"
        command:
        - "/bin/sh"
        - "-ec"
        - |
          /bin/cluster-capacity --podspec=/test-pod/pod.yaml --
verbose
        restartPolicy: "Never"
        serviceAccountName: cluster-capacity-sa
        volumes:
        - name: test-volume
          configMap:
            name: cluster-capacity-configmap

```

- 1** A required environment variable letting the cluster capacity tool know that it is running inside a cluster as a pod.
 The **pod.yaml** key of the **ConfigMap** is the same as the pod specification file name, though it is not required. By doing this, the input pod spec file can be accessed inside the pod as **/test-pod/pod.yaml**.

7. Run the cluster capacity image as a job in a pod:

```
$ oc create -f cluster-capacity-job.yaml
```

8. Check the job logs to find the number of pods that can be scheduled in the cluster:

```
$ oc logs jobs/cluster-capacity-job
small-pod pod requirements:
- CPU: 150m
- Memory: 100Mi
```

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the following predicates:: Insufficient cpu (2).

Pod distribution among nodes:

```
small-pod
- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)
```

CHAPTER 44. REVISION HISTORY: CLUSTER ADMINISTRATION

44.1. TUES MAR 06 2018

Affected Topic	Description of Change
Managing Security Context Constraints	Added a new Example Security Context Constraints Settings section.

44.2. FRI FEB 23 2018

Affected Topic	Description of Change
Default Scheduling	Reorganized topic and updated the list of predicates and policies in the Scheduler Policy section.
Controlling Pod Placement	Created new topic from the Controlling Pod Placement section of the Default Scheduling topic. No change to the content.
Managing Security Context Constraints	Noted the importance of <code>-z</code> flag usage when granting access to service accounts .
Configuring Service Accounts	Noted the importance of <code>-z</code> flag usage when granting access to service accounts .

44.3. WED FEB 21 2018

Affected Topic	Description of Change
Managing Nodes	Added information about the <code>--dry-run</code> option for <code>oc adm drain</code> .

44.4. FRI FEB 16 2018

Affected Topic	Description of Change
Handling Out of Resource Errors	Adjusted the math in the Example Scenario .
Overcommitting	Updated the Tune Buffer Chunk Limit section with instructions to use file buffering.
Garbage Collection	Added clarifying details about the default behavior of garbage collection.

44.5. TUE FEB 06 2018

Affected Topic	Description of Change
Managing Networking	Changed the Disabling Host Name Collision Prevention For Routes and Ingress Objects section to mention the ability to give users the rights to edits host names on routes and ingress objects.

44.6. FRI DEC 22 2017

Affected Topic	Description of Change
Image Signatures	Added information on configuring automatic image signature import .

44.7. WED NOV 29 2017

OpenShift Container Platform 3.7 Initial Release