



OpenShift Container Platform 3.4

Administrator Solutions

OpenShift Container Platform 3.4 Administrator Solutions Guide

OpenShift Container Platform 3.4 Administrator Solutions

OpenShift Container Platform 3.4 Administrator Solutions Guide

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 Administrator Solutions topics cover the concepts explored in the Cluster Administration reference documentation, but with a focus on presenting the content in a more accessible step-by-step format, with easy to follow examples and sample configurations.

Table of Contents

CHAPTER 1. OVERVIEW	4
CHAPTER 2. MASTER AND NODE CONFIGURATION	5
2.1. OVERVIEW	5
2.2. PREREQUISITES	5
2.3. CONFIGURING MASTERS AND NODES	5
2.3.1. Making Configuration Changes Using Ansible	5
2.3.1.1. Using the htpasswd command	7
2.3.2. Making Manual Configuration Changes	9
2.4. CONFIGURATION OPTIONS	9
2.4.1. Master Configuration File Options	9
2.4.2. Node Configuration File Options	16
CHAPTER 3. USER AND ROLE MANAGEMENT	19
3.1. LIMITING AND MONITORING USERS AND PROJECTS	19
3.1.1. Setting Limits for Users and Projects	19
3.1.1.1. Configuration Options	20
3.1.2. Limiting the Number of Projects a User Can Have	23
3.1.3. Controlling and Monitoring Resource Usage	24
3.2. DETERMINING WHICH ROLES USERS GET BY DEFAULT	24
3.2.1. Leveraging Default Groups	25
3.2.2. Viewing Roles and Users for a Project	26
3.2.3. Viewing Roles and Users for the Cluster	26
3.3. CONTROLLING USER PERMISSIONS WITH ROLES	28
3.3.1. Adding a Role to a User	28
3.3.2. Removing a Role from a User	28
3.3.3. Adding a Cluster Role to a User for All Projects	28
3.3.4. Removing a Cluster Role from a User for All Projects	28
3.3.5. Adding a Role to a Group	29
3.3.6. Removing a Role from a Group	29
3.3.7. Adding a Cluster Role to a Group for All Projects	29
3.3.8. Removing a Cluster Role from a Group for All Projects	29
3.4. SHARING TEMPLATES FOR USE IN PROJECTS ACROSS THE CLUSTER	29
3.5. CREATING A CLUSTER ADMINISTRATOR USER	30
3.5.1. Creating an Administrator Within a Project	30
3.5.2. Creating a Cluster Administrator	30
CHAPTER 4. AUTHENTICATION	31
4.1. OVERVIEW	31
4.2. BASIC AUTHENTICATION (REMOTE)	31
4.2.1. Configuring Authentication on the Master	31
4.2.2. Troubleshooting	32
4.2.3. Verifying Users	33
4.3. REQUEST HEADER AUTHENTICATION	33
4.3.1. Configuring Authentication on the Master	34
4.3.2. Creating Users with Request Header Authentication	36
4.3.3. Verifying Users	36
4.4. KEYSTONE AUTHENTICATION	36
4.4.1. Configuring Authentication on the Master	36
4.4.2. Creating Users with Keystone Authentication	37
4.4.3. Verifying Users	38
4.5. LDAP AUTHENTICATION	38

4.5.1. Configuring Authentication on the Master	39
4.5.2. Creating Users with LDAP Authentication	41
4.5.3. Verifying Users	41
4.6. GITHUB AUTHENTICATION	41
4.6.1. Registering the Application on GitHub	41
4.6.2. Configuring Authentication on the Master	42
4.6.3. Creating Users with GitHub Authentication	43
4.6.4. Verifying Users	43
CHAPTER 5. CERTIFICATE MANAGEMENT	45
5.1. OVERVIEW	45
5.2. CHANGING AN APPLICATION'S SELF-SIGNED CERTIFICATE TO CA-SIGNED CERTIFICATE	45

CHAPTER 1. OVERVIEW

The OpenShift Container Platform Administrator Solutions guide is new as of version 3.3, and the topics cover the most common tasks faced by OpenShift Container Platform administrators, with a focus on use cases and examples that guide the reader through each task.

In this initial version of the guide, you can learn how to:

- [Configure masters and nodes using Ansible](#)
- [Set limits for users and projects](#)
- [Determine which roles users get by default](#)
- [Control user permissions with roles](#)
- [Share templates across the cluster](#)
- [Create a cluster administrator account](#)
- [Configure authentication providers](#)
- [Certificate Management](#)

Your feedback on this guide would be greatly appreciated. You can let us know if you find it to be helpful, or if there is a topic you would like us to cover, by contacting openshift-docs@redhat.com.

CHAPTER 2. MASTER AND NODE CONFIGURATION

2.1. OVERVIEW

The master and node configuration files determine the make-up of your OpenShift Container Platform cluster, and define a range of options. These include overriding the default plug-ins, connecting to etcd, automatically creating service accounts, building image names, customizing project requests, configuring volume plug-ins, and much more.

This topic covers the many options available for customizing your OpenShift Container Platform masters and nodes, and shows you how to make changes to the configuration after installation.

The `/etc/origin/master/master-config.yaml` and `/etc/origin/node/node-config.yaml` files define a wide range of options that can be configured on the [OpenShift master](#) and [nodes](#). These options include overriding the default plug-ins, connecting to etcd, automatically creating service accounts, building image names, customizing project requests, configuring volume plug-ins, and much more.

2.2. PREREQUISITES

For testing environments deployed via the [quick install](#), one master should be sufficient. The quick installation method should not be used for production environments.

Production environments should be installed using the [advanced install](#). In production environments, it is a good idea to use [multiple masters](#) for the purposes of [high availability](#) (HA). A cluster architecture of three masters is recommended, and [HAproxy](#) is the recommended solution for this.

CAUTION

If etcd is being installed on the *master hosts*, you must configure your cluster to use at least three masters. It cannot use only two masters, because etcd would not be able to decide which one is authoritative. The only way to successfully run only two masters is if you install etcd on hosts other than the masters.

2.3. CONFIGURING MASTERS AND NODES

The method you use to configure your master and node configuration files must match the method that was used to install your OpenShift cluster. If you followed the:

- [Advanced installation](#) method using Ansible, then make your configuration changes [in the Ansible playbook](#).
- [Quick installation](#) method, then make your changes [manually in the configuration files](#) themselves.

2.3.1. Making Configuration Changes Using Ansible

For this section, familiarity with Ansible is assumed.

Only a portion of the available host configuration options are [exposed to Ansible](#). After an OpenShift Container Platform install, Ansible creates an inventory file with some substituted values. Modifying this inventory file and re-running the Ansible installer playbook is how you customize your OpenShift Container Platform cluster.

While OpenShift supports using Ansible as the advanced install method, using an Ansible playbook and inventory file, you can also use other management tools, such as [Puppet](#), [Chef](#), [Salt](#)).

Use Case: Configure the cluster to use HTTPasswd authentication



NOTE

- This use case assumes you have already set up [SSH keys](#) to all the nodes referenced in the playbook.
- The `htpasswd` utility is in the `httpd-tools` package:

```
# yum install httpd-tools
```

To modify the Ansible inventory and make configuration changes:

1. Open the `./hosts` inventory file:

Example 2.1. Sample inventory file:

```
[OSEv3:children]
masters
nodes

[OSEv3:vars]
ansible_ssh_user=cloud-user
ansible_become=true
deployment_type=openshift-enterprise

[masters]
ec2-52-6-179-239.compute-1.amazonaws.com openshift_ip=172.17.3.88
openshift_public_ip=52-6-179-239
openshift_hostname=master.example.com
openshift_public_hostname=ose3-master.public.example.com
containerized=True
[nodes]
ec2-52-6-179-239.compute-1.amazonaws.com openshift_ip=172.17.3.88
openshift_public_ip=52-6-179-239
openshift_hostname=master.example.com
openshift_public_hostname=ose3-master.public.example.com
containerized=True openshift_schedulable=False
ec2-52-95-5-36.compute-1.amazonaws.com openshift_ip=172.17.3.89
openshift_public_ip=52.3.5.36 openshift_hostname=node.example.com
openshift_public_hostname=ose3-node.public.example.com
containerized=True
```

2. Add the following new variables to the `[OSEv3:vars]` section of the file:

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
# Defining htpasswd users
```

```

openshift_master_htpasswd_users={'<name>': '<hashed-password>',
'<name>': '<hashed-password>'}
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>

```

For HTTPasswd authentication, you can use either the `openshift_master_htpasswd_users` variable to create the specified user(s) and password(s) or the `openshift_master_htpasswd_file` variable to specify a pre-generated flat file (the *htpasswd* file) with the users and passwords already created.

Because OpenShift Container Platform requires a hashed password to configure HTTPasswd authentication, you can use the `htpasswd` command, [as shown in the following section](#), to generate the hashed password(s) for your user(s) or to create the flat file with the users and associated hashed passwords.

The following example changes the authentication method from the default `deny all` setting to `htpasswd` and use the specified file to generate user IDs and passwords for the `jsmith` and `bloblaw` users.

```

# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'jsmith':
'$apr1$wIwXkFLI$bAygTKGmPOqaJftB', 'bloblaw':
'7IRJ$20DmeLoxf4I6sUEKfiA$2aDJqLJe'}
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>

```

3. Re-run the ansible playbook for these modifications to take effect:

```

$ ansible-playbook -b -i ./hosts ~/src/openshift-
ansible/playbooks/byo/config.yml

```

The playbook updates the configuration, and restarts the OpenShift master service to apply the changes.

You have now modified the master and node configuration files using Ansible, but this is just a simple use case. From here you can see which [master](#) and [node configuration](#) options are [exposed to Ansible](#) and customize your own Ansible inventory.

2.3.1.1. Using the `htpasswd` command

To configure the OpenShift Container Platform cluster to use HTTPasswd authentication, you need at least one user with a hashed password to include in the [inventory file](#).

You can:

- [Generate the username and password](#) to add directly to the `./hosts` inventory file.
- [Create a flat file](#) to pass the credentials to the `./hosts` inventory file.

To create a user and hashed password:

1. Run the following command to add the specified user:

```
$ htpasswd -n <user_name>
```



NOTE

You can include the **-b** option to supply the password on the command line:

```
$ htpasswd -nb <user_name> <password>
```

2. Enter and confirm a clear-text password for the user.

For example:

```
$ htpasswd -n myuser
New password:
Re-type new password:
myuser:$apr1$vdW.cI3j$WSKIOzUPs6Q
```

The command generates a hashed version of the password.

You can then use the hashed password when configuring [HTPasswd authentication](#). The hashed password is the string after the `:`. In the above example, you would enter:

```
openshift_master_htpasswd_users={'myuser':
'$apr1$wIwXkFLI$bAygTISk2eKGmqaJftB'}
```

To create a flat file with a user name and hashed password:

1. Execute the following command:

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```



NOTE

You can include the **-b** option to supply the password on the command line:

```
$ htpasswd -c -b <user_name> <password>
```

2. Enter and confirm a clear-text password for the user.

For example:

```
htpasswd -c users.htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

The command generates a file that includes the user name and a hashed version of the user's password.

You can then use the password file when configuring [HTPasswd authentication](#).



NOTE

For more information on the `htpasswd` command, see [HTPasswd Identity Provider](#).

2.3.2. Making Manual Configuration Changes

After installing OpenShift Container Platform using the [quick install](#), you can make modifications to the master and node configuration files to customize your cluster.

Use Case: Configure the cluster to use HTPasswd authentication

To manually modify a configuration file:

1. Open the configuration file you want to modify, which in this case is the `/etc/origin/master/master-config.yaml` file:
2. Add the following new variables to the `identityProviders` stanza of the file:

```
oauthConfig:
  ...
identityProviders:
- name: my_htpasswd_provider
  challenge: true
  login: true
  mappingMethod: claim
  provider:
    apiVersion: v1
    kind: HTPasswdPasswordIdentityProvider
    file: /path/to/users.htpasswd
```

3. Save your changes and close the file.
4. Restart the master for the changes to take effect:

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

You have now manually modified the master and node configuration files, but this is just a simple use case. From here you can see all the [master](#) and [node configuration](#) options, and further customize your own cluster by making further modifications.

2.4. CONFIGURATION OPTIONS

2.4.1. Master Configuration File Options

The table below contains the options available for configuring your OpenShift Container Platform `master-config.yaml` file. Use this table as a reference, and then follow the section on [making manual configuration changes](#) and substitute in whatever values you want to change.

Table 2.1. Master Configuration File Options

Option	Description
servingInfo	<p>Describes how to start serving. For example:</p> <pre>servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: ca.crt keyFile: master.server.key maxRequestsInFlight: 500 requestTimeoutSeconds: 3600</pre>
corsAllowedOrigins	Specifies the host name to use to access the API server from a web application.
apiLevels	A list of API levels that should be enabled on startup; for example, v1beta3 and v1 .
apiServerArguments	<p>Contains key value pairs that match the API server's command-line arguments and are passed directly to the Kubernetes API server. These are not migrated, but if you reference a value that does not exist, then the server will not start.</p> <pre>apiServerArguments: event-ttl: - "15m"</pre>
assetConfig	<p>If present, then the asset server starts based on the defined parameters. For example:</p> <pre>assetConfig: logoutURL: "" masterPublicURL: https://master.ose32.example.com:8443 publicURL: https://master.ose32.example.com:8443/console/ servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: "" keyFile: master.server.key maxRequestsInFlight: 0 requestTimeoutSeconds: 0</pre>
controllers	A list of the controllers that should be started. If set to none , then no controllers will start automatically. The default value is * which will start all controllers. When using * , you may exclude controllers by prepending a - in front of the controller name. No other values are recognized at this time.
pauseControllers	When set to true , this instructs the master to not automatically start controllers, but instead to wait until a notification to the server is received before launching them.

Option	Description
controllerLeaseTTL	Enables controller election, instructing the master to attempt to acquire a lease before controllers start, and renewing it within a number of seconds defined by this value. Setting this value as a non-negative forces pauseControllers=true . The value default is off (0, or omitted) and controller election can be disabled with -1.
admissionConfig	Contains admission control plug-in configuration. OpenShift has a configurable list of admission controller plug-ins that are triggered whenever API objects are created or modified. This option allows you to override the default list of plug-ins; for example, disabling some plug-ins, adding others, changing the ordering, and specifying configuration. Both the list of plug-ins and their configuration can be controlled from Ansible.
disabledFeatures	Lists features that should <i>not</i> be started. This is defined as omit empty because it is unlikely that you would want to manually disable features.
etcdStorageConfig	Contains information about how API resources are stored in etcd. These values are only relevant when etcd is the backing store for the cluster.
etcdClientInfo	<p>Contains information about how to connect to etcd. Specifies if etcd is run as embedded or non-embedded, and the hosts. The rest of the configuration is handled by the Ansible inventory. For example:</p> <pre> etcdClientInfo: ca: ca.crt certFile: master.etcd-client.crt keyFile: master.etcd-client.key urls: - https://m1.aos.example.com:4001 </pre>
kubernetesMasterConfig	Contains information about how to connect to kubelet's KubernetesMasterConfig. If present, then start the kubernetes master in this process.
etcdConfig	<p>If present, then etcd starts based on the defined parameters. For example:</p> <pre> etcdConfig: address: master.ose32.example.com:4001 peerAddress: master.ose32.example.com:7001 peerServingInfo: bindAddress: 0.0.0.0:7001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key servingInfo: bindAddress: 0.0.0.0:4001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key storageDirectory: /var/lib/origin/openshift.local.etcd </pre>

Option	Description
oauthConfig	<p>If present, then the /oauth endpoint starts based on the defined parameters. For example:</p> <pre> oauthConfig: assetPublicURL: https://master.ose32.example.com:8443/console/ grantConfig: method: auto identityProviders: - challenge: true login: true mappingMethod: claim name: httpasswd_all provider: apiVersion: v1 kind: HTTPasswdPasswordIdentityProvider file: /etc/origin/openshift-passwd masterCA: ca.crt masterPublicURL: https://master.ose32.example.com:8443 masterURL: https://master.ose32.example.com:8443 sessionConfig: sessionMaxAgeSeconds: 3600 sessionName: ssn sessionSecretsFile: /etc/origin/master/session-secrets.yaml tokenConfig: accessTokenMaxAgeSeconds: 86400 authorizeTokenMaxAgeSeconds: 500 </pre>
assetConfig	<p>If present, then the asset server starts based on the defined parameters. For example:</p> <pre> assetConfig: logoutURL: "" masterPublicURL: https://master.ose32.example.com:8443 publicURL: https://master.ose32.example.com:8443/console/ servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: "" keyFile: master.server.key maxRequestsInFlight: 0 requestTimeoutSeconds: 0 </pre>
dnsConfig	<p>If present, then start the DNS server based on the defined parameters. For example:</p> <pre> dnsConfig: bindAddress: 0.0.0.0:8053 bindNetwork: tcp4 </pre>

Option	Description
serviceAccountConfig	<p>Holds options related to service accounts:</p> <ul style="list-style-type: none"> • LimitSecretReferences (boolean): Controls whether or not to allow a service account to reference any secret in a namespace without explicitly referencing them. • ManagedNames (string): A list of service account names that will be auto-created in every namespace. If no names are specified, then the ServiceAccountsController will not be started. • MasterCA (string): The certificate authority for verifying the TLS connection back to the master. The service account controller will automatically inject the contents of this file into pods so that they can verify connections to the master. • PrivateKeyFile (string): Contains a PEM-encoded private RSA key, used to sign service account tokens. If no private key is specified, then the service account TokensController will not be started. • PublicKeyFiles (string): A list of files, each containing a PEM-encoded public RSA key. If any file contains a private key, then OpenShift uses the public portion of the key. The list of public keys is used to verify service account tokens; each key is tried in order until either the list is exhausted or verification succeeds. If no keys are specified, then service account authentication will not be available.
masterClients	<p>Holds all the client connection information for controllers and other system components:</p> <ul style="list-style-type: none"> • OpenShiftLoopbackKubeConfig (string): the .kubeconfig filename for system components to loopback to this master. • ExternalKubernetesKubeConfig (string): the .kubeconfig filename for proxying to Kubernetes.
imageConfig	<p>Holds options that describe how to build image names for system components:</p> <ul style="list-style-type: none"> • Format (string): Describes how to determine image names for system components • Latest (boolean): Defines whether to attempt to use the latest system component images or the latest release.

Option	Description
imagePolicyConfig	<p>Controls limits and behavior for importing images:</p> <ul style="list-style-type: none"> • MaxImagesBulkImportedPerRepository (integer): Controls the number of images that are imported when a user does a bulk import of a Docker repository. This number is set low to prevent users from importing large numbers of images accidentally. This can be set to -1 for no limit. • DisableScheduledImport (boolean): Allows scheduled background import of images to be disabled. • ScheduledImageImportMinimumIntervalSeconds (integer): The minimum number of seconds that can elapse between when image streams scheduled for background import are checked against the upstream repository. The default value is 900 (15 minutes). • MaxScheduledImageImportsPerMinute (integer): The maximum number of image streams that can be imported in the background, per minute. The default value is 60. This can be set to -1 for unlimited imports. <p>This can be controlled with the Ansible inventory</p>
policyConfig	<p>Holds information about where to locate critical pieces of bootstrapping policy. This is controlled by Ansible, so you may not need to modify this:</p> <ul style="list-style-type: none"> • BootstrapPolicyFile (string): Points to a template that contains roles and rolebindings that will be created if no policy object exists in the master namespace. • OpenShiftSharedResourcesNamespace (string): The namespace where shared OpenShift resources are located, such as shared templates. • OpenShiftInfrastructureNamespace (string): The namespace where OpenShift infrastructure resources are located, such as controller service accounts.

Option	Description
projectConfig	<p>Holds information about project creation and defaults:</p> <ul style="list-style-type: none"> • DefaultNodeSelector (string): Holds the default project node label selector. • ProjectRequestMessage (string): The string presented to a user if they are unable to request a project via the projectrequest API endpoint. • ProjectRequestTemplate (string): The template to use for creating projects in response to projectrequest. It is in the format <code><namespace>/<template></code>. It is optional, and if it is not specified, a default template is used. • SecurityAllocator: Controls the automatic allocation of UIDs and MCS labels to a project. If nil, allocation is disabled: <ul style="list-style-type: none"> ◦ mcsAllocatorRange (string): Defines the range of MCS categories that will be assigned to namespaces. The format is <code><prefix>/<numberOfLabels>[, <maxCategory>]</code>. The default is <code>s0/2</code> and will allocate from <code>c0</code> → <code>c1023</code>, which means a total of 535k labels are available. If this value is changed after startup, new projects may receive labels that are already allocated to other projects. The prefix may be any valid SELinux set of terms (including user, role, and type). However, leaving the prefix at its default allows the server to set them automatically. For example, <code>s0 : /2</code> would allocate labels from <code>s0:c0,c0</code> to <code>s0:c511,c511</code> whereas <code>s0 : /2, 512</code> would allocate labels from <code>s0:c0,c0,c0</code> to <code>s0:c511,c511,511</code>. ◦ mcsLabelsPerProject (integer): Defines the number of labels to reserve per project. The default is <code>5</code> to match the default UID and MCS ranges. ◦ uidAllocatorRange (string): Defines the total set of Unix user IDs (UIDs) automatically allocated to projects, and the size of the block each namespace gets. For example, <code>1000-1999/10</code> would allocate ten UIDs per namespace, and would be able to allocate up to 100 blocks before running out of space. The default is to allocate from 1 billion to 2 billion in 10k blocks, which is the expected size of ranges for container images when user namespaces are started.
routingConfig	<p>Holds information about routing and route generation:</p> <ul style="list-style-type: none"> • Subdomain (string): The suffix appended to <code>\$service.\$namespace.</code> to form the default route hostname. Can be controlled via Ansible with <code>openshift_master_default_subdomain</code>. Example: <pre> routingConfig: subdomain: "" </pre>

Option	Description
networkConfig	<p>To be passed to the compiled-in-network plug-in. Many of the options here can be controlled in the Ansible inventory.</p> <ul style="list-style-type: none"> • NetworkPluginName (string) • ClusterNetworkCIDR (string) • HostSubnetLength (unsigned integer) • ServiceNetworkCIDR (string) • ExternalIPNetworkCIDRs (string array): Controls which values are acceptable for the service external IP field. If empty, no external IP may be set. It can contain a list of CIDRs which are checked for access. If a CIDR is prefixed with !, then IPs in that CIDR are rejected. Rejections are applied first, then the IP is checked against one of the allowed CIDRs. For security purposes, you should ensure this range does not overlap with your nodes, pods, or service CIDRs. <p>For Example:</p> <pre>networkConfig: clusterNetworkCIDR: 10.3.0.0/16 hostSubnetLength: 8 networkPluginName: example/openshift-ovs-subnet # serviceNetworkCIDR must match kubernetesMasterConfig.servicesSubnet serviceNetworkCIDR: 179.29.0.0/16</pre>
volumeConfig	<p>Contains options for configuring volume plug-ins in the master node:</p> <ul style="list-style-type: none"> • DynamicProvisioningEnabled (boolean): Default value is true, and toggles dynamic provisioning off when false.

2.4.2. Node Configuration File Options

The table below contains the options available for configuring your OpenShift Container Platform *node-config.yaml* file. Use this table as a reference, and then follow the section on [making manual configuration changes](#) and substitute in whatever values you want to change.

Table 2.2. Node Configuration File Options

Option	Description
nodeName	<p>The value of the nodeName (string) is used to identify this particular node in the cluster. If possible, this should be your fully qualified hostname. If you are describing a set of static nodes to the master, then this value must match one of the values in the list.</p>

Option	Description
nodeIP	A node may have multiple IPs. This specifies the IP to use for pod traffic routing. If left unspecified, a network look-up is performed on the nodeName, and the first non-loopback address is used.
servingInfo	Describes how to start serving.
masterKubeConfig	The filename for the .kubeconfig file that describes how to connect this node to the master.
dnsDomain	Holds the domain suffix.
dnsIP	(string) Contains the IP. Can be controlled with openshift_dns_ip in the Ansible inventory.
networkPluginName, omit empty	Deprecated and maintained for backward compatibility, use NetworkConfig.NetworkPluginName instead.
networkConfig	Provides network options for the node: <ul style="list-style-type: none"> • NetworkPluginName (string): Specifies the networking plug-in. • MTU (unsigned integer): Maximum transmission unit for the network packets.
volumeDirectory	The directory that volumes will be stored under.
imageConfig	Holds options that describe how to build image names for system components.
allowDisableDocker	If this is set to true , then the Kubelet will ignore errors from Docker. This means that a node can start on a machine that does not have Docker started.
podManifestConfig	Holds the configuration for enabling the Kubelet to create pods based from a manifest file or files placed locally on the node.
authConfig	Holds authn/authz configuration options.
dockerConfig	Holds Docker-related configuration options.
kubeletArguments, omit empty	Key-value pairs that are passed directly to the Kubelet that matches the Kubelet's command line arguments. These are not migrated or validated, so if you use them, then they may become invalid. Use caution, because these values override other settings in the node configuration that may cause invalid configurations.

Option	Description
proxyArguments, omitEmpty	ProxyArguments are key-value pairs that are passed directly to the Proxy that matches the Proxy's command-line arguments. These are not migrated or validated, so if you use them they may become invalid. Use caution, because these values override other settings in the node configuration that may cause invalid configurations.
iptablesSyncPeriod	(string) How often iptables rules are refreshed. This can be controlled with openshift_node_iptables_sync_period from the Ansible inventory.
volumeConfig	Contains options for configuring volumes on the node. It can be used to apply a filesystem quota if the underlying volume directory is on XFS with grpquota enabled .

CHAPTER 3. USER AND ROLE MANAGEMENT

3.1. LIMITING AND MONITORING USERS AND PROJECTS

3.1.1. Setting Limits for Users and Projects

How can I create limits for users and projects?

You can place limits within your OpenShift cluster using [ResourceQuotas](#) and [LimitRanges](#). These quotas and limits allow you to control pod and container limits, object counts, and compute resources. Currently, these limits and quotas only apply to projects and not to users. However, you can make a quota-like [limit on how many project requests a user can make](#).

Creating a quota in a project to limit the number of pods

To create a quota in the "awesomeproject" that limits the number of pods that can be created to a maximum of 10:

1. Create a **resource-quota.yaml** file with the following contents:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "10"
```

2. Create the quota using the file you just wrote to apply it to the "awesomeproject":

```
$ oc create -f resource-quota.yaml -n awesomeproject
```

After the quota has been in effect for a little while, you can view the usage statistics for the hard limit set on pods.

3. If required, list the quotas defined in the project to see the names of all defined quotas:

```
$ oc get quota -n awesomeproject
NAME              AGE
resource-quota    39m
```

4. Describe the resource quota for which you want statistics:

```
$ oc describe quota resource-quota -n awesomeproject
Name:      resource-quota
Namespace: awesomeproject
Resource   Used Hard
-----
pods       3 10
```

5. Optionally, you can [configure the quota synchronization period](#), which controls how long to wait before restoring quota usage after resources are deleted.

6. If you want to remove an active quota to no longer enforce the limits of a project:

```
$ oc delete quota <quota_name>
```

3.1.1.1. Configuration Options

The [procedure above](#) is just a basic example. The following are references to all the available options for limits and quotas:

This *LimitRange* example explains all the [container limits](#) and [pod limits](#) that you can place within your project:

Example 3.1. Limit Range Object Definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits" 1
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "200m" 4
        memory: "6Mi" 5
    - type: "Container"
      max:
        cpu: "2" 6
        memory: "1Gi" 7
      min:
        cpu: "100m" 8
        memory: "4Mi" 9
      default:
        cpu: "300m" 10
        memory: "200Mi" 11
      defaultRequest:
        cpu: "200m" 12
        memory: "100Mi" 13
      maxLimitRequestRatio:
        cpu: "10" 14
```

- 1 The name of the limit range object.
- 2 The maximum amount of CPU that a pod can request on a node across all containers.
- 3 The maximum amount of memory that a pod can request on a node across all containers.
- 4 The minimum amount of CPU that a pod can request on a node across all containers.
- 5 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](#).

Example 3.2. 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.

These **ResourceQuota** examples explain all the [Object Counts](#) and [Compute Resources](#) that you can place within your project:

Example 3.3. *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.

Example 3.4. *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.

Example 3.5. *compute-resources.yaml*

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4" 1
    requests.cpu: "1" 2
    requests.memory: 1Gi 3
    limits.cpu: "2" 4
    limits.memory: 2Gi 5

```

- 1 The total number of pods in a non-terminal state that can exist in the project.
- 2 Across all pods in a non-terminal state, the sum of CPU requests cannot exceed 1 core.

- 3 Across all pods in a non-terminal state, the sum of memory requests cannot exceed 1Gi.
- 4 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed 2 cores.
- 5 Across all pods in a non-terminal state, the sum of memory limits cannot exceed 2Gi.

3.1.2. Limiting the Number of Projects a User Can Have

You can [limit the number of projects](#) that a user may request by categorizing users with label selectors with the `oc label` command. A label selector consists of the label name and the label value:

```
label=value
```

Once users are labeled, you must [modify the default project template](#) in the *master-config.yaml* file [using an admission control plug-in](#). This allows some users to create more projects than others, and you can define different values (or levels) for each label.

Limiting how many projects a user can request by defining three different privilege levels

The label is named `level`, and the possible values are `bronze`, `silver`, `gold`, and `platinum`. Platinum users do not have a maximum number of project requests, gold users can request up to 10 projects, silver users up to 7 projects, bronze users up to 5 projects, and any users without a label are by default only allowed 2 projects.

Each user can only have one value per label. For example, a user cannot be both `gold` and `silver` for the `level` label. However, when configuring the *master-config.yaml* file, you could select users that have any value for a label with a wildcard; for example, `level=*`.

To define privilege levels for project requests:

1. Apply label selectors to users. For example, to apply the `level` label selector with a value of `bronze`:

```
$ oc label user <user_name> level=bronze
```

Repeat this step for all bronze users, and then for the other levels.

2. Optionally, verify the previous step by viewing the list of labeled users for each value:

```
$ oc get users -l level=bronze
$ oc get users -l level=silver
$ oc get users -l level=gold
$ oc get users -l level=platinum
```

If you need to remove a label from a user to make a correction:

```
$ oc label user <user_name> level-
```

3. Modify the *master-config.yaml* file to define project limits for this label with the numbers stated in this use case. Find the `admissionConfig` line and create the configuration below it:

```
admissionConfig:
```

```

pluginConfig:
  ProjectRequestLimit:
    configuration:
      apiVersion: v1
      kind: ProjectRequestLimitConfig
      limits:
        - selector:
            level: platinum
        - selector:
            level: gold
            maxProjects: 10
        - selector:
            level: silver
            maxProjects: 7
        - selector:
            level: bronze
            maxProjects: 5
        - maxProjects: 2

```

4. Restart the master host for the changes to take effect.

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

NOTE

If you use a custom project template to limit the number of projects per user, then you must ensure that you keep the modifications by including the following:

```
ProjectRequester = "openshift.io/requester"
```

Ownership is established using the `openshift.io/requester` annotation, so your custom project template must have the same annotation.

3.1.3. Controlling and Monitoring Resource Usage

If you configure a project to have ResourceQuota restrictions, then the amount of the defined quota currently being used is stored on the ResourceQuota object itself. In that case, you could check the amount of used resources, such as CPU usage:

```
$ oc get quota
```

However, this would not tell you what is actually being consumed. To determine what is actually being consumed, use the `oc describe` command:

```
$ oc describe quota <quota-name>
```

Alternatively, you can set up [cluster metrics](#) for more detailed statistics.

3.2. DETERMINING WHICH ROLES USERS GET BY DEFAULT

When a user first logs in, there is a default set of permissions that is applied to that user. The scope of permissions that a user can have is controlled by the [various types](#) of [roles within OpenShift](#):

- **ClusterRoles**
- **ClusterRoleBindings**
- **Roles** (project-scoped)
- **RoleBindings** (project-scoped)

You may want to modify the default set of permissions. In order to do this, it's important to understand the default groups and roles assigned, and to be aware of the roles and users bound to [each project](#) or [the entire cluster](#).

3.2.1. Leveraging Default Groups

There are special groups that are assigned to users. You can target users with these groups, but you cannot modify them. These special groups are as follows:

Group	Description
system:authenticated	This is assigned to all users who are identifiable to the API. Everyone who is not system:anonymous (the user) is in this group.
system:authenticated:oauth	This is assigned to all users who have identified using an oauth token issued by the embedded oauth server. This is not applied to service accounts (they use service account tokens), or certificate users.
system:unauthenticated	This is assigned to users who have not presented credentials. Invalid credentials are rejected with a 401 error, so this is specifically users who did not try to authenticate at all.

You may find it helpful to target users with the special groups listed above. For example, you could share a template with all users by granting **system:authenticated** access to the template.

The "default" permissions of users are defined by which roles are bound to the **system:authenticated** and **system:authenticated:oauth** groups. As mentioned above, you are not able to modify membership to these groups, but you can [change the roles bound to these groups](#). For example, to bind a role to the **system:authenticated** group for all projects in the cluster:

```
$ oadm policy add-cluster-role-to-group <role> system:authenticated
```

Currently, by default the **system:authenticated** and **system:authenticated:oauth** groups receive the following roles:

Role	Description
shared-resource-viewer	For the openshift project. Allows users to see templates and pull images.

Role	Description
basic-user	For the the entire cluster. Allows users to see their own account, check for information about requesting projects, see which projects they can view, and check their own permissions.
self-provisioner	Allows users to request projects.
system:oauth-token-deleter	Allows users to delete any oauth token for which they know the details.
cluster-status	Allows users to see which APIs are enabled, and basic API server information such as versions.
system:webhook	Allows users to hit the webhooks for a build if they have enough additional information.

3.2.2. Viewing Roles and Users for a Project

To view a list of all users that are bound to the project and their roles:

```
$ oc get rolebindings
NAME                                ROLE                                USERS    GROUPS
SERVICE ACCOUNTS  SUBJECTS
system:image-pullers  /system:image-puller
system:serviceaccounts:asdfasdf4asdf
admin                /admin                            jsmith
system:deployers     /system:deployer
deployer
system:image-builders /system:image-builder
builder
```

3.2.3. Viewing Roles and Users for the Cluster

To view a list of users and what they have access to across the entire cluster:

```
$ oc get clusterrolebindings
NAME                                ROLE                                SERVICE
USERS                                GROUPS                                SUBJECTS
ACCOUNTS
system:job-controller              /system:job-controller
openshift-infra/job-controller
system:build-controller           /system:build-controller
openshift-infra/build-controller
system:node-admins               /system:node-admin
system:master system:node-admins
registry-registry-role           /system:registry
default/registry
system:pv-provisioner-controller  /system:pv-provisioner-
```

```

controller
openshift-infra/pv-provisioner-controller
basic-users /basic-user
system:authenticated
system:namespace-controller /system:namespace-
controller
openshift-infra/namespace-controller
system:discovery-binding /system:discovery
system:authenticated, system:unauthenticated
system:build-strategy-custom-binding /system:build-strategy-
custom system:authenticated
cluster-status-binding /cluster-status
system:authenticated, system:unauthenticated
system:webhooks /system:webhook
system:authenticated, system:unauthenticated
system:gc-controller /system:gc-controller
openshift-infra/gc-controller
cluster-readers /cluster-reader
system:cluster-readers
system:pv-recycler-controller /system:pv-recycler-
controller
openshift-infra/pv-recycler-controller
system:daemonset-controller /system:daemonset-
controller
openshift-infra/daemonset-controller
cluster-admins /cluster-admin
system:admin system:cluster-admins
system:hpa-controller /system:hpa-controller
openshift-infra/hpa-controller
system:build-strategy-source-binding /system:build-strategy-
source system:authenticated
system:replication-controller /system:replication-
controller
openshift-infra/replication-controller
system:sdn-readers /system:sdn-reader
system:nodes
system:build-strategy-docker-binding /system:build-strategy-
docker system:authenticated
system:routers /system:router
system:routers
system:oauth-token-deleters /system:oauth-token-
deleter system:authenticated,
system:unauthenticated
system:node-proxiers /system:node-proxier
system:nodes
system:nodes /system:node
system:nodes
self-provisioners /self-provisioner
system:authenticated:oauth
system:service-serving-cert-controller /system:service-serving-
cert-controller
openshift-infra/service-serving-cert-controller
system:registries /system:registry
system:registries
system:pv-binder-controller /system:pv-binder-
controller

```

```
openshift-infra/pv-binder-controller
system:build-strategy-jenkinspipeline-binding /system:build-strategy-
jenkinspipeline                             system:authenticated
system:deployment-controller                /system:deployment-
controller
openshift-infra/deployment-controller
system:masters                             /system:master
system:masters
system:service-load-balancer-controller    /system:service-load-
balancer-controller
openshift-infra/service-load-balancer-controller
```

These commands can generate huge lists, so you may want to pipe the output into a text file that you can search through more easily.

3.3. CONTROLLING USER PERMISSIONS WITH ROLES

You can define [roles](#) (or permissions) for a user before their initial log in so they can start working immediately. You can assign many different types of roles to users such as admin, basic-user, self-provisioner, and cluster-reader.

For a complete list of all available roles:

```
$ oadm policy
```

The following section includes examples of some common operations related to adding (binding) and removing roles from users and groups. For a complete list of available local policy operations, see [Managing Role Bindings](#).

3.3.1. Adding a Role to a User

To bind a role to a user for the current project:

```
$ oadm policy add-role-to-user <role> <user_name>
```

You can specify a project with the `-n` flag.

3.3.2. Removing a Role from a User

To remove a role from a user for the current project:

```
$ oadm policy remove-role-from-user <role> <user_name>
```

You can specify a project with the `-n` flag.

3.3.3. Adding a Cluster Role to a User for All Projects

To bind a [cluster role](#) to a user for all projects:

```
$ oadm policy add-cluster-role-to-user <role> <user_name>
```

3.3.4. Removing a Cluster Role from a User for All Projects

To remove a [cluster role](#) from a user for all projects:

```
$ oadm policy remove-cluster-role-from-user <role> <user_name>
```

3.3.5. Adding a Role to a Group

To bind a role to a specified group in the current project:

```
$ oadm policy add-role-to-group <role> <groupname>
```

You can specify a project with the `-n` flag.

3.3.6. Removing a Role from a Group

To remove a role from a specified group in the current project:

```
$ oadm policy remove-role-from-group <role> <groupname>
```

You can specify a project with the `-n` flag.

3.3.7. Adding a Cluster Role to a Group for All Projects

To bind a role to a specified group for all projects in the cluster:

```
$ oadm policy add-cluster-role-to-group <role> <groupname>
```

3.3.8. Removing a Cluster Role from a Group for All Projects

To remove a role from a specified group for all projects in the cluster:

```
$ oadm policy remove-cluster-role-from-group <role> <groupname>
```

3.4. SHARING TEMPLATES FOR USE IN PROJECTS ACROSS THE CLUSTER

Templates are project-scoped resources, so you cannot create them to be readily available at a cluster level. The easiest way to share templates across the entire cluster is with the **openshift** project, which by default is already set up to share templates. The templates can be annotated, and are displayed in the web console where users can access them. Users have **get** access only to the templates and images in this project, via the **shared-resource-viewer** role.

The **shared-resource-viewer** role exists to allow templates to be shared across project boundaries. Users with this role have the ability to see all existing templates and pull images from that project. However, the user still needs to know which project to look in, because they will not be able to view the project in their `oc get projects` list.

By default, this role is granted to the **system:authenticated** group in the **openshift** project. This allows users to process the specified template from the **openshift** project and create the items in the current project:

```
$ oc process openshift//<template-name> | oc create -f -
```

You can also add the registry viewer role to a user, allowing them to view and pull images from a project:

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

3.5. CREATING A CLUSTER ADMINISTRATOR USER

Cluster administrator is a very powerful role, which has ultimate control within the cluster, including the power to destroy that cluster. You can grant this role to other users if they absolutely need to have ultimate control. However, you may first want to examine the other available roles if you do not want to create such a powerful user. For example, `admin` is a constrained role that has the power to do many things inside of their project, but cannot affect (or destroy) the entire cluster.

3.5.1. Creating an Administrator Within a Project

To create a basic administrator role within a project:

```
$ oadm policy add-role-to-user admin <user_name> -n <project_name>
```

3.5.2. Creating a Cluster Administrator

To create a cluster administrator with ultimate control over the cluster:



WARNING

Be very careful when granting cluster administrator role to a user. Ensure that the user truly needs that level of power within the cluster. When OpenShift is first installed, a certificate based user is created and the credentials are saved in ***admin.kubeconfig***. This cluster administrator user can do absolutely anything to any resource on the entire cluster, which can result in destruction if not used carefully.

```
$ oadm policy add-cluster-role-to-user cluster-admin <user_name>
```

CHAPTER 4. AUTHENTICATION

4.1. OVERVIEW

OpenShift Container Platform supports many different authentication methods, as defined in [Configuring Authentication](#):

- [Basic Authentication \(Remote\)](#)
- [Request Header](#)
- [Keystone](#)
- [LDAP](#)
- [GitHub](#)

4.2. BASIC AUTHENTICATION (REMOTE)

Basic Authentication is a generic backend integration mechanism that allows users to log in to OpenShift Container Platform with credentials validated against a remote identity provider.

CAUTION

Basic Authentication must use an HTTPS connection to the remote server in order to prevent potential snooping of the user ID and password, and to prevent man-in-the-middle attacks.

With `BasicAuthPasswordIdentityProvider` configured, users send their username and password to OpenShift Container Platform, which then validates those credentials against a remote server by making a server-to-server request, passing the credentials as a Basic Auth header. This requires users to send their credentials to OpenShift Container Platform during login.



NOTE

This only works for username/password login mechanisms, and OpenShift Container Platform must be able to make network requests to the remote authentication server.

4.2.1. Configuring Authentication on the Master

1. If you have:

- Already completed the installation of Openshift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ mkdir basicauthconfig; cp master-config.yaml basicauthconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

For example:

```
$ openshift start master --public-  
master=https://myapiserver.com:8443 --write-  
config=basicauthconfig
```



NOTE

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new *master-config.yaml* file's **identityProviders** stanza.
3. Copy [the example BasicAuthPasswordIdentityProvider configuration](#) and paste it to replace the existing stanza.
4. Make the following modifications to the **identityProviders** stanza:
 - a. Set the provider **name** to something unique and relevant to your deployment. This name is prefixed to the returned user ID to form an identity name.
 - b. If required, [set mappingMethod](#) to control how mappings are established between the provider's identities and user objects.
 - c. Specify the **HTTPS url** to use to connect to a server that accepts credentials in Basic authentication headers.
 - d. Optionally, set the **ca** to the certificate bundle to use in order to validate server certificates for the configured URL, or leave it empty to use the system-trusted roots.
 - e. Optionally, remove or set the **certFile** to the client certificate to present when making requests to the configured URL.
 - f. If **certFile** is specified, then you must set the **keyFile** to the key for the client certificate.
5. Save your changes and close the file.
6. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-  
config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their Basic authentication credentials.

4.2.2. Troubleshooting

The most common issue relates to network connectivity to the backend server. For simple debugging, run `curl` commands on the master. To test for a successful login, replace the `<user>` and

<password> in the following example command with valid credentials. To test an invalid login, replace them with false credentials.

```
curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key
/path/to/client.key -u <user>:<password> -v
https://www.example.com/remote-idp
```

Successful responses

A **200** status with a **sub** (subject) key indicates success:

```
{"sub": "userid"}
```

The subject must be unique to the authenticated user, and must not be able to be modified.

A successful response may optionally provide additional data, such as:

- A display name using the **name** key:

```
{"sub": "userid", "name": "User Name", ...}
```

- An email address using the **email** key:

```
{"sub": "userid", "email": "user@example.com", ...}
```

- A preferred user name using the **preferred_username** key:

```
{"sub": "014fbff9a07c", "preferred_username": "bob", ...}
```

The **preferred_username** key is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift Container Platform user for the authenticated identity.

Failed responses

- A **401** response indicates failed authentication.
- A non-**200** status or the presence of a non-empty **"error"** key indicates an error:
{"error": "Error message"}

4.2.3. Verifying Users

Once one or more users have logged in, you can run **oc get users** to view a list of users and verify that users were created successfully.

From here, you might want to learn how to [control user roles](#).

4.3. REQUEST HEADER AUTHENTICATION

Configuring Request Header authentication allows users to log in to OpenShift Container Platform using request header values, such as **X-Remote-User**. It is typically used in combination with an authenticating proxy, which authenticates the user and then provides OpenShift Container Platform with the user's identity via a request header value. This is similar to how [the remote user plug-in](#) in

[OpenShift Enterprise 2](#) allowed administrators to provide Kerberos, LDAP, and many other forms of enterprise authentication. The benefit of this configuration is that user credentials can be handled by the proxy and never seen by OpenShift.

The proxy must be able to make network requests to the OpenShift Container Platform server. Unauthenticated login attempts are redirected to a configured proxy URL. The proxy can authenticate browser clients regardless of how it is configured, but it must (currently) use either Basic Auth or Kerberos in order to work with the `oc` CLI tooling.

For users to authenticate using this identity provider, they must access `https://<master>/oauth/authorize` via an authenticating proxy. You can configure the OAuth server to redirect unauthenticated requests to the proxy.

4.3.1. Configuring Authentication on the Master

1. If you have:

- Already completed the installation of OpenShift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ mkdir reqheadauthconfig; cp master-config.yaml  
reqheadauthconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-  
config=<directory>
```

For example:

```
$ openshift start master --public-  
master=https://myapiserver.com:8443 --write-  
config=reqheadauthconfig
```



NOTE

If you are installing with Ansible, then you must add the `identityProvider` configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new `master-config.yaml` file's `identityProviders` stanza.
3. View [the example RequestHeaderIdentityProvider configuration](#) and use it as a guide to replace the existing stanza.
4. Modify the `identityProviders` stanza based on which headers you plan to pass in.
 - a. Set the provider name to something unique and relevant to your deployment. This name is prefixed to the returned user ID to form an identity name.

- b. If required, `set mappingMethod` to control how mappings are established between the provider's identities and user objects.
- c. Set the `challenge` parameter to `true` to redirect unauthenticated requests from clients expecting `WWW-Authenticate` challenges.
- d. Set the `provider.challengeURL` parameter to the proxy URL to which to send clients expecting `WWW-Authenticate` challenges, like the `oc` CLI client. This parameter can include the `${url}` and `${query}` tokens in the query portion of the URL.
- e. Set the `login` parameter to `true` to redirect unauthenticated requests from clients expecting login flows.
- f. Set the `provider.loginURL` parameter to the proxy URL to which to send clients expecting login flows, like web browser clients. This parameter can include the `${url}` and `${query}` tokens in the query portion of the URL.
- g. Set the `clientCA` parameter to the certificate bundle to use to check incoming requests for a valid client certificate before the request's headers are checked for a user name.



WARNING

If you expect unauthenticated requests to reach the OAuth server, a `clientCA` parameter (and optionally, `clientCommonNames`) should be set for this identity provider. Otherwise, any direct request to the OAuth server can impersonate any identity from this provider, merely by setting a request header.

- h. Optionally, set the `clientCommonNames` parameter to a list of Common Names (`cn`). If set, a valid client certificate with a Common Name (`cn`) in the specified list must be presented before the request headers are checked for user names. If empty, then any Common Name is allowed. This must be used in combination with `clientCA`.
- i. Set the `headers` parameter to the header names to check, in order, for the user identity. The first header containing a value is used as the identity. This parameter is required and is case-insensitive.
- j. Optionally, set the `emailHeaders` parameter to the header names to check, in order, for an email address. The first header containing a value is used as the email address. This parameter is case-insensitive.
- k. Optionally, set the `nameHeaders` parameter to the header names to check, in order, for a display name. The first header containing a value is used as the display name. This parameter is case-insensitive.
- l. Optionally, set the `preferredUsernameHeaders` parameter to the header names to check, in order, for a preferred user name (if different than the immutable identity determined from the headers specified in `headers`). The first header containing a value is used as the preferred user name when provisioning. This parameter is case-insensitive.

5. Save your changes and close the file.
6. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be redirected to the authenticating proxy, which will authenticate the user.

4.3.2. Creating Users with Request Header Authentication

You do not create users in OpenShift Container Platform when integrating with an external authentication provider, such as the system that the proxy server is using as an authentication server. That server is the system of record, meaning that users are defined there, and any user with a valid user name for the configured authentication server can log in.

To add a user to OpenShift Container Platform, the user must exist on the system the proxy is using as an authentication server, and if required you must add the users to that system.

4.3.3. Verifying Users

Once one or more users have logged in, you can run `oc get users` to view a list of users and verify that users were created successfully.

From here, you might want to examine [configuring LDAP failover](#) for an example of Request Header authentication in use with Apache. You can also learn how to [control user roles](#).

4.4. KEYSTONE AUTHENTICATION

[Keystone](#) is an OpenStack project that provides identity, token, catalog, and policy services. You can integrate your OpenShift Container Platform cluster with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database. Once configured, this configuration allows users to log in to OpenShift Container Platform with their Keystone credentials.

4.4.1. Configuring Authentication on the Master

1. If you have:

- Already completed the installation of Openshift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ cd /etc/origin/master
$ mkdir keystoneconfig; cp master-config.yaml keystoneconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```


For example:

```
$ openshift start master --public-  
master=https://myapiserver.com:8443 --write-config=keystoneconfig
```



NOTE

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new *keystoneconfig/master-config.yaml* file's **identityProviders** stanza.
3. Copy [the example KeystonePasswordIdentityProvider configuration](#) and paste it to replace the existing stanza.
4. Make the following modifications to the **identityProviders** stanza:
 - a. Change the provider name ("my_keystone_provider") to match your Keystone server. This name is prefixed to provider user names to form an identity name.
 - b. If required, [change mappingMethod](#) to control how mappings are established between the provider's identities and user objects.
 - c. Change the **domainName** to the domain name of your OpenStack Keystone server. In Keystone, usernames are domain-specific. Only a single domain is supported.
 - d. Specify the **url** to use to connect to your OpenStack Keystone server.
 - e. Optionally, change the **ca** to the certificate bundle to use in order to validate server certificates for the configured URL.
 - f. Optionally, change the **certFile** to the client certificate to present when making requests to the configured URL.
 - g. If **certFile** is specified, then you must change the **keyFile** to the key for the client certificate.
5. Save your changes and close the file.
6. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-  
config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their Keystone credentials.

4.4.2. Creating Users with Keystone Authentication

You do not create users in OpenShift Container Platform when integrating with an external

authentication provider, such as, in this case, Keystone. Keystone is the system of record, meaning that users are defined in a Keystone database, and any user with a valid Keystone user name for the configured authentication server can log in.

To add a user to OpenShift Container Platform, the user must exist in the Keystone database, and if required you must create a new Keystone account for the user.

4.4.3. Verifying Users

Once one or more users have logged in, you can run `oc get users` to view a list of users and verify that users were created successfully:

Example 4.1. Output of `oc get users` command

```
$ oc get users
NAME                                UID                                FULL NAME
IDENTITIES
bobsmith        a0c1d95c-1cb5-11e6-a04a-002186a28631    Bob Smith
keystone:bobsmith 1
```

- 1 Identities in OpenShift Container Platform are comprised of the identity provider name prefixed to the Keystone username.

From here, you might want to learn how to [control user roles](#).

4.5. LDAP AUTHENTICATION

LDAP uses bind operations to authenticate applications, and you can integrate your OpenShift Container Platform cluster to use LDAPv3 authentication. Configuring LDAP authentication allows users to log in to OpenShift Container Platform with their LDAP credentials.

During authentication, the LDAP directory is searched for an entry that matches the provided user name. If a single unique match is found, a simple bind is attempted using the distinguished name (DN) of the entry plus the provided password.

**WARNING**

The basic authentication configuration covered by this topic is not enough to create a secure LDAP authentication solution for OpenShift Container Platform. It has a single point of failure, meaning that if the single LDAP authentication server became unavailable then all OpenShift Container Platform operations requiring authentication would also be unavailable.

Additionally, this basic configuration has no access control of its own; all LDAP users matching the configured filter are able to log into OpenShift Container Platform.

With the [SSSD failover setup](#), FreeIPA and Active Directory can also set rules to specifically restrict which users can and cannot access OpenShift Container Platform.

The following advanced topic begin where this basic LDAP authentication topic ends and describe [configuring LDAP failover](#).

4.5.1. Configuring Authentication on the Master

1. If you have:

- Already completed the installation of Openshift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ cd /etc/origin/master
$ mkdir ldapconfig; cp master-config.yaml ldapconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-
config=<directory>
```

For example:

```
$ openshift start master --public-
master=https://myapiserver.com:8443 --write-config=ldapconfig
```

**NOTE**

If you are installing with Ansible, then you must add the **identityProvider** configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

2. Edit the new `master-config.yaml` file's `identityProviders` stanza.

3. Copy [the example LDAPPasswordIdentityProvider configuration](#) and paste it to replace the existing stanza.
4. Make the following modifications to the `identityProviders` stanza:
 - a. Change the provider name (`"my_ldap_provider"`) to something unique and relevant to your deployment. This name is prefixed to the returned user name to form an identity name.
 - b. If required, [change `mappingMethod`](#) to control how mappings are established between the provider's identities and user objects.
 - c. Change `id` to the attribute to use as the identity, which must be unique and immutable within the identity provider. This option can accept multiple attributes. If more than one is specified, they will be checked in order and the first non-empty attribute will be used. At least one attribute is required. If none of the listed attribute have a value, then authentication fails.
 - d. Change `email` to the attribute to use as the email address. This option can accept multiple attributes. If more than one is specified, they will be checked in order and the first non-empty attribute will be used.
 - e. Change `name` to the attribute to use as the display name. This option can accept multiple attributes. If more than one is specified, they will be checked in order and the first non-empty attribute will be used.
 - f. Optionally, change `preferredUsername` to the attribute to use as the preferred OpenShift Container Platform username when provisioning a user for this identity. If unspecified, the `id` attribute is used as the preferred username. This option can accept multiple attributes. If more than one is specified, they will be checked in order and the first non-empty attribute will be used.

The attribute you select as the `preferredUsername` should still be unique, even within the identity provider. The `preferredUsername` attribute is only used when provisioning the user for the initial login. Afterward, the existing OpenShift Container Platform user is looked up by their identity provider ID, in case the `preferredUsername` attribute changes.

Using `preferredUsername` is helpful when the immutable `id` attribute is not a human-recognizable value, and there is another attribute with a value that is more recognizable to the user. For example, if the `id` is something like `"e43adf8cc243"`, you could set `preferredUsername` to `login`, which could have potentially mutable values, such as `"bobsmith"`.
 - g. Change the `ca` to the certificate bundle to use in order to validate server certificates for the configured URL. If empty, system trusted roots are used. This setting only applies if `insecure: false`. If the LDAP server requires a different certificate chain, this attribute should contain the filesystem path of that certificate or certificate bundle.
 - h. If required, modify the `insecure` parameter. The default is `false`, and this must be `false` when using `ldaps://` URLs. When `false`, `ldaps://` URLs connect using TLS, and `ldap://` URLs are upgraded to TLS. When `true`, no TLS connection is made to the server, however, setting this to `true` creates an invalid configuration for LDAP.
 - i. Define an RFC 2255 URL that [specifies the LDAP host and search parameters](#) to use.
5. Save your changes and close the file.

6. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --master-config=
<path/to/modified/config>/master-config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their LDAP credentials.

4.5.2. Creating Users with LDAP Authentication

You do not create users in OpenShift Container Platform when integrating with an external authentication provider, such as, in this case, LDAP. LDAP is the system of record, meaning that users are defined in LDAP, and any user with a valid LDAP ID for the configured authentication server can log in.

To add a user to OpenShift Container Platform, the user must exist in the LDAP system, and if required you must create a new LDAP account for the user.

4.5.3. Verifying Users

Once one or more users have logged in, you can run `oc get users` to view a list of users and verify that users were created successfully:

Example 4.2. Output of `oc get users` command

```
$ oc get users
NAME          UID                                FULL NAME
IDENTITIES
bobsmith      166a2367-33fc-11e6-bb22-4ccc6a0ad630  Bob Smith
ldap_provider:uid=bsmith,ou=users,dc=example,dc=com 1
```

- 1 Identities in OpenShift Container Platform are comprised of the identity provider name prefixed to the LDAP distinguished name (DN).

From here, you might want to learn how to [control user roles](#).

4.6. GITHUB AUTHENTICATION

GitHub uses OAuth, and you can integrate your OpenShift Container Platform cluster to use that OAuth authentication. OAuth basically facilitates a token exchange flow.

Configuring GitHub authentication allows users to log in to OpenShift Container Platform with their GitHub credentials. To prevent anyone with any GitHub user ID from logging in to your OpenShift Container Platform cluster, you can restrict access to only those in specific GitHub organizations.

4.6.1. Registering the Application on GitHub

1. On GitHub, click [Settings](#) → [OAuth applications](#) → [Developer applications](#) → [Register an application](#) to navigate to the page for a [new OAuth application](#).

2. Type an application name. For example: **My OpenShift Install**
3. Type a homepage URL. For example: <https://myapiserver.com:8443>
4. Optionally, type an application description.
5. Type the authorization callback URL, where the end of the URL contains the identity provider **name** (defined in the `identityProviders` stanza of the [master configuration file](#), which you configure in the next section of this topic):

```
<apiserver>/oauth2callback/<identityProviderName>
```

For example:

```
https://myapiserver.com:8443/oauth2callback/github/
```

6. Click **Register application**. GitHub provides a Client ID and a Client Secret. Keep this window open so you can copy these values and paste them into the master configuration file.

4.6.2. Configuring Authentication on the Master

1. If you have:

- Already completed the installation of OpenShift, then copy the `/etc/origin/master/master-config.yaml` file into a new directory; for example:

```
$ cd /etc/origin/master
$ mkdir githubconfig; cp master-config.yaml githubconfig
```

- Not yet installed OpenShift Container Platform, then start the OpenShift Container Platform API server, specifying the hostname of the (future) OpenShift Container Platform master and a directory to store the configuration file created by the start command:

```
$ openshift start master --public-master=<apiserver> --write-
config=<directory>
```

For example:

```
$ openshift start master --public-
master=https://myapiserver.com:8443 --write-config=githubconfig
```



NOTE

If you are installing with Ansible, then you must add the `identityProvider` configuration to the Ansible playbook. If you use the following steps to modify your configuration manually after installing with Ansible, then you will lose any modifications whenever you re-run the install tool or upgrade.

**NOTE**

Using `openshift start master` on its own would auto-detect host names, but GitHub must be able to redirect to the exact host name that you specified when registering the application. For this reason, you cannot auto-detect the ID because it might redirect to the wrong address. Instead, you must specify the hostname that web browsers use to interact with your OpenShift Container Platform cluster.

2. Edit the new *master-config.yaml* file's `identityProviders` stanza.
3. Copy the example `GitHubIdentityProvider` configuration and paste it to replace the existing stanza.
4. Make the following modifications to the `identityProviders` stanza:
 - a. Change the provider `name` to match the callback URL you configured on GitHub. For example, if you defined the callback URL as <https://myapiserver.com:8443/oauth2callback/github/> then the `name` must be `github`.
 - b. Change `clientId` to the Client ID from GitHub that you [registered previously](#).
 - c. Change `clientSecret` to the Client Secret from GitHub that you [registered previously](#).
 - d. Change `organizations` to include a list of one or more GitHub organizations to which a user must have membership in order to authenticate. If specified, only GitHub users that are members of at least one of the listed organizations will be allowed to log in. If this is not specified, then any person with a valid GitHub account can log in.
5. Save your changes and close the file.
6. Start the OpenShift Container Platform API server, specifying the configuration file you just modified:

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

Once configured, any user logging in to the OpenShift Container Platform web console will be prompted to log in using their GitHub credentials. On their first login, the user must click **authorize application** to permit GitHub to use their username, password, and organization membership with OpenShift Container Platform. The user is then redirected back to the web console.

4.6.3. Creating Users with GitHub Authentication

You do not create users in OpenShift Container Platform when integrating with an external authentication provider, such as, in this case, GitHub. GitHub is the system of record, meaning that users are defined by GitHub, and any user belonging to a specified organization can log in.

To add a user to OpenShift Container Platform, you must add that user to an approved organization on GitHub, and if required create a new GitHub account for the user.

4.6.4. Verifying Users

Once one or more users have logged in, you can run `oc get users` to view a list of users and verify that users were created successfully:

Example 4.3. Output of `oc get users` command

```
$ oc get users
NAME          UID                                FULL NAME
IDENTITIES
bobsmith      433b5641-066f-11e6-a6d8-acfc32c1ca87  Bob Smith
github:873654 1
```

- 1 Identities in OpenShift Container Platform are comprised of the identity provider name and GitHub's internal numeric user ID. This way, if a user changes their GitHub username or e-mail they can still log in to OpenShift Container Platform instead of relying on the credentials attached to the GitHub account. This creates a stable login.

From here, you might want to learn how to [control user roles](#).

CHAPTER 5. CERTIFICATE MANAGEMENT

5.1. OVERVIEW

Over the lifetime of an OpenShift Container Platform cluster, certificates will enter various phases of their lifecycle. The following procedures describe how to manage various parts of that lifecycle.

5.2. CHANGING AN APPLICATION'S SELF-SIGNED CERTIFICATE TO CA-SIGNED CERTIFICATE

Some application templates create a self-signed certificate that is then directly presented by the application to clients. As an example, by default and as part of the OpenShift Container Platform Ansible installer deployment process, the metrics deployer creates self-signed certificates.

These self-signed certificates are not recognized by browsers. To mitigate this issue, use a publicly signed certificate, then configure it to re-encrypt traffic with the self-signed certificate.

1. Delete the existing route:

```
$ oc delete route hawkular-metrics -n openshift-infra
```

With the route deleted, the certificates that will be used in the new route with the re-encrypt strategy must be assembled from the existing wildcard and self-signed certificates created by the metrics deployer. The following certificates must be available:

- Wildcard CA certificate
- Wildcard private key
- Wildcard certificate
- Hawkular CA certificate

Each certificate must be available as a file on the file system for the new route.

You can retrieve the Hawkular CA and store it in a file by executing the following command:

```
$ oc get secrets hawkular-metrics-certificate -n openshift-infra \
  \
  -o jsonpath='{.data.hawkular-metrics-ca\.certificate}' | base64 \
  -d > hawkular-internal-ca.crt
```

2. Locate the wildcard private key, certificate, and CA certificate. Place each into a separate file, such as *wildcard.key*, *wildcard.crt*, and *wildcard.ca*.
3. Create the new re-encrypt route:

```
$ oc create route reencrypt hawkular-metrics-reencrypt \
  -n openshift-infra \
  --hostname hawkular-metrics.ocp.example.com \
  --key wildcard.key \
  --cert wildcard.crt \
```

```
--ca-cert wildcard.ca \  
--service hawkular-metrics \  
--dest-ca-cert hawkular-internal-ca.crt
```