OpenShift Enterprise 3.0

Administrator Guide

OpenShift Enterprise 3.0 Administration

Last Updated: 2018-10-18

OpenShift Enterprise 3.0 Administrator Guide

OpenShift Enterprise 3.0 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 Administration topics cover the basics of installing and running OpenShift in your environment. Configuration, management, and logging are also covered. Use these topics to quickly set up your OpenShift environment and configure it based on your organizational needs.

Table of Contents

CHAPTER 1. OVERVIEW

OpenShift Administration topics cover the basics of installing and running OpenShift in your environment. Configuration, management, and logging are also covered. Use these topics to quickly set up your OpenShift environment and configure it based on your organizational needs.

CHAPTER 2. MASTER AND NODE CONFIGURATION

2.1. OVERVIEW

The **openshift start** command is used to launch OpenShift servers. The command and its subcommands (**master** to launch a master server and **node** to launch a node server) all take a limited set of arguments that are sufficient for launching servers in a development or experimental environment.

However, these arguments are insufficient to describe and control the full set of configuration and security options that are necessary in a production environment. To provide those options, it is necessary to use the dedicated master and node configuration files.

Master configuration files and node configuration files are fully specified with no default values. Therefore, any empty value indicates that you want to start up with an empty value for that parameter. This makes it easy to reason about exactly what your configuration is, but it also makes it difficult to remember all of the options to specify. To make this easier, the configuration files can be created with the **--write-config** option and then used with the **--config** option.

2.2. CREATING NEW CONFIGURATION FILES

For masters, the **openshift start** command accepts options that indicate that it should simply write the configuration files that it would have used, then terminate. For nodes, a configuration file can be written using the **oadm create-node-config** command. Creating new configuration files is useful to get a starting point for defining your configuration.

The following commands write the relevant launch configuration file(s), certificate files, and any other necessary files to the specified **--write-config** or **--node-dir** directory.

To create configuration files for an all-in-one server (a master and a node on the same host) in the specified directory:

```
$ openshift start --write-config=/openshift.local.config
```

To create a master configuration file and other required files in the specified directory:

```
$ openshift start master --write-config=/openshift.local.config/master
```

To create a node configuration file and other related files in the specified directory:

```
$ oadm create-node-config --node-dir=/openshift.local.config/node-
<node_hostname> --node=<node_hostname> --hostnames=<hostname>,<ip_address>
```

For the **--hostnames** option in the above command, use a comma-delimited list of every host name or IP address you want server certificates to be valid for. The above command also assumes that certificate files are located in an ***openshift.local.config/master/*** directory. If they are not, you can include options to specify their location. Run the command with the **-h** option to see details.

2.3. LAUNCHING SERVERS USING CONFIGURATION FILES

Once you have modified the master and/or node configuration files to your specifications, you can use them when launching servers by specifying them as an argument. Keep in mind that if you specify a configuration file, none of the other command line options you pass are respected.

To launch an all-in-one server using a master configuration and a node configuration file:

```
$ openshift start --master-config=/openshift.local.config/master/master-
config.yaml --node-config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

To launch a master server using a master configuration file:

```
$ openshift start master --config=/openshift.local.config/master/master-
config.yaml
```

To launch a node server using a node configuration file:

```
$ openshift start node --config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

2.4. MASTER CONFIGURATION FILES

The following ***master-config.yaml*** file is a sample master configuration file taken at a point in time. You can create a new master configuration file to see the valid options for your installed version of OpenShift.

Sample Master Configuration File

```
apiLevels:
- v1beta3
- v1
apiVersion: v1
assetConfig:
  logoutURL: ""
  masterPublicURL: https://10.0.2.15:8443
  publicURL: https://10.0.2.15:8443/console/
  servingInfo:
    bindAddress: 0.0.0.0:8443
    certFile: master.server.crt
    clientCA: ""
    keyFile: master.server.key
    maxRequestsInFlight: 0
    requestTimeoutSeconds: 0
controllers: '*'
corsAllowedOrigins:
- 10.0.2.15:8443
- 127.0.0.1
- localhost
dnsConfig:
  bindAddress: 0.0.0.0:53
etcdClientInfo:
  ca: ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
  - https://10.0.2.15:4001
etcdConfig:
  address: 10.0.2.15:4001
  peerAddress: 10.0.2.15:7001
  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: /root/openshift.local.etcd
etcdStorageConfig:
  kubernetesStoragePrefix: kubernetes.io
  kubernetesStorageVersion: v1
  openShiftStoragePrefix: openshift.io
  openShiftStorageVersion: v1
imageConfig:
  format: openshift/origin-${component}:${version}
  latest: false
kind: MasterConfig
kubeletClientInfo:
  ca: ca.crt
  certFile: master.kubelet-client.crt
  keyFile: master.kubelet-client.key
  port: 10250
kubernetesMasterConfig:
  apiLevels:
  - v1beta3
  - v1
  apiServerArguments: null
  controllerArguments: null
  masterCount: 1
  masterIP: 10.0.2.15
  podEvictionTimeout: 5m
  schedulerConfigFile: ""
  servicesNodePortRange: 30000-32767
  servicesSubnet: 172.30.0.0/16
  staticNodeNames: []
masterClients:
  externalKubernetesKubeConfig: ""
  openshiftLoopbackKubeConfig: openshift-master.kubeconfig
masterPublicURL: https://10.0.2.15:8443
networkConfig:
  clusterNetworkCIDR: 10.1.0.0/16
  hostSubnetLength: 8
  networkPluginName: ""
  serviceNetworkCIDR: 172.30.0.0/16
oauthConfig:
  assetPublicURL: https://10.0.2.15:8443/console/
  grantConfig:
    method: auto
  identityProviders:
  - challenge: true
    login: true
    name: anypassword
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider
  masterPublicURL: https://10.0.2.15:8443
  masterURL: https://10.0.2.15:8443
  sessionConfig:
    sessionMaxAgeSeconds: 300
    sessionName: ssn
    sessionSecretsFile: ""
  tokenConfig:
    accessTokenMaxAgeSeconds: 86400
    authorizeTokenMaxAgeSeconds: 300
policyConfig:
  bootstrapPolicyFile: policy.json
  openshiftInfrastructureNamespace: openshift-infra
  openshiftSharedResourcesNamespace: openshift
projectConfig:
  defaultNodeSelector: ""
  projectRequestMessage: ""
  projectRequestTemplate: ""
  securityAllocator:
    mcsAllocatorRange: s0:/2
    mcsLabelsPerProject: 5
    uidAllocatorRange: 1000000000-1999999999/10000
routingConfig:
  subdomain: router.default.svc.cluster.local
serviceAccountConfig:
  managedNames:
  - default
  - builder
  - deployer
  masterCA: ca.crt
  privateKeyFile: serviceaccounts.private.key
  publicKeyFiles:
  - serviceaccounts.public.key
servingInfo:
  bindAddress: 0.0.0.0:8443
  certFile: master.server.crt
  clientCA: ca.crt
  keyFile: master.server.key
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 3600
```

2.5. NODE CONFIGURATION FILES

The following ***node-config.yaml*** file is a sample node configuration file taken at a point in time. You can create a new node configuration file to see the valid options for your installed version of OpenShift.

Example 2.1. Sample Node Configuration File

```
allowDisabledDocker: true
apiVersion: v1
dnsDomain: cluster.local
dnsIP: 10.0.2.15
dockerConfig:
  execHandlerName: native
```

```
imageConfig:
  format: openshift/origin-${component}:${version}
  latest: false
kind: NodeConfig
masterKubeConfig: node.kubeconfig
networkConfig:
  mtu: 1450
  networkPluginName: ""
nodeIP: ""
nodeName: node1.example.com
podManifestConfig: 1
  path: "/path/to/pod-manifest-file" 2
  fileCheckIntervalSeconds: 30 3
servingInfo:
  bindAddress: 0.0.0.0:10250
  certFile: server.crt
  clientCA: node-client-ca.crt
  keyFile: server.key
volumeDirectory: /root/openshift.local.volumes
```

1. Allows pods to be placed directly on certain set of nodes, or on all nodes without going through the scheduler. You can then use pods to perform the same administrative tasks and support the same services on each node.

2. Specifies the path for the pod manifest file or directory. If it is a directory, then it is expected to contain one or more manifest files. This is used by the Kubelet to create pods on the node.

3. This is the interval (in seconds) for checking the manifest file for new data. The interval must be a positive value.

CHAPTER 3. MANAGING NODES

3.1. 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.

3.2. LISTING NODES

To list all nodes that are known to the master:

```
$ oc get nodes
NAME                 LABELS
STATUS
node1.example.com    kubernetes.io/hostname=node1.example.com      Ready
node2.example.com    kubernetes.io/hostname=node2.example.com      Ready
```

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 3.1. 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
```

```
Labels:   kubernetes.io/hostname=node1.example.com
CreationTimestamp: Wed, 10 Jun 2015 17:22:34 +0000
Conditions:
  Type  Status LastHeartbeatTime   LastTransitionTime   Reason     Message
  Ready  True  Wed, 10 Jun 2015 19:56:16 +0000  Wed, 10 Jun 2015 17:22:34
+0000  kubelet is posting ready status
Addresses: 127.0.0.1
Capacity:
 memory: 1017552Ki
 pods:  100
 cpu:  2
Version:
 Kernel Version:  3.17.4-301.fc21.x86_64
 OS Image:   Fedora 21 (Twenty One)
 Container Runtime Version: docker://1.6.0
 Kubelet Version:  v0.17.1-804-g496be63
 Kube-Proxy Version:  v0.17.1-804-g496be63
ExternalID:   node1.example.com
Pods:    (2 in total)
  docker-registry-1-9yyw5
  router-1-maytv
No events.
```

3.3. ADDING NODES

To add nodes to your existing OpenShift 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.

3.4. DELETING NODES

When you delete a node with the CLI, although the node object is deleted in Kubernetes, the pods that exist on the node itself are not deleted. However, the pods cannot be accessed by OpenShift. The behavior around deleting nodes and pods with the CLI is under active development.

To delete a node:

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

3.5. 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
```

3.6. LISTING PODS ON NODES

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

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

To list all or selected pods on selected nodes:

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

3.7. 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:

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

For example:

```
$ oadm 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:

```
$ oadm 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.

3.8. 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:

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

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

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

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

```
$ oadm 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.

CHAPTER 4. ROUTING FROM EDGE LOAD BALANCERS

4.1. OVERVIEW

Pods inside of an OpenShift cluster are only reachable via their IP addresses on the cluster network. An edge load balancer can be used to accept traffic from outside networks and proxy the traffic to pods inside the OpenShift cluster. In cases where the load balancer is not part of the cluster network, routing becomes a hurdle as the internal cluster network is not accessible to the edge load balancer.

To solve this problem where the OpenShift cluster is using OpenShift SDN as the cluster networking solution, there are two ways to achieve network access to the pods.

4.2. INCLUDING THE LOAD BALANCER IN THE SDN

If possible, run an OpenShift node instance on the load balancer itself that uses OpenShift SDN as the networking plug-in. This way, the edge machine gets its own Open vSwitch bridge that the SDN automatically configures to provide access to the pods and nodes that reside in the cluster. The *routing table* is dynamically configured by the SDN as pods are created and deleted, and thus the routing software is able to reach the pods.

Mark the load balancer machine as an unschedulable node so that no pods end up on the load balancer itself:

```
$ oadm manage-node <load_balancer_hostname> --schedulable=false
```

If the load balancer comes packaged as a Docker container, then it is even easier to integrate with OpenShift: Simply run the load balancer as a pod with the host port exposed. The pre-packaged HAProxy router in OpenShift runs in precisely this fashion.

4.3. ESTABLISHING A TUNNEL USING A RAMP NODE

In some cases, the previous solution is not possible. For example, an **F5 BIG-IP®** host cannot run an OpenShift node instance or the OpenShift SDN because **F5®** uses a custom, incompatible Linux kernel and distribution.

Instead, to enable **F5 BIG-IP®** to reach pods, you can choose an existing node within the cluster network as a *ramp node* and establish a tunnel between the **F5 BIG-IP®** host and the designated ramp node. Because it is otherwise an ordinary OpenShift node, the ramp node has the necessary configuration to route traffic to any pod on any node in the cluster network. The ramp node thus assumes the role of a gateway through which the **F5 BIG-IP®** host has access to the entire cluster network.

Following is an example of establishing an **ipip** tunnel between an **F5 BIG-IP®** host and a designated ramp node.

On the F5 BIG-IP® host:

1. Set the following variables:

```
# F5_IP=10.3.89.66 1
# RAMP_IP=10.3.89.89 2
# TUNNEL_IP1=10.3.91.216 3
# CLUSTER_NETWORK=10.1.0.0/16 4
```

1 2 The **F5_IP** and **RAMP_IP** variables refer to the **F5 BIG-IP®** host's and the ramp node's IP addresses, respectively, on a shared, internal network.

3 An arbitrary, non-conflicting IP address for the **F5®** host's end of the **ipip** tunnel.

4 The overlay network CIDR that the OpenShift SDN uses to assign addresses to pods.

2. Delete any old route, self, tunnel and SNAT pool:

```
# tmsh delete net route $CLUSTER_NETWORK || true
# tmsh delete net self SDN || true
# tmsh delete net tunnels tunnel SDN || true
# tmsh delete ltm snatpool SDN_snatpool || true
```

3. Create the new tunnel, self, route and SNAT pool and use the SNAT pool in the virtual servers:

```
# tmsh create net tunnels tunnel SDN \
    \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsh create net self SDN \{ address \
    ${TUNNEL_IP1}/24 allow-service all vlan SDN \}
# tmsh create net route $CLUSTER_NETWORK interface SDN
# tmsh create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1 }
# tmsh modify ltm virtual  ose-vserver source-address-translation {
type snat pool SDN_snatpool }
# tmsh modify ltm virtual  https-ose-vserver source-address-
translation { type snat pool SDN_snatpool }
```

On the ramp node:

1. Set the following variables:

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217  1
```

1 A second, arbitrary IP address for the ramp node's end of the **ipip** tunnel.

2. Delete any old tunnel:

```
# ip tunnel del tun1 || true
```

3. Create the **ipip** tunnel on the ramp node, using a suitable L2-connected interface (e.g., **eth0**):

```
# ip tunnel add tun1 mode ipip \
    remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

4. SNAT the tunnel IP with an unused IP from the SDN subnet:

```
# source /etc/openshift-sdn/config.env
# subaddr=$(echo $OPENSHIFT_SDN_TAP1_ADDR | cut -d "." -f 1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

5. Assign this **RAMP_SDN_IP** as an additional address to **tun0** (the local SDN's gateway):

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

6. Modify the OVS rules for SNAT:

```
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"cookie=0x999,ip,nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN_
IP},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"cookie=0x999,ip,nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_I
P1},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "cookie=0x999, table=0, arp, arp_tpa=${RAMP_SDN_IP},
actions=output:2"
```

7. Mark the ramp node as an unschedulable node so that no pods end up on the ramp node itself:

```
$ oadm manage-node <ramp_node_hostname> --schedulable=false
```

4.3.1. Configuring a Highly-Available Ramp Node

You can use OpenShift's **ipfailover** feature, which uses **keepalived** internally, to make the ramp node highly available from **F5 BIG-IP®**'s point of view. To do so, first bring up two nodes, for example called **ramp-node-1** and **ramp-node-2**, on the same L2 subnet.

Then, choose some unassigned IP address from within the same subnet to use for your virtual IP, or *VIP*. This will be set as the **RAMP_IP** variable with which you will configure your tunnel on **F5 BIG-IP®**.

For example, suppose you are using the **10.20.30.0/24** subnet for your ramp nodes, and you have assigned **10.20.30.2** to **ramp-node-1** and **10.20.30.3** to **ramp-node-2**. For your VIP, choose some unassigned address from the same **10.20.30.0/24** subnet, for example **10.20.30.4**. Then, to configure **ipfailover**, mark both nodes with a label, such as **f5rampnode**:

```
$ oc label node ramp-node-1 f5rampnode=true
$ oc label node ramp-node-2 f5rampnode=true
```

Similar to instructions from the **ipfailover** documentation, you must now create a service account and add it to the **privileged** SCC. First, create the **f5ipfailover** service account:

```
$ echo '
    { "kind": "ServiceAccount",
      "apiVersion": "v1",
      "metadata": { "name": "f5ipfailover" }
    }
  ' | oc create -f -
```

Next, you can manually edit the **privileged** SCC and add the **f5ipfailover** service account, or you can script editing the **privileged** SCC if you have **`jq`** installed. To manually edit the **privileged** SCC, run:

```
$ oc edit scc privileged
```

Then add the **f5ipfailover** service account in form **system:serviceaccount:<project>:<name>** to the **`users`** section:

```
...
users:
- system:serviceaccount:openshift-infra:build-controller
- system:serviceaccount:default:router
- system:serviceaccount:default:f5ipfailover
```

Alternatively, to script editing **privileged** SCC if you have **`jq`** installed, run:

```
$ oc get scc privileged -o json |
    jq '.users |= .+ ["system:serviceaccount:default:f5ipfailover"]' |
      oc replace scc -f -
```

Finally, configure **ipfailover** using your chosen VIP (the **`RAMP_IP`** variable) and the **f5ipfailover** service account, assigning the VIP to your two nodes using the **f5rampnode** label you set earlier:

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 1
# oadm ipfailover <name-tag> \
    --virtual-ips=$RAMP_IP \
    --interface=$IFNAME \
    --watch-port=0 \
    --replicas=2 \
    --service-account=f5ipfailover  \
    --selector='f5rampnode=true'
```

1 The interface where **`RAMP_IP`** should be configured.

With the above setup, the VIP (the **`RAMP_IP`** variable) is automatically re-assigned when the ramp node host that currently has it assigned fails.

CHAPTER 5. AGGREGATING CONTAINER LOGS

5.1. OVERVIEW

As an OpenShift administrator, you may want to view the logs from all containers in one user interface. The currently supported method for aggregating container logs in OpenShift Enterprise is using a centralized file system. Additional supported methods are planned for inclusion in future releases.

NOTE

As packaging improvements are made, these instructions will be simplified.

5.2. USING A CENTRALIZED FILE SYSTEM

This method reads all container logs and forwards them to a central server for storage on the file system.

5.2.1. Installing fluentd (td-agent) on Nodes

Perform the following steps on each node to install and configure **fluentd** (**td-agent**):

1. Run the following commands:

```
# export RPM=td-agent-2.2.0-0.x86_64.rpm
# curl https://packages.treasuredata.com/2/redhat/7/x86_64/$RPM \
    -o /tmp/$RPM
# yum localinstall /tmp/$RPM
# /opt/td-agent/embedded/bin/gem install fluent-plugin-kubernetes
# mkdir -p /etc/td-agent/config.d
# chown td-agent:td-agent /etc/td-agent/config.d
```

2. Create a directory to house the logs:

```
# mkdir -p /var/log/td-agent/tmp
# chown td-agent:td-agent /var/log/td-agent/tmp
```

   To allow **td-agent** access to the containers logs, create the ***/etc/sysconfig/td-agent*** file and ensure it contains the following:

```
DAEMON_ARGS=
TD_AGENT_ARGS="/usr/sbin/td-agent --log /var/log/td-agent/td-agent.log --use-v1-config"
```

3. Add the following line to the ***/etc/td-agent/td-agent.conf*** file:

```
 @include config.d/*.conf
```

4. Create the ***/etc/td-agent/config.d/kubernetes.conf*** file with the following contents:

```
  <source>
   type tail
   path /var/lib/docker/containers/*/*-json.log
   pos_file /var/log/td-agent/tmp/fluentd-docker.pos
   time_format %Y-%m-%dT%H:%M:%S
   tag docker.*
   format json
   read_from_head true
  </source>

  <match docker.var.lib.docker.containers.*.*.log>
   type kubernetes
   container_id ${tag_parts[5]}
   tag docker.${name}
  </match>

  <match kubernetes>
   type copy
   <store>
     type forward
     send_timeout 60s
     recover_wait 10s
     heartbeat_interval 1s
     phi_threshold 16
     hard_timeout 60s
     log_level trace
     require_ack_response true
     heartbeat_type tcp
     <server>
       name logging_name ❶
       host host_name ❷
       port 24224
       weight 60
     </server>

     <secondary>
       type file
       path /var/log/td-agent/forward-failed
     </secondary>
   </store>

   <store>
     type file
     path /var/log/td-agent/containers.log
     time_slice_format %Y%m%d
     time_slice_wait 10m
     time_format %Y%m%dT%H%M%S%z
     compress gzip
     utc
   </store>
  </match>
```

❶ The name for the master that will be used during logging.

❷ The IP or a DNS resolvable name used to access the master.

5. Enable **fluentd**:

```
# systemctl enable td-agent
# systemctl start td-agent
```

TIP

Any errors are logged in the ***/var/log/td-agent/td-agent.log*** file.

5.2.2. Optional Method to Verify Working Nodes

You can optionally set up the master to be the aggregator to test and verify that the nodes are working properly.

1. Install **fluentd** (**td-agent**) on the master:

   ```
   # export RPM=td-agent-2.2.0-0.x86_64.rpm
   # curl https://packages.treasuredata.com/2/redhat/7/x86_64/$RPM \
       -o /tmp/$RPM
   # yum localinstall /tmp/$RPM
   # mkdir -p /etc/td-agent/config.d
   # chown td-agent:td-agent /etc/td-agent/config.d
   ```

2. Ensure port **24224** is open on the master’s firewall to allow the nodes access.

3. Configure **fluentd** to aggregate container logs by adding the following line to the ***/etc/td-agent/td-agent.conf*** file:

   ```
     @include config.d/*.conf
   ```

4. Create the ***/etc/td-agent/config.d/kubernetes.conf*** file with the following contents:

   ```
       <match kubernetes.**>
           type file
           path /var/log/td-agent/containers.log
           time_slice_format %Y%m%d
           time_slice_wait 10m
           time_format %Y%m%dT%H%M%S%z
           compress gzip
           utc
       </match>
   ```

5. Enable **fluentd**:

   ```
   # systemctl enable td-agent
   # systemctl start td-agent
   ```

   TIP

   Any errors are logged in the ***/var/log/td-agent/td-agent.log*** file.

You should now find all the containers' logs available on the master in the ***/var/log/td-agent/containers.log*** file.

CHAPTER 6. CONFIGURING AUTHENTICATION

6.1. OVERVIEW

The OpenShift master includes a built-in OAuth server. Developers and administrators obtain OAuth access tokens to authenticate themselves to the API.

As an administrator, you can configure OAuth using a master configuration file to specify an identity provider. If you installed OpenShift using the Quick Installation or Advanced Installation method, the Deny All identity provider is used by default, which denies access for all user names and passwords. To allow access, you must choose a different identity provider and configure the master configuration file appropriately (located at ***/etc/openshift/master/master-config.yaml*** by default).

When running a master without a configuration file, the Allow All identity provider is used by default, which allows any non-empty user name and password to log in. This is useful for testing purposes. To use other identity providers, or to modify any token, grant, or session options, you must run the master from a configuration file.

6.2. IDENTITY PROVIDERS

You can configure the master for authentication using your desired identity provider by modifying the master configuration file. The following sections detail the identity providers supported by OpenShift.

There are three parameters common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
challenge	When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header. Not supported by all identity providers. To prevent cross-site request forgery (CSRF) attacks against browser clients Basic authentication challenges are only sent if a **X-CSRF-Token** header is present on the request. Clients that expect to receive Basic **WWW-Authenticate** challenges should set this header to a non-empty value.
login	When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider. Not supported by all identity providers.

6.2.1. Allow All

Set **AllowAllPasswordIdentityProvider** in the **identityProviders** stanza to allow any non-empty user name and password to log in. This is the default identity provider when running OpenShift without a master configuration file.

Example 6.1. Master Configuration Using AllowAllPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
```

```
  - name: my_allow_provider 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider
```

1 This provider name is prefixed to provider user names to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

6.2.2. Deny All

Set **DenyAllPasswordIdentityProvider** in the **`identityProviders`** stanza to deny access for all user names and passwords.

Example 6.2. Master Configuration Using DenyAllPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_deny_provider 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: DenyAllPasswordIdentityProvider
```

1 This provider name is prefixed to provider user names to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

6.2.3. HTPasswd

Set **HTPasswdPasswordIdentityProvider** in the **`identityProviders`** stanza to validate user names and passwords against a flat file generated using **`htpasswd`**.

NOTE

The **htpasswd** utility is in the **httpd-tools** package:

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

Only MD5, bcrypt, and SHA encryption types are supported. MD5 encryption is recommended, and is the default for **htpasswd**. Plaintext and crypt hashes are not currently supported.

The flat file is re-read if its modification time changes, without requiring a server restart.

To create the file, run:

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

To add or update a login to the file, run:

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

To remove a login from the file, run:

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

Example 6.3. Master Configuration Using HTPasswdPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /path/to/users.htpasswd 4
```

1 This provider name is prefixed to provider user names to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **WWW-Authenticate** challenge header for this provider.

3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4 File generated using **htpasswd**.

6.2.4. LDAP Authentication

Set **LDAPPasswordIdentityProvider** in the **identityProviders** stanza to validate user names and passwords against an LDAPv3 server, using simple bind authentication.

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. Here are the steps taken:

1. Generate a search filter by combining the attribute and filter in the configured **`url`** with the user-provided user name.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny access.
3. Attempt to bind to the LDAP server using the DN of the entry retrieved from the search, and the user-provided password.
4. If the bind is unsuccessful, deny access.
5. If the bind is successful, build an identity using the configured attributes as the identity, email address, display name, and preferred user name.

The configured **`url`** is an RFC 2255 URL, which specifies the LDAP host and search parameters to use. The syntax of the URL is:

```
ldap://host:port/basedn?attribute?scope?filter
```

For the above example:

URL Component	Description
`ldap`	For regular LDAP, use the string **`ldap`**. For secure LDAP (LDAPS), use **`ldaps`** instead.
`host:port`	The name and port of the LDAP server. Defaults to **`localhost:389`** for ldap and **`localhost:636`** for LDAPS.
`basedn`	The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but it could also specify a subtree in the directory.
`attribute`	The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use **`uid`**. It is recommended to choose an attribute that will be unique across all entries in the subtree you will be using.
`scope`	The scope of the search. Can be either either **`one`** or **`sub`**. If the scope is not provided, the default is to use a scope of **`sub`**.
`filter`	A valid LDAP search filter. If not provided, defaults to **`(objectClass=*)`**

When doing searches, the attribute, filter, and provided user name are combined to create a search filter that looks like:

```
(&(<filter>)(<attribute>=<username>))
```

For example, consider a URL of:

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

When a client attempts to connect using a user name of **bob**, the resulting search filter will be **`(&(enabled=true)(cn=bob))`**.

If the LDAP directory requires authentication to search, specify a **`bindDN`** and **`bindPassword`** to use to perform the entry search.

Example 6.4. Master Configuration Using LDAPPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
        id: 4
        - dn
        email: 5
        - mail
        name: 6
        - cn
        preferredUsername: 7
        - uid
      bindDN: "" 8
      bindPassword: "" 9
      ca: my-ldap-ca-bundle.crt 10
      insecure: false 11
      url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 12
```

1 This provider name is prefixed to the returned user ID to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4 List of attributes to use as the identity. First non-empty attribute is used. At least one attribute is required. If none of the listed attribute have a value, authentication fails.

5 List of attributes to use as the email address. First non-empty attribute is used.

6 List of attributes to use as the display name. First non-empty attribute is used.

7 List of attributes to use as the preferred user name when provisioning a user for this identity. First non-empty attribute is used.

8 Optional DN to use to bind during the search phase.

9 Optional password to use to bind during the search phase.

10 Certificate bundle to use to validate server certificates for the configured URL. If empty, system trusted roots are used. Only applies if **insecure: false**.

11 When **true**, no TLS connection is made to the server. When **false**, **`ldaps://`** URLs connect using TLS, and **`ldap://`** URLs are upgraded to TLS.

12 An RFC 2255 URL which specifies the LDAP host and search parameters to use, as described above.

6.2.5. Basic Authentication (Remote)

Set **BasicAuthPasswordIdentityProvider** in the **`identityProviders`** stanza to validate user names and passwords against a remote server using a server-to-server Basic authentication request. User names and passwords are validated against a remote URL that is protected by Basic authentication and returns JSON.

A **`401`** response indicates failed authentication.

A non-**`200`** status, or the presence of a non-empty "error" key, indicates an error:

```
{"error":"Error message"}
```

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

```
{"sub":"userid"} 1
```

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

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

- A display name using the **`name`** key. For example:

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

- An email address using the **`email`** key. For example:

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

- A preferred user name using the **`preferred_username`** key. This is useful when the unique, unchangeable subject is a database key or UID, and a more human-readable name exists. This is used as a hint when provisioning the OpenShift user for the authenticated identity. For example:

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

Example 6.5. Master Configuration Using BasicAuthPasswordIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_remote_basic_auth_provider 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://www.example.com/remote-idp 4
      ca: /path/to/ca.file 5
      certFile: /path/to/client.crt 6
      keyFile: /path/to/client.key 7
```

1 This provider name is prefixed to the returned user ID to form an identity name.

2 When **true**, unauthenticated token requests from non-web clients (like the CLI) are sent a **`WWW-Authenticate`** challenge header for this provider.

3 When **true**, unauthenticated token requests from web clients (like the web console) are redirected to a login page backed by this provider.

4 URL accepting credentials in Basic authentication headers.

5 Optional: Certificate bundle to use to validate server certificates for the configured URL.

6 Optional: Client certificate to present when making requests to the configured URL.

7 Key for the client certificate. Required if **`certFile`** is specified.

6.2.6. Request Header

Set **RequestHeaderIdentityProvider** in the **`identityProviders`** stanza to identify users from request header values, such as **`X-Remote-User`**. It is typically used in combination with an authenticating proxy, which sets the request header value. This is similar to how the remote user plug-in in OpenShift Enterprise 2 allowed administrators to provide Kerberos, LDAP, and many other forms of enterprise authentication.

For users to authenticate using this identity provider, they must access *<master>/oauth/authorize* via an authenticating proxy. You can either proxy the entire master API server so that all access goes through the proxy, or you can configure the OAuth server to redirect unauthenticated requests to the proxy.

To redirect unauthenticated requests from clients expecting login flows:

1. Set the **`login`** parameter to **true**.

2. Set the **`provider.loginURL`** parameter to the proxy URL to send those clients to.

To redirect unauthenticated requests from clients expecting **`WWW-Authenticate`** challenges:

1. Set the **`challenge`** parameter to **true**.

2. Set the **provider.challengeURL** parameter to the proxy URL to send those clients to.

The **provider.challengeURL** and **provider.loginURL** parameters can include the following tokens in the query portion of the URL:

- **${url}** is replaced with the current URL, escaped to be safe in a query parameter.
  For example: https://www.example.com/sso-login?then=${url}

- **${query}** is replaced with the current query string, unescaped.
  For example: https://www.example.com/auth-proxy/oauth/authorize?${query}

> **WARNING**
>
> If you expect unauthenticated requests to reach the OAuth server, a **clientCA** parameter should be set for this identity provider, so that incoming requests are checked for a valid client certificate before the request's headers are checked for a user name. Otherwise, any direct request to the OAuth server can impersonate any identity from this provider, merely by setting a request header.

Example 6.6. Master Configuration Using RequestHeaderIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_request_header_provider 1
    challenge: true 2
    login: true 3
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://www.example.com/challenging-
proxy/oauth/authorize?${query}" 4
      loginURL: "https://www.example.com/login-proxy/oauth/authorize?
${query}" 5
      clientCA: /path/to/client-ca.file 6
      headers: 7
      - X-Remote-User
      - SSO-User
```

1 This provider name is prefixed to the user name in the request header to form an identity name.

2 **RequestHeaderIdentityProvider** can only respond to clients that request **WWW-Authenticate** challenges by redirecting to a configured **challengeURL**. The configured URL should respond with a **WWW-Authenticate** challenge.

3 **RequestHeaderIdentityProvider** can only respond to clients requesting a login flow by redirecting to a configured **loginURL**. The configured URL should respond with a login flow.

4

Optional: URL to redirect unauthenticated */oauth/authorize* requests to, for clients which expect interactive logins. *${url}* is replaced with the current URL, escaped to be safe in a query

5 Optional: URL to redirect unauthenticated */oauth/authorize* requests to, for clients which expect **`WWW-Authenticate`** challenges. *${url}* is replaced with the current URL, escaped to be safe in a query parameter. *${query}* is replaced with the current query string.

6 Optional: PEM-encoded certificate bundle. If set, a valid client certificate must be presented and validated against the certificate authorities in the specified file before the request headers are checked for user names.

7 Header names to check, in order, for user names. The first header containing a value is used as the user name. Required, case-insensitive.

Example 6.7. Apache Authentication Using RequestHeaderIdentityProvider

This example configures an authentication proxy on the same host as the master. Having the proxy and master on the same host is merely a convenience and may not be suitable for your environment. For example, if you were already running a router on the master, port 443 would not be available.

It is also important to note that while this reference configuration uses Apache's **mod_auth_form**, it is by no means required and other proxies can easily be used if the following requirements are met:

1. Block the **`X-Remote-User`** header from client requests to prevent spoofing.
2. Enforce client certificate authentication in the **RequestHeaderIdentityProvider** configuration.
3. Require the **`X-Csrf-Token`** header be set for all authentication request using the challenge flow.
4. Only the */oauth/authorize* endpoint should be proxied, and redirects should not be rewritten to allow the backend server to send the client to the correct location.

Installing the Prerequisites

The **mod_auth_form** module is shipped as part of the **mod_session** package that is found in the Optional channel:

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl
```

Generate a CA for validating requests that submit the trusted header. This CA should be used as the file name for **`clientCA`** in the master's identity provider configuration.

```
# oadm ca create-signer-cert \
  --cert='/etc/openshift/master/proxyca.crt' \
  --key='/etc/openshift/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/openshift/master/proxyca.serial.txt'
```

Generate a client certificate for the proxy. This can be done using any x509 certificate tooling. For convenience, the **`oadm`** CLI can be used:

```
# oadm create-api-client-config \
  --certificate-authority='/etc/openshift/master/proxyca.crt' \
  --client-dir='/etc/openshift/master/proxy' \
  --signer-cert='/etc/openshift/master/proxyca.crt' \
  --signer-key='/etc/openshift/master/proxyca.key' \
  --signer-serial='/etc/openshift/master/proxyca.serial.txt' \
  --user='system:proxy' ❶

# pushd /etc/openshift/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt ❷
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

❶ The user name can be anything, however it is useful to give it a descriptive name as it will appear in logs.

❷ When running the authentication proxy on a different host name than the master, it is important to generate a certificate that matches the host name instead of using the default master certificate as shown above. The value for **`masterPublicURL`** in the */etc/openshift/master/master-config.yaml* file must be included in the **`X509v3 Subject Alternative Name`** in the certificate that is specified for **`SSLCertificateFile`**. If a new certificate needs to be created, the **`oadm ca create-server-cert`** command can be used.

Configuring Apache

Unlike OpenShift Enterprise 2, this proxy does not need to reside on the same host as the master. It uses a client certificate to connect to the master, which is configured to trust the **`X-Remote-User`** header.

Configure Apache per the following:

```
LoadModule auth_form_module modules/mod_auth_form.so
LoadModule session_module modules/mod_session.so
LoadModule request_module modules/mod_request.so

# Nothing needs to be served over HTTP.  This virtual host simply
redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www.adoc
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN and
X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
  ServerName www.example.com
  DocumentRoot /var/www.adoc
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It's critical to enforce client certificates on the Master.
Otherwise
  # requests could spoof the X-Remote-User header by accessing the
Master's
  # /oauth/authorize endpoint directly.
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

  # Send all requests to the console
  RewriteEngine              On
  RewriteRule     ^/console(.*)$     https://%
{HTTP_HOST}:8443/console$1 [R,L]

  # In order to using the challenging-proxy an X-Csrf-Token must be
present.
  RewriteCond %{REQUEST_URI} ^/challenging-proxy
  RewriteCond %{HTTP:X-Csrf-Token} ^$ [NC]
  RewriteRule ^.* - [F,L]

  <Location /challenging-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize
    AuthType basic
  </Location>

  <Location /login-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize

    # mod_auth_form providers are implemented by mod_authn_dbm,
mod_authn_file,
    # mod_authn_dbd, mod_authnz_ldap and mod_authn_socache.
    AuthFormProvider file
    AuthType form
    AuthName openshift
    ErrorDocument 401 /login.adoc
  </Location>

  <ProxyMatch /oauth/authorize>
    AuthUserFile /etc/openshift/htpasswd
    AuthName openshift
    Require valid-user
    RequestHeader set X-Remote-User %{REMOTE_USER}s

    # For ldap:
    # AuthBasicProvider ldap
    # AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-
domain,dc=com?uid?sub?(objectClass=*)"
```

```
    # It's possible to remove the mod_auth_form usage and replace it
with
    # something like mod_auth_kerb, mod_auth_gsspai or even
mod_auth_mellon.
    # The former would be able to support both the login and challenge
flows
    # from the Master.  Mellon would likely only support the login flow.

    # For Kerberos
    # yum install mod_auth_gssapi
    # AuthType GSSAPI
    # GssapiCredStore keytab:/etc/httpd.keytab
  </ProxyMatch>

</VirtualHost>

RequestHeader unset X-Remote-User
```

Additional mod_auth_form Requirements

A sample login page is available from the openshift_extras repository. This file should be placed in the **DocumentRoot** location (***/var/www.adoc*** by default).

Creating Users

At this point, you can create the users in the system Apache is using to store accounts information. In this example, file-backed authentication is used:

```
# yum -y install httpd-tools
# touch /etc/openshift/htpasswd
# htpasswd -c /etc/openshift/htpasswd <user_name>
```

Configuring the Master

The **identityProviders** stanza in the ***/etc/openshift/master/master-config.yaml*** file must be updated as well:

```
  identityProviders:
  - name: requestheader
    challenge: true
    login: true
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://[MASTER]/challenging-proxy/oauth/authorize?
${query}"
      loginURL: "https://[MASTER]/login-proxy/oauth/authorize?${query}"
      clientCA: /etc/openshift/master/proxyca.crt
      headers:
      - X-Remote-User
```

Restarting Services

Finally, restart the following services:

```
# systemctl restart httpd
# systemctl restart openshift-master
```

Verifying the Configuration

1. Test by bypassing the proxy. You should be able to request a token if you supply the correct client certificate and header:

   ```
   # curl -L -k -H "X-Remote-User: joe" \
      --cert /etc/pki/tls/certs/authproxy.pem \
      https://[MASTER]:8443/oauth/token/request
   ```

2. If you do not supply the client certificate, the request should be denied:

   ```
   # curl -L -k -H "X-Remote-User: joe" \
      https://[MASTER]:8443/oauth/token/request
   ```

3. This should show a redirect to the configured **`challengeURL`** (with additional query parameters):

   ```
   # curl -k -v -H 'X-Csrf-Token: 1' \
      '<masterPublicURL>/oauth/authorize?client_id=openshift-
   challenging-client&response_type=token'
   ```

4. This should show a 401 response with a **`WWW-Authenticate`** basic challenge:

   ```
   #  curl -k -v -H 'X-Csrf-Token: 1' \
       '<redirected challengeURL from step 3 +query>'
   ```

5. This should show a redirect with an access token:

   ```
   #  curl -k -v -u <your_user>:<your_password> \
       -H 'X-Csrf-Token: 1' '<redirected_challengeURL_from_step_3
   +query>'
   ```

6.2.7. GitHub

Set **GitHubIdentityProvider** in the **`identityProviders`** stanza to use GitHub as an identity provider, using the OAuth integration.

NOTE

Using GitHub as an identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Example 6.8. Master Configuration Using GitHubIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
```

```
  - name: github ①
    challenge: false ②
    login: true ③
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      clientID: ... ④
      clientSecret: ... ⑤
```

① This provider name is prefixed to the GitHub numeric user ID to form an identity name. It is also used to build the callback URL.

② **GitHubIdentityProvider** cannot be used to send **`WWW-Authenticate`** challenges.

③ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to GitHub to log in.

④ The client ID of a registered GitHub OAuth application. The application must be configured with a callback URL of **`<master>/oauth2callback/<identityProviderName>`**.

⑤ The client secret issued by GitHub.

6.2.8. Google

Set **GoogleIdentityProvider** in the **`identityProviders`** stanza to use Google as an identity provider, using Google's OpenID Connect integration.

NOTE

Using Google as an identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Example 6.9. Master Configuration Using GoogleIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: google ①
    challenge: false ②
    login: true ③
    provider:
      apiVersion: v1
      kind: GoogleIdentityProvider
      clientID: ... ④
      clientSecret: ... ⑤
      hostedDomain: "" ⑥
```

① This provider name is prefixed to the Google numeric user ID to form an identity name. It is also used to build the redirect URL.

2. **GoogleIdentityProvider** cannot be used to send **`WWW-Authenticate`** challenges.
3. When **true**, unauthenticated token requests from web clients (like the web console) are redirected to Google to log in.
4. The client ID of a registered Google project. The project must be configured with a redirect URI of **`<master>/oauth2callback/<identityProviderName>`**.
5. The client secret issued by Google.
6. Optional hosted domain to restrict sign-in accounts to. If empty, any Google account is allowed to authenticate.

6.2.9. OpenID Connect

Set **OpenIDIdentityProvider** in the **`identityProviders`** stanza to integrate with an OpenID Connect identity provider using an Authorization Code Flow.

NOTE

ID Token and **UserInfo** decryptions are not supported.

By default, the **openid** scope is requested. If required, extra scopes can be specified in the **`extraScopes`** field.

Claims are read from the JWT **`id_token`** returned from the OpenID identity provider and, if specified, from the JSON returned by the **`UserInfo`** URL.

At least one claim must be configured to use as the user's identity. The standard identity claim is **`sub`**.

You can also indicate which claims to use as the user's preferred user name, display name, and email address. If multiple claims are specified, the first one with a non-empty value is used. The standard claims are:

`sub`	The user identity.
`preferred_username`	The preferred user name when provisioning a user.
`email`	Email address.
`name`	Display name.

NOTE

Using an OpenID Connect identity provider requires users to get a token using **`<master>/oauth/token/request`** to use with command-line tools.

Example 6.10. Standard Master Configuration Using OpenIDIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect ❶
    challenge: false ❷
    login: true ❸
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ... ❹
      clientSecret: ... ❺
      claims:
        id:
        - sub ❻
        preferredUsername:
        - preferred_username
        name:
        - name
        email:
        - email
      urls:
        authorize: https://myidp.example.com/oauth2/authorize ❼
        token: https://myidp.example.com/oauth2/token ❽
```

❶ This provider name is prefixed to the value of the identity claim to form an identity name. It is also used to build the redirect URL.

❷ **OpenIDIdentityProvider** cannot be used to send **`WWW-Authenticate`** challenges.

❸ When **true**, unauthenticated token requests from web clients (like the web console) are redirected to the authorize URL to log in.

❹ The client ID of a client registered with the OpenID provider. The client must be allowed to redirect to **`<master>/oauth2callback/<identityProviderName>`**.

❺ The client secret.

❻ Use the value of the **`sub`** claim in the returned **`id_token`** as the user's identity.

❼ Authorization Endpoint described in the OpenID spec. Must use **`https`**.

❽ Token Endpoint described in the OpenID spec. Must use **`https`**.

A custom certificate bundle, extra scopes, extra authorization request parameters, and **`userInfo`** URL can also be specified:

Example 6.11. Full Master Configuration Using OpenIDIdentityProvider

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
```

```
    challenge: false
    login: true
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ...
      clientSecret: ...
      ca: my-openid-ca-bundle.crt 1
      extraScopes: 2
      - email
      - profile
      extraAuthorizeParameters: 3
        include_granted_scopes: "true"
      claims:
        id: 4
        - custom_id_claim
        - sub
        preferredUsername: 5
        - preferred_username
        - email
        name: 6
        - nickname
        - given_name
        - name
        email: 7
        - custom_email_claim
        - email
      urls:
        authorize: https://myidp.example.com/oauth2/authorize
        token: https://myidp.example.com/oauth2/token
        userInfo: https://myidp.example.com/oauth2/userinfo 8
```

1 Certificate bundle to use to validate server certificates for the configured URLs. If empty, system trusted roots are used.

2 Optional list of scopes to request, in addition to the **openid** scope, during the authorization token request.

3 Optional map of extra parameters to add to the authorization token request.

4 List of claims to use as the identity. First non-empty claim is used. At least one claim is required. If none of the listed claims have a value, authentication fails.

5 List of claims to use as the preferred user name when provisioning a user for this identity. First non-empty claim is used.

6 List of claims to use as the display name. First non-empty claim is used.

7 List of claims to use as the email address. First non-empty claim is used.

8 UserInfo Endpoint described in the OpenID spec. Must use **`https`**.

6.3. TOKEN OPTIONS

The OAuth server generates two kinds of tokens:

Access tokens	Longer-lived tokens that grant access to the API.
Authorize codes	Short-lived tokens whose only use is to be exchanged for an access token.

Use the **`tokenConfig`** stanza to set token options:

Example 6.12. Master Configuration Token Options

```
oauthConfig:
  ...
  tokenConfig:
    accessTokenMaxAgeSeconds: 86400 ①
    authorizeTokenMaxAgeSeconds: 300 ②
```

① Set **`accessTokenMaxAgeSeconds`** to control the lifetime of access tokens. The default lifetime is 24 hours.

② Set **`authorizeTokenMaxAgeSeconds`** to control the lifetime of authorize codes. The default lifetime is five minutes.

6.4. GRANT OPTIONS

To configure how the OAuth server responds to token requests for a client the user has not previously granted permission, set the **`method`** value in the **`grantConfig`** stanza. Valid values for **`method`** are:

`auto`	Auto-approve the grant and retry the request.
`prompt`	Prompt the user to approve or deny the grant.
`deny`	Auto-deny the grant and return a failure error to the client.

Example 6.13. Master Configuration Grant Options

```
oauthConfig:
  ...
  grantConfig:
    method: auto
```

6.5. SESSION OPTIONS

The OAuth server uses a signed and encrypted cookie-based session during login and redirect flows.

Use the **`sessionConfig`** stanza to set session options:

Example 6.14. Master Configuration Session Options

```
oauthConfig:
  ...
  sessionConfig:
    sessionMaxAgeSeconds: 300 ①
    sessionName: ssn ②
    sessionSecretsFile: "..." ③
```

① Controls the maximum age of a session; sessions auto-expire once a token request is complete. If auto-grant is not enabled, sessions must last as long as the user is expected to take to approve or reject a client authorization request.

② Name of the cookie used to store the session.

③ File name containing serialized **`SessionSecrets`** object. If empty, a random signing and encryption secret is generated at each server start.

If no **`sessionSecretsFile`** is specified, a random signing and encryption secret is generated at each start of the master server. This means that any logins in progress will have their sessions invalidated if the master is restarted. It also means that if multiple masters are configured, they will not be able to decode sessions generated by one of the other masters.

To specify the signing and encryption secret to use, specify a **`sessionSecretsFile`**. This allows you separate secret values from the configuration file and keep the configuration file distributable, for example for debugging purposes.

Multiple secrets can be specified in the **`sessionSecretsFile`** to enable rotation. New sessions are signed and encrypted using the first secret in the list. Existing sessions are decrypted and authenticated by each secret until one succeeds.

Example 6.15. Session Secret Configuration:

```
apiVersion: v1
kind: SessionSecrets
secrets: ①
- authentication: "..." ②
  encryption: "..." ③
- authentication: "..."
  encryption: "..."
...
```

① List of secrets used to authenticate and encrypt cookie sessions. At least one secret must be specified. Each secret must set an authentication and encryption secret.

② Signing secret, used to authenticate sessions using HMAC. Recommended to use a secret with 32 or 64 bytes.

3 Encrypting secret, used to encrypt sessions. Must be 16, 24, or 32 characters long, to select AES-128, AES-192, or AES-256.

CHAPTER 7. SERVICE ACCOUNTS

7.1. OVERVIEW

When a person uses the command line or web console, their API token authenticates them to the OpenShift 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.

7.2. USERNAMES AND GROUPS

Every service account has an associated username that can be granted roles, just like a regular user. The username 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
```

Every service account is also a member of two groups:

- **system:serviceaccounts**, which includes all service accounts in the system
- **system:serviceaccounts:<project>**, which 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:serviceaccounts -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:serviceaccounts:managers -n top-
secret
```

7.3. ENABLE 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 master configuration**serviceAccountConfig** stanza to specify a **privateKeyFile** (for signing), and a matching public key file in the **publicKeyFiles** list:

```
serviceAccountConfig:
  ...
  masterCA: ca.crt ①
  privateKeyFile: serviceaccounts.private.key ②
  publicKeyFiles:
  - serviceaccounts.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.

7.4. MANAGED SERVICE ACCOUNTS

Service accounts are required in each project to run builds, deployments, and other pods. The **managedNames** setting in the master configuration file controls which service accounts are automatically created in every project:

```
serviceAccountConfig:
  ...
  managedNames: ①
  - builder ②
  - deployer ③
  - default ④
  - ...
```

①②List of service accounts to automatically create in every project

③ 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 Docker registry.

④ 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.

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 Docker registry.

7.5. INFRASTRUCTURE SERVICE ACCOUNTS

Several infrastructure controllers run using service account credentials. The following service accounts are created in the OpenShift infrastructure namespace at server start, and given the following roles cluster-wide:

- The **replication-controller** service account is assigned the **system:replication-controller** role

- The **deployment-controller** service account is assigned the **system:deployment-controller** role
- The **build-controller** service account is 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 namespace where those service accounts are created, set the **`openshiftInfrastructureNamespace`** field in the master configuration file:

```
policyConfig:
  ...
  openshiftInfrastructureNamespace: openshift-infra
```

Set **`limitSecretReferences`** field in master configuration file 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 namespace.

```
serviceAccountConfig:
  ...
  limitSecretReferences: false
```

CHAPTER 8. MANAGING AUTHORIZATION POLICIES

8.1. OVERVIEW

You can use the CLI to view authorization policies and the administrator CLI to manage the roles and bindings within a policy.

8.2. VIEWING ROLES AND BINDINGS

Roles grant various levels of access in the system-wide cluster policy as well as project-scoped local policies. 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 role in the cluster policy can view cluster policy and all local policies. Users with the **admin**default role in a given local policy can view that project-scoped policy.

8.2.1. Viewing Cluster Policy

To view the cluster roles and their associated rule sets in the cluster policy:

```
$ oc describe clusterPolicy default
```

Example 8.1. Viewing Cluster Roles

```
$ oc describe clusterPolicy default
Name:    default
Created:   4 hours ago
Labels:    <none>
Last Modified:   2015-06-10 17:22:25 +0000 UTC
admin    Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [create delete get list update watch] [pods/proxy projects
resourcegroup:exposedkube resourcegroup:exposedopenshift
resourcegroup:granter secrets]    [][]
    [get list watch]   [pods/exec pods/portforward resourcegroup:allkube
resourcegroup:allkube-status resourcegroup:allopenshift-status
resourcegroup:policy] [][]
    [get update]    [imagestreams/layers]               [][]
basic-user   Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [get]     [users]                 [~][]
    [list]     [projectrequests]               [][]
    [get list]    [clusterroles]                [][]
    [list]     [projects]                [][]
    [create]    [subjectaccessreviews]               [][]
IsPersonalSubjectAccessReview
cluster-admin   Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [*]     [*]                 [][]
    [*]     []                 [][*]
cluster-reader   Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get list watch]   [*]                 [][]
    [get]     []                 [][*]
cluster-status   Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get]     []                 [][/api /healthz /healthz/* /osapi
/version]
edit    Verbs     Resources                Resource Names Non-Resource
URLs    Extension
    [create delete get list update watch] [pods/proxy
resourcegroup:exposedkube resourcegroup:exposedopenshift secrets]
[][]
    [get list watch]   [pods/exec pods/portforward projects
resourcegroup:allkube resourcegroup:allkube-status
resourcegroup:allopenshift-status]  [][]
self-provisioner  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [create]    [projectrequests]               [][]
system:build-controller  Verbs     Resources                Resource
Names Non-Resource URLs    Extension
    [get list watch]   [builds]                [][]
    [update]    [builds]                [][]
    [get]     [imagestreams]                [][]
    [create delete get list]  [pods]                 [][]
    [create update]    [events]                [][]
system:component  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [*]     [*]                 [][]
system:deployer   Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get list]    [replicationcontrollers]              [][]
    [get update]    [replicationcontrollers]              [][]
    [create get list watch]   [pods]                 [][]
system:deployment-controller Verbs     Resources
Resource Names Non-Resource URLs    Extension
    [list watch]    [replicationcontrollers]              [][]
    [get update]    [replicationcontrollers]              [][]
    [create delete get list update]  [pods]                 [][]
    [create update]    [events]                [][]
system:image-builder  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get update]    [imagestreams/layers]               [][]
system:image-pruner  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [delete]    [images]                [][]
    [get list]    [buildconfigs builds deploymentconfigs images
imagestreams pods replicationcontrollers]       [][]
    [update]    [imagestreams/status]               [][]
system:image-puller  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get]     [imagestreams/layers]               [][]
system:node   Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [get list watch]   [services]                [][]
    [create get list watch]   [nodes]                 [][]
    [update]    [nodes/status]                [][]
    [create update]    [events]                [][]
    [get list watch]   [pods]                 [][]
    [create delete get]   [pods]                 [][]
    [update]    [pods/status]              [][]
    [get]     [secrets]                [][]
    [get]     [persistentvolumeclaims persistentvolumes]            []
[]
    [get]     [endpoints]               [][]
system:node-proxier  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [list watch]    [endpoints services]               [][]
system:oauth-token-deleter Verbs     Resources                Resource
Names Non-Resource URLs    Extension
    [delete]    [oauthaccesstokens oauthauthorizetokens]            []
[]
system:registry   Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [delete get]    [images]                [][]
    [get]     [imagestreamimages imagestreams imagestreamtags]
[][]
    [update]    [imagestreams]               [][]
    [create]    [imagestreammappings]               [][]
system:replication-controller Verbs     Resources
Resource Names Non-Resource URLs    Extension
    [list watch]    [replicationcontrollers]              [][]
    [get update]    [replicationcontrollers]              [][]
    [list watch]    [pods]                [][]
    [create delete]    [pods]                 [][]
    [create update]    [events]                [][]
system:router   Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [list watch]    [endpoints routes]               [][]
system:sdn-manager  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [create delete get list watch]  [hostsubnets]                [][]
    [get list watch]   [nodes]                 [][]
    [create get]    [clusternetworks]               [][]
system:sdn-reader  Verbs     Resources                Resource Names
Non-Resource URLs    Extension
    [get list watch]   [hostsubnets]                [][]
    [get list watch]   [nodes]                 [][]
    [get]     [clusternetworks]              [][]
system:webhook   Verbs     Resources                Resource Names Non-
Resource URLs    Extension
    [create get]    [buildconfigs/webhooks]               [][]
view    Verbs     Resources                Resource Names Non-Resource
URLs    Extension
    [get list watch]   [projects resourcegroup:allkube
resourcegroup:allkube-status resourcegroup:allopenshift-status
resourcegroup:exposedopenshift]  [][]
```

To view the current set of cluster bindings, which shows the users and groups that are bound to various roles:

```
$ oc describe clusterPolicyBindings :default
```

Example 8.2. Viewing Cluster Bindings

```
$ oc describe clusterPolicyBindings :default
Name:      :default
Created:     4 hours ago
Labels:      <none>
Last Modified:     2015-06-10 17:22:26 +0000 UTC
Policy:      <none>
RoleBinding[basic-users]:
      Role: basic-user
      Users: []
      Groups: [system:authenticated]
RoleBinding[cluster-admins]:
      Role: cluster-admin
      Users: []
      Groups: [system:cluster-admins]
RoleBinding[cluster-readers]:
      Role: cluster-reader
      Users: []
      Groups: [system:cluster-readers]
RoleBinding[cluster-status-binding]:
      Role: cluster-status
      Users: []
      Groups: [system:authenticated system:unauthenticated]
RoleBinding[self-provisioners]:
      Role: self-provisioner
      Users: []
      Groups: [system:authenticated]
RoleBinding[system:build-controller]:
      Role: system:build-controller
      Users: [system:serviceaccount:openshift-infra:build-controller]
      Groups: []
RoleBinding[system:deployment-controller]:
      Role: system:deployment-controller
      Users: [system:serviceaccount:openshift-infra:deployment-
controller]
      Groups: []
RoleBinding[system:masters]:
      Role: system:master
      Users: []
      Groups: [system:masters]
RoleBinding[system:node-proxiers]:
      Role: system:node-proxier
      Users: []
      Groups: [system:nodes]
RoleBinding[system:nodes]:
      Role: system:node
      Users: []
      Groups: [system:nodes]
RoleBinding[system:oauth-token-deleters]:
      Role: system:oauth-token-deleter
      Users: []
      Groups: [system:authenticated system:unauthenticated]
RoleBinding[system:registrys]:
      Role: system:registry
      Users: []
      Groups: [system:registries]
RoleBinding[system:replication-controller]:
      Role: system:replication-controller
      Users: [system:serviceaccount:openshift-infra:replication-
controller]
      Groups: []
RoleBinding[system:routers]:
      Role: system:router
      Users: []
      Groups: [system:routers]
RoleBinding[system:sdn-readers]:
      Role: system:sdn-reader
      Users: []
      Groups: [system:nodes]
RoleBinding[system:webhooks]:
      Role: system:webhook
      Users: []
      Groups: [system:authenticated system:unauthenticated]
```

8.2.2. Viewing Local Policy

While the list of local roles and their associated rule sets are not viewable within a local policy, all of the default roles are still applicable and can be added to users or groups, other than the **cluster-admin** default role. The local bindings, however, are viewable.

To view the current set of local bindings, which shows the users and groups that are bound to various roles:

```
$ oc describe policyBindings :default
```

By default, the current project is used when viewing local policy. Alternatively, a project can be specified with the **-n** flag. This is useful for viewing the local policy of another project, if the user already has the **admin**default role in it.

Example 8.3. Viewing Local Bindings

```
$ oc describe policyBindings :default -n joe-project
Name:     :default
Created:    About a minute ago
Labels:     <none>
Last Modified:    2015-06-10 21:55:06 +0000 UTC
Policy:     <none>
RoleBinding[admins]:
     Role: admin
     Users: [joe]
     Groups: []
RoleBinding[system:deployers]:
     Role: system:deployer
     Users: [system:serviceaccount:joe-project:deployer]
     Groups: []
RoleBinding[system:image-builders]:
     Role: system:image-builder
     Users: [system:serviceaccount:joe-project:builder]
     Groups: []
```

```
RoleBinding[system:image-pullers]:
     Role: system:image-puller
     Users: []
     Groups: [system:serviceaccounts:joe-project]
```

By default in a local policy, only the binding for the **admin** role is immediately listed. However, if other default roles are added to users and groups within a local policy, they become listed as well.

8.3. 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 **`oadm policy`** commands.

When managing a user or group's associated roles for a local policy 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 8.1. Local Policy Operations

Command	Description
`$ oadm policy who-can <verb> <resource>`	Indicates which users can perform an action on a resource.
`$ oadm policy add-role-to-user <role> <username>`	Binds a given role to specified users in the current project.
`$ oadm policy remove-role-from-user <role> <username>`	Removes a given role from specified users in the current project.
`$ oadm policy remove-user <username>`	Removes specified users and all of their roles in the current project.
`$ oadm policy add-role-to-group <role> <groupname>`	Binds a given role to specified groups in the current project.
`$ oadm policy remove-role-from-group <role> <groupname>`	Removes a given role from specified groups in the current project.
`$ oadm policy remove-group <groupname>`	Removes specified groups and all of their roles in the current project.

You can also manage role bindings for the cluster policy using the following operations. The **`-n`** flag is not used used for these operations because the cluster policy uses non-namespaced resources.

Table 8.2. Cluster Policy Operations

Command	Description
`$ oadm policy add-cluster-role-to-user <role> <username>`	Binds a given role to specified users for all projects in the cluster.
`$ oadm policy remove-cluster-role-from-user <role> <username>`	Removes a given role from specified users for all projects in the cluster.
`$ oadm policy add-cluster-role-to-group <role> <groupname>`	Binds a given role to specified groups for all projects in the cluster.
`$ oadm policy remove-cluster-role-from-group <role> <groupname>`	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:

```
$ oadm policy add-role-to-user admin alice -n joe-project
```

You can then view the local bindings and verify the addition in the output:

```
$ oc describe policyBindings :default -n joe-project
Name:     :default
Created:    5 minutes ago
Labels:     <none>
Last Modified:    2015-06-10 22:00:44 +0000 UTC
Policy:     <none>
RoleBinding[admins]:
     Role: admin
     Users: [alice joe] 1
     Groups: []
RoleBinding[system:deployers]:
     Role: system:deployer
     Users: [system:serviceaccount:joe-project:deployer]
     Groups: []
RoleBinding[system:image-builders]:
     Role: system:image-builder
     Users: [system:serviceaccount:joe-project:builder]
     Groups: []
RoleBinding[system:image-pullers]:
     Role: system:image-puller
     Users: []
     Groups: [system:serviceaccounts:joe-project]
```

1 The **alice** user has been added to the **admins `RoleBinding`**.

CHAPTER 9. MANAGING SECURITY CONTEXT CONSTRAINTS

9.1. OVERVIEW

Security context constraints allow administrators to control permissions for pods. To learn more about this API type please refer to the security context constraints (SCCs) architecture documentation. You may manage SCCs in your instance as normal API objects using the CLI.

NOTE

You must have **cluster-admin** privileges to manage SCCs.

9.2. LISTING SECURITY CONTEXT CONSTRAINTS

To get a current list of SCCs:

```
$ oc get scc
NAME         PRIV      CAPS      HOSTDIR   SELINUX     RUNASUSER
privileged   true      []        true      RunAsAny    RunAsAny
restricted   false     []        false     MustRunAs   MustRunAsRange
```

9.3. EXAMINING A SECURITY CONTEXT CONSTRAINTS OBJECT

To examine a particular SCC, use **`oc get`**, **`oc describe`**, **`oc export`**, or **`oc edit`**.

```
$ oc edit scc restricted
allowHostDirVolumePlugin: false
allowHostNetwork: false
allowHostPorts: false
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: v1
groups:
- system:authenticated
kind: SecurityContextConstraints
metadata:
  creationTimestamp: 2015-09-08T07:37:54Z
  name: restricted 1
  resourceVersion: "58"
  selfxref: /api/v1/securitycontextconstraints/restricted
  uid: 849d9228-55fc-11e5-976b-080027c5bfa9
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
```

1 The SCC name specified in the **`oc edit`** command.

9.4. CREATING NEW SECURITY CONTEXT CONSTRAINTS

To create a new SCC, first define the SCC in a JSON or YAML file:

Example 9.1. Security Context Constraint Object Definition

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: scc-admin
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
users:
- my-admin-user
groups:
- my-admin-group
```

Although this example definition was written by hand, another way is to modify the definition obtained from examining a particular SCC.

Then, run **`oc create`** passing the file to create it:

```
$ oc create -f scc_admin.yaml
securitycontextconstraints/scc-admin

$ oc get scc
NAME         PRIV      CAPS      HOSTDIR   SELINUX     RUNASUSER
privileged   true      []        true      RunAsAny    RunAsAny
restricted   false     []        false     MustRunAs   MustRunAsRange
scc-admin    true      []        false     RunAsAny    RunAsAny
```

9.5. DELETING SECURITY CONTEXT CONSTRAINTS

To delete an SCC:

```
$ oc delete scc <scc_name>
```

NOTE

If you delete the default SCCs, they will not be regenerated upon restart, unless you delete all SCCs. If any constraint already exists within the system, no regeneration will take place.

9.6. UPDATING SECURITY CONTEXT CONSTRAINTS

To update an existing SCC:

```
$ oc edit scc <scc_name>
```

9.7. UPDATING THE DEFAULT SECURITY CONTEXT CONSTRAINTS

If you would like to reset your security context constraints to the default settings for any reason you may delete the existing security context constraints and restart your master. The default security context constraints will only be recreated if no security context constraints exist in the system.

9.8. HOW DO I?

The following describe common scenarios and procedures using SCCs.

9.8.1. 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.

2. Run:

   ```
   $ oc edit scc <name>
   ```

3. Add the user or group to the **users** or **groups** field of the SCC.

For example, to allow the **e2e-user** access to the **privileged** SCC, add their user:

```
$ oc edit scc privileged

allowHostDirVolumePlugin: true
allowPrivilegedContainer: true
apiVersion: v1
groups:
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  creationTimestamp: 2015-06-15T20:44:53Z
  name: privileged
  resourceVersion: "58"
  selfxref: /api/v1/securitycontextconstraints/privileged
  uid: 602a0838-139f-11e5-8aa4-080027c5bfa9
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
users:
- system:serviceaccount:openshift-infra:build-controller
- e2e-user 1
```

1 The **e2e-user** added to the users section.

9.8.2. Grant a Service Account Access to the Privileged SCC

First, create a service account. For example, to create service account `My_SVCACCT` in project `My_Project`:

```
$ cat <<EOF | oc create -n My_Project -f -
kind: ServiceAccount
apiVersion: v1
metadata:
  name: My_SVCACCT 1
EOF
```

Then, add the service account to the **`privileged`** SCC.

```
$ oc edit scc privileged
```

Add the following under **`users`**:

```
   - system:serviceaccount:My_Project:My_SVCACCT
```

9.8.3. 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. Edit the **restricted** SCC:

   ```
   $ oc edit scc restricted
   ```

2. Change the **`runAsUser.Type`** strategy to **RunAsAny**.

IMPORTANT

This allows images to run as the root UID if no **USER** is specified in the ***Dockerfile***.

9.8.4. 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 **`oadm 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.

9.8.5. 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 or edit the **privileged** SCC:

   ```
   $ oc edit scc <name>
   ```

2. Add the allowed capability using the **`allowedCapabilities`** field.

3. When creating the pod, request the capability in the **`securityContext.capabilities.add`** field.

9.8.6. Modify Cluster Default Behavior

To modify your cluster so that it does not pre-allocate UIDs, allows containers to run as any user, and prevents privileged containers:

1. Edit the **restricted** SCC:

   ```
    $ oc edit scc restricted
   ```

2. Change **`runAsUser.Type`** to **RunAsAny**.

3. Ensure **`allowPrivilegedContainer`** is set to false.

4. Save the changes.

To modify your cluster so that it does not pre-allocate UIDs and does not allow containers to run as root:

1. Edit the **restricted** SCC:

   ```
    $ oc edit scc restricted
   ```

2. Change **`runAsUser.Type`** to **MustRunAsNonRoot**.

3. Save the changes.

9.8.7. 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 the **privileged** SCC:

1. Edit the **restricted** SCC:

   ```
   $ oc edit scc restricted
   ```

2. Add **`allowHostDirVolumePlugin: true`**.

3. Save the changes.

CHAPTER 10. SCHEDULER

10.1. OVERVIEW

The Kubernetes 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.

10.2. GENERIC SCHEDULER

The existing generic scheduler is the default platform-provided scheduler "engine" that selects a node to host the pod in a 3-step operation:

1. Filter the nodes

2. Prioritize the filtered list of nodes

3. Select the best fit node

10.2.1. Filter the nodes

The available nodes are filtered based on the constraints or requirements specified. This is done by running each of the nodes through the list of filter functions called 'predicates'.

10.2.2. 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 is 1) and then combined by just adding the scores for each node provided by all the priority functions. This weight attribute can be used by administrators to give higher importance to some priority functions.

10.2.3. 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.

10.3. AVAILABLE PREDICATES

There are several predicates provided out of the box in Kubernetes. Some of these predicates can be customized by providing certain parameters. Multiple predicates can be combined to provide additional filtering of nodes.

10.3.1. 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.

PodFitsPorts deems a node to be fit for hosting a pod based on the absence of port conflicts.

```
{"name" : "PodFitsPorts"}
```

- ***PodFitsResources*** determines a fit based on resource availability. 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"}
```

NoDiskConflict determines fit based on non-conflicting disk volumes. It evaluates if a pod can fit due to the volumes it requests, and those that are already mounted. It is GCE and Amazon EBS specific.

```
{"name" : "NoDiskConflict"}
```

MatchNodeSelector determines fit based on node selector query that is defined in the pod.

```
{"name" : "MatchNodeSelector"}
```

HostName determines fit based on the presence of the Host parameter and a string match with the name of the host.

```
{"name" : "HostName"}
```

10.3.2. Configurable Predicates

These predicates can be configured by the user to tweak their functioning. They can be given any user-defined name. The type of the predicate is identified by the argument that they take. 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.

ServiceAffinity filters out nodes that do not belong to the specified topological level defined by the provided labels. This predicate takes in a list of labels and ensures affinity within the nodes (that have the same label values) for pods belonging to the same service. If the pod specifies a value for the labels in its NodeSelector, then the nodes matching those labels are the ones where the pod is scheduled. If the pod does not specify the labels in its NodeSelector, then the first pod can be placed on any node based on availability and all subsequent pods of the service will be scheduled on nodes that have the same label values.

```
{"name" : "Zone", "argument" : {"serviceAffinity" : {"labels" :
["zone"]}}}
```

LabelsPresence checks whether a particular node has a certain label defined or not, regardless of its value. Matching by label can be useful, for example, where nodes have their physical location or status defined by labels.

```
{"name" : "RequireRegion", "argument" : {"labelsPresence" : {"labels" :
["region"], "presence" : true}}}
```

- If "presence" is false, and any of the requested labels match any of the nodes's labels, it returns false. Otherwise, it returns true.

- If "presence" is true, and any of the requested labels do not match any of the node's labels, it returns false. Otherwise, it returns true.

10.4. AVAILABLE PRIORITY FUNCTIONS

A custom set of priority functions can be specified to configure the scheduler. There are several priority functions provided out-of-the-box in Kubernetes. Some of these priority functions can be customized by providing certain parameters. Multiple priority functions can be combined and different weights can be given to each in order to impact the prioritization. A weight is required to be specified and cannot be 0 or negative.

10.4.1. Static Priority Functions

These priority functions do not take any configuration parameters or inputs from the user. These are specified in the scheduler configuration using their exact name as well as the weight.

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}
```

ServiceSpreadingPriority spreads pods by minimizing the number of pods belonging to the same service onto the same machine.

```
{"name" : "ServiceSpreadingPriority", "weight" : 1}
```

EqualPriority gives an equal weight of one to all nodes, if no priority configs are provided. It is not required/recommended outside of testing.

```
{"name" : "EqualPriority", "weight" : 1}
```

10.4.2. Configurable Priority Functions

These priority functions can be configured by the user by providing certain parameters. They can be given any user-defined name. The type of the priority function is identified by the argument that they take. Since these are configurable, multiple priority functions of the same type (but different configuration parameters) can be combined as long as their user-defined names are different.

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.

```
{"name" : "RackSpread", "weight" : 1, "argument" : {"serviceAntiAffinity"
: {"label" : "rack"}}}
```

LabelPreference prefers nodes that have a particular label defined or not, regardless of its value.

```
{"name" : "RackPreferred", "weight" : 1, "argument" : {"labelPreference" :
{"label" : "rack"}}}
```

10.5. SCHEDULER POLICY

The selection of the predicate and priority functions defines the policy for the scheduler. Administrators can provide a JSON file that specifies the predicates and priority functions to configure the scheduler. The path to the scheduler policy file can be specified in the master configuration file. In the absence of the scheduler policy file, the default configuration gets applied.

It is important to note that the predicates and priority functions defined in the scheduler configuration file will completely override the default scheduler policy. If any of the default predicates and priority functions are required, they have to be explicitly specified in the scheduler configuration file.

10.5.1. Default Scheduler Policy

The default scheduler policy includes the following predicates:

1. PodFitsPorts

2. PodFitsResources

3. NoDiskConflict

4. MatchNodeSelector

5. HostName

The default scheduler policy includes the following priority functions. Each of the priority function has a weight of '1' applied to it:

1. LeastRequestedPriority

2. BalancedResourceAllocation

3. ServiceSpreadingPriority

10.6. USE CASES

One of the important use cases for scheduling within OpenShift is to support flexible affinity and anti-affinity policies.

10.6.1. Infrastructure Topological Levels

Administrators can define multiple topological levels for their infrastructure (nodes). This is done by specifying labels on nodes (eg: 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 (eg. regions → zones → racks). Lastly, administrators can specify affinity and anti-affinity rules at each of these levels in any combination.

10.6.2. 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 will be 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 will not get scheduled.

10.6.3. 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 will be spread across nodes that belong to that level. This ensures that the application is well spread for high availability purposes. The scheduler will try to balance the service pods across all applicable nodes as evenly as possible.

10.7. 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" : "PodFitsPorts"},
  {"name" : "PodFitsResources"},
  {"name" : "NoDiskConflict"},
  {"name" : "MatchNodeSelector"},
  {"name" : "HostName"}
 ],
 "priorities" : [
  {"name" : "LeastRequestedPriority", "weight" : 1},
  {"name" : "BalancedResourceAllocation", "weight" : 1},
  {"name" : "ServiceSpreadingPriority", "weight" : 1}
 ]
}
```

IMPORTANT

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 priority functions listed above.

Three topological levels defined as 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"}}}
 ]
}
```

Three topological levels defined as 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"}}}
 ]
}
```

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}}}
 ]
}
```

Configuration example combining static and configurable predicates and priority functions:

```
{
 "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}
 ]
}
```

10.8. SCHEDULER EXTENSIBILITY

As is the case with almost everything else in Kubernetes/OpenShift, the scheduler is built using a plug-in model and the current implementation itself is a plug-in. There are two ways to extend the scheduler functionality:

- Enhancements
- Replacement

10.8.1. Enhancements

The scheduler functionality can be enhanced by adding new predicates and priority functions. They can either be contributed upstream or maintained separately. These predicates and priority functions would need to be registered with the scheduler factory and then specified in the scheduler policy file.

10.8.2. Replacement

Since the scheduler is a plug-in, it can be replaced in favor of an alternate implementation. 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.

CHAPTER 11. PRUNING OBJECTS

11.1. OVERVIEW

Over time, API objects created in OpenShift 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 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.

11.2. BASIC PRUNE OPERATIONS

The CLI groups prune operations under a common parent command.

```
$ oadm 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.

11.3. 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:

```
$ oadm prune deployments [<options>]
```

Table 11.1. 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**)

To see what a pruning operation would delete:

```
$ oadm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

To actually perform the prune operation:

```
$ oadm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

11.4. 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:

```
$ oadm prune builds [<options>]
```

Table 11.2. 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:

```
$ oadm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m
```

To actually perform the prune operation:

```
$ oadm prune builds --orphans --keep-complete=5 --keep-failed=1 \
    --keep-younger-than=60m --confirm
```

11.5. PRUNING IMAGES

In order to prune images that are no longer required by the system due to age and status, administrators may run the following command:

```
$ oadm 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 rolesystem:image-pruner or greater (for example, **cluster-admin**).

Table 11.3. Prune Images CLI Configuration Options

Option	Description
`--certificate-authority`	The path to a certificate authority file to use when communicating with the OpenShift-managed registries. Defaults to the certificate authority data from the current user's config file.
`--confirm`	Indicate that pruning should occur, instead of performing a dry-run.
`--keep-tag-revisions=<N>`	For each image stream, keep up to at most N image revisions per tag. (default `60m`)
`--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`)

OpenShift uses the following logic to determine which images and layers to prune:

- Remove any image "managed by OpenShift" (i.e., 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`.
- 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.

To see what a pruning operation would delete:

```
$ oadm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

To actually perform the prune operation:

```
$ oadm prune images --keep-tag-revisions=3 --keep-younger-than=60m --
confirm
```

CHAPTER 12. MONITORING ROUTERS

12.1. 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.

12.2. 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.conf*** file on the container.

12.3. VIEWING LOGS

To view a router log, run the **`oc log`** 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.

12.4. 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/containers/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.conf*** 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/containers/router/certs*** directory.

- Certificates that are used for connecting to backends for re-encrypt terminated routes are stored in the ***/var/lib/containers/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 13. HIGH AVAILABILITY

13.1. OVERVIEW

This topic describes how to set up highly-available services on your OpenShift cluster.

The Kubernetes replication controller ensures that the deployment requirements, in particular the number of replicas, are satisfied when the appropriate resources are available. When run with two or more replicas, the router can be resilient to failures, providing a highly-available service. Depending on how the router instances are discovered (via a service, DNS entry, or IP addresses), this could impose operational requirements to handle failure cases when one or more router instances are "unreachable".

For some IP-based traffic services, virtual IP addresses (VIPs) should always be serviced for as long as a single instance is available. This simplifies the operational overhead and handles failure cases gracefully.

IMPORTANT

Even though a service is highly available, performance can still be affected.

Use cases for high-availability include:

- I want my cluster to be assigned a resource set and I want the cluster to automatically manage those resources.

- I want my cluster to be assigned a set of VIPs that the cluster manages and migrates (with zero or minimal downtime) on failure conditions, and I should not be required to perform any manual interactions to update the upstream "discovery" sources (e.g., DNS). The cluster should service all the assigned VIPs when at least a single node is available, despite the current available resources not being sufficient to reach the desired state.

You can configure a highly-available router or network setup by running multiple instances of the pod and fronting them with a balancing tier. This can be something as simple as DNS round robin, or as complex as multiple load-balancing layers.

13.2. CONFIGURING IP FAILOVER

Using IP failover involves switching IP addresses to a redundant or stand-by set of nodes on failure conditions.

The **`oadm ipfailover`** command helps set up the VIP failover configuration. As an administrator, you can configure IP failover on an entire cluster, or on a subset of nodes, as defined by the labeled selector. If you are running in production, match the labeled selector with at least two nodes to ensure you have failover protection and provide a **`--replicas=<n>`** value that matches the number of nodes for the given labeled selector:

```
$ oadm ipfailover [<Ip_failover_config_name>] <options> --replicas=<n>
```

The **`oadm ipfailover`** command ensures that a failover pod runs on each of the nodes matching the constraints or label used. This pod uses VRRP (Virtual Router Redundancy Protocol) with Keepalived to ensure that the service on the watched port is available, and, if needed, Keepalived will automatically float the VIPs if the service is not available.

13.2.1. 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.

Option	Variable Name	Notes
`--virtual-ips`	`OPENSHIFT_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.)

13.2.2. Configuring a Highly-available Routing Service

The following steps describe how to set up a highly-available router environment with IP failover:

1. Label the nodes for the service. This step can be optional if you run the service on any of the nodes in your Kubernetes cluster and use VIPs that can float within those nodes. This process may already exist within a complex cluster, in that nodes may be filtered by any constraints or requirements specified (e.g., nodes with SSD drives, or higher CPU, memory, or disk requirements, etc.).
   The following example defines a label as router instances that are servicing traffic in the US west geography **ha-router=geo-us-west**

   ```
   $ oc label nodes openshift-node-{5,6,7,8,9} "ha-router=geo-us-west"
   ```

2. OpenShift's **ipfailover** internally uses **keepalived**, so ensure that multicast is enabled on the nodes labeled above and that the nodes can accept network traffic for 224.0.0.18 (the VRRP multicast IP address). Depending on your environment's multicast configuration, you may need to add an `iptables` rule to each of the above labeled nodes. If you do need to add the `iptables` rules, please also ensure that the rules persist after a system restart:

   ```
   $ for node in openshift-node-{5,6,7,8,9}; do   ssh $node <<EOF

   export interface=${interface:-"eth0"}
   echo "Check multicast enabled ... ";
   ifconfig $interface | grep -i MULTICAST

   echo "Check multicast groups ... "
   netstat -g -n | grep 224.0.0 | grep $interface

   echo "Optionally, add accept rule and persist it ... "
   ```

```
sudo /sbin/iptables -I INPUT -i $interface -d 224.0.0.18/32 -j
ACCEPT

echo "Please ensure the above rule is added on system restarts."

EOF
done;
```

3. Depending on your environment policies, you can either reuse the **router** service account created previously or create a new **ipfailover** service account.
   Ensure that either the **router** service account exists as described in Deploying a Router or create a new **ipfailover** service account. The example below creates a new service account with the name **ipfailover**:

```
$ echo '
  { "kind": "ServiceAccount",
    "apiVersion": "v1",
    "metadata": { "name": "ipfailover" }
  }
  ' | oc create -f -
```

4. You can manually edit the **privileged** SCC and add the **ipfailover** service account, or you can script editing the **privileged** SCC if you have `jq` installed.

   a. To manually edit the **privileged** SCC, run:

   ```
   $ oc edit scc privileged
   ```

   Then add the **ipfailover** service account in form **system:serviceaccount:<project>:<name>** to the `users` section:

   ```
   ...
   users:
   - system:serviceaccount:openshift-infra:build-controller
   - system:serviceaccount:default:router
   - system:serviceaccount:default:ipfailover
   ```

   b. Alternatively, to script editing **privileged** SCC if you have `jq` installed, run:

   ```
   $ oc get scc privileged -o json |
       jq '.users |= .+
   ["system:serviceaccount:default:ipfailover"]' |
         oc replace scc -f -
   ```

5. Start the router with at least two replicas on nodes matching the labels used in the first step. The following example runs three instances using the **ipfailover** service account:

```
$ oadm router ha-router-us-west --replicas=3  \
       --selector="ha-router=geo-us-west" --labels="ha-router=geo-
us-west" \
       --credentials="$KUBECONFIG" --service-account=ipfailover
```

NOTE

The above command runs fewer router replicas than available nodes, so that, in the chance of node failures, Kubernetes can still ensure three available instances until the number of available nodes labeled **ha-router=geo-us-west** is below three. Additionally, the router uses the host network as well as ports 80 and 443, so fewer number of replicas are running to ensure a higher Service Level Availability (SLA). If there are no constraints on the service being setup for failover, it is possible to target the service to run on one or more, or even all, of the labeled nodes.

6. Finally, configure the VIPs and failover for the nodes labeled with **ha-router=geo-us-west** in the first step. Ensure the number of replicas match the number of nodes and that they satisfy the label setup in the first step. The name of the **ipfailover** configuration (**ipf-ha-router-us-west** in the example below) should be different from the name of the router configuration (**ha-router-us-west**) as both the router and **ipfailover** create deployment configuration with those names. Specify the VIPs addresses and the port number that **ipfailover** should monitor on the desired instances:

```
$ oadm ipfailover ipf-ha-router-us-west --replicas=5  --watch-
port=80    \
     --selector="ha-router=geo-us-west" --virtual-ips="10.245.2.101-
105" \
     --credentials="$KUBECONFIG" --service-account=ipfailover --
create
```

13.2.3. Configuring a Highly-available Network Service

The following steps describe how to set up a highly-available IP-based network service with IP failover:

1. Label the nodes for the service. This step can be optional if you run the service on any of the nodes in your Kubernetes cluster and use VIPs that can float within those nodes. This process may already exist within a complex cluster, in that the nodes may be filtered by any constraints or requirements specified (e.g., nodes with SSD drives, or higher CPU, memory, or disk requirements, etc.).
   The following example labels a highly-available cache service that is listening on port 9736 as **ha-cache=geo**:

```
$ oc label nodes openshift-node-{6,3,7,9} "ha-cache=geo"
```

2. OpenShift's **ipfailover** internally uses **keepalived**, so ensure that multicast is enabled on the nodes labeled above and that the nodes can accept network traffic for 224.0.0.18 (the VRRP multicast IP address). Depending on your environment's multicast configuration, you may need to add an **`iptables`** rule to each of the above labeled nodes. If you do need to add the **`iptables`** rules, please also ensure that the rules persist after a system restart:

```
$ for node in openshift-node-{6,3,7,9}; do   ssh $node <<EOF
export interface=${interface:-"eth0"}
echo "Check multicast enabled ... ";
ifconfig $interface | grep -i MULTICAST

echo "Check multicast groups ... "
netstat -g -n | grep 224.0.0 | grep $interface
```

```
echo "Optionally, add accept rule and persist it ... "
sudo /sbin/iptables -I INPUT -i $interface -d 224.0.0.18/32 -j
ACCEPT

echo "Please ensure the above rule is added on system restarts."

EOF
done;
```

3. Create a new **ipfailover** service account:

```
$ echo '
  { "kind": "ServiceAccount",
    "apiVersion": "v1",
    "metadata": { "name": "ipfailover" }
  }
  ' | oc create -f -
```

4. You can manually edit the **privileged** SCC and add the **ipfailover** service account, or you can script editing the **privileged** SCC if you have `jq` installed.

   a. To manually edit the **privileged** SCC, run:

   ```
   $ oc edit scc privileged
   ```

   Then add the **ipfailover** service account in form **system:serviceaccount:<project>:<name>** to the `users` section:

   ```
   ...
   users:
   - system:serviceaccount:openshift-infra:build-controller
   - system:serviceaccount:default:router
   - system:serviceaccount:default:ipfailover
   ```

   b. Alternatively, to script editing **privileged** SCC if you have `jq` installed, run:

   ```
   $ oc get scc privileged -o json |
       jq '.users |= .+
   ["system:serviceaccount:default:ipfailover"]' |
         oc replace scc -f -
   ```

5. Run a **geo-cache** service with two or more replicas. An example configuration for running a **geo-cache** service is provided here.

   IMPORTANT

   Be sure to replace the **myimages/geo-cache** Docker image referenced in the file with your intended image. Also, change the number of replicas to the desired amount and ensure the label matches the one used in the first step.

   ```
   $ oc create -n <namespace> -f ./examples/geo-cache.json
   ```

6. Finally, configure the VIPs and failover for the nodes labeled with **ha-cache=geo** in the first step.

Ensure the number of replicas match the number of nodes and that they satisfy the label setup in first step. Specify the VIP addresses and the port number that **ipfailover** should monitor for the desired instances:

```
$ oadm ipfailover ipf-ha-geo-cache --replicas=4 --selector="ha-
cache=geo" \
        --virtual-ips=10.245.2.101-104 --watch-port=9736  \
 --credentials="$KUBECONFIG" --service-account=ipfailover --create
```

Using the above example, you can now use the VIPs 10.245.2.101 through 10.245.2.104 to send traffic to the geo-cache service. If a particular geo-cache instance is "unreachable", perhaps due to a node failure, Keepalived ensures that the VIPs automatically float amongst the group of nodes labeled "ha-cache=geo" and the service is still reachable via the virtual IP addresses.

CHAPTER 14. SELF-PROVISIONED PROJECTS

14.1. OVERVIEW

You can allow developers to create their own projects. An accessible endpoint exists that will provision a project according to a template. This endpoint is made accessible when a developer creates a new project.

14.2. TEMPLATE FOR NEW PROJECTS

The API server automatically provisions projects based on the template that is defined in 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:

```
$ oadm create-bootstrap-project-template -o yaml > template.yaml
```

2. Modify the template by adding objects or modifying existing objects, then load the template:

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

3. 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.
PROJECT_ADMIN_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.

14.3. DISABLING SELF-PROVISIONING

Deleting the **self-provisioners**cluster role binding will deny permissions for self-provisioning any new projects. When disabling self-provisioning, set the **`projectRequestMessage`** parameter in the ***master-config.yaml*** file instructing developers on how to request a new project. This parameter is a string that will be presented to the developer in the web console and command line when they attempt to self-provision a project. For example:

```
Contact your system administrator at projectname@example.com to request a
project.
```

or:

```
To request a new project, fill out the project request form located at
https://internal.example.com/openshift-project-request.
```

CHAPTER 15. PERSISTENT STORAGE USING NFS

15.1. OVERVIEW

You can provision your OpenShift cluster with persistent storage using NFS. Some familiarity with Kubernetes and NFS is assumed.

The Kubernetes persistent volume framework allows administrators to provision a cluster with persistent storage and gives users a way to request those resources without having any knowledge of the underlying infrastructure.

For a detailed example, see the guide for WordPress and MySQL using NFS.

IMPORTANT

High-availability of storage in the infrastructure is left to the underlying storage provider.

15.2. PROVISIONING

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift. All that is required for NFS is a distinct list of servers and paths and the `PersistentVolume` API.

Example 15.1. Persistent Volume Object Definition

```
{
  "apiVersion": "v1",
  "kind": "PersistentVolume",
  "metadata": {
    "name": "pv0001"
  },
  "spec": {
    "capacity": {
        "storage": "5Gi"
    },
    "accessModes": [ "ReadWriteOnce" ],
    "nfs": {
        "path": "/tmp",
        "server": "172.17.0.2"
    },
    "persistentVolumeReclaimPolicy": "Recycle"
  }
}
```

15.2.1. Enforcing Disk Quotas

Use disk partitions to enforce disk quotas and size constraints. Each partition can be its own export. Each export is one persistent volume. Kubernetes enforces unique names for persistent volumes, but the uniqueness of the NFS volume’s server and path is up to the administrator.

Enforcing quotas in this way allows the end user to request persistent storage by a specific amount (e.g, 10Gi) and be matched with a corresponding volume of equal or greater capacity.

15.2.2. Volume Security

Users request storage with a `PersistentVolumeClaim`. This claim only lives in the user's namespace and can only be referenced by a pod within that same namespace. Any attempt to access a persistent volume across a namespace causes the pod to fail.

Each NFS volume must be mountable by all nodes in the cluster.

15.3. RECLAIMING RESOURCES

NFS implements the Kubernetes **Recyclable** plug-in interface. Automatic processes handle reclamation tasks based on policies set on each persistent volume.

By default, persistent volumes are set to **Retain**. NFS volumes which are set to **Recycle** are scrubbed (i.e., `rm -rf` is run on the volume) after being released from their claim (i.e, after the user's `PersistentVolumeClaim` bound to the volume is deleted). Once recycled, the NFS volume can be bound to a new claim.

15.4. AUTOMATION

As discussed, clusters can be provisioned with persistent storage using NFS in the following way:

- Disk partitions can be used to enforce storage quotas.

- Security can be enforced by restricting volumes to the namespace that has a claim to them.

- Reclamation of discarded resources can be configured for each persistent volume.

They are many ways that you can use scripts to automate the above tasks. You can use an example Ansible playbook to help you get started.

15.5. SELINUX AND NFS EXPORT SETTINGS

By default, SELinux does not allow writing from a pod to a remote NFS server. The NFS volume mounts correctly, but is read-only.

To enable writing in SELinux on each node:

```
# setsebool -P virt_use_nfs 1
```

The `-P` option makes the bool persistent between reboots.

Additionally, in order to enable arbitrary container users to read and write the volume, each exported volume on the NFS server itself should conform to the following:

- Each export must be:

```
/<example_fs> *(rw,all_squash)
```

- Each export must be owned by **nfsnobody**:

```
chown -R nfsnobody:nfsnobody /<example_fs>
```

- Each export must have the following permissions:

```
chmod 777 /<example_fs>
```

IMPORTANT

The export definition above allows arbitrary network clients to mount this volume. Exports can be restricted to a range of IP addresses for hosts that will access the volume. See **`man exports`** for more information.

IMPORTANT

Starting in OpenShift Enterprise 3.1, the export values have changed. See the OpenShift Enterprise 3.1 documentation for instructions on ensuring proper security for NFS in 3.1.

CHAPTER 16. IPTABLES

16.1. OVERVIEW

This topic describes how administrators should work with iptables. openshift-sdn takes care of adding the necessary iptables rules to make it work. Kubernetes and Docker also manage iptables for port forwarding and services.

16.2. RESTARTING

Docker doesn’t monitor the iptables rules that it adds for exposing ports from containers and hence if iptables service is restarted, then these rules are lost. So, to safely restart iptables, it is recommended that the rules are saved and restored.

```
$ iptables-save > /path/to/iptables.bkp
$ systemctl restart iptables
$ iptables-restore < /path/to/iptables.bkp
```

CHAPTER 17. NATIVE CONTAINER ROUTING

17.1. OVERVIEW

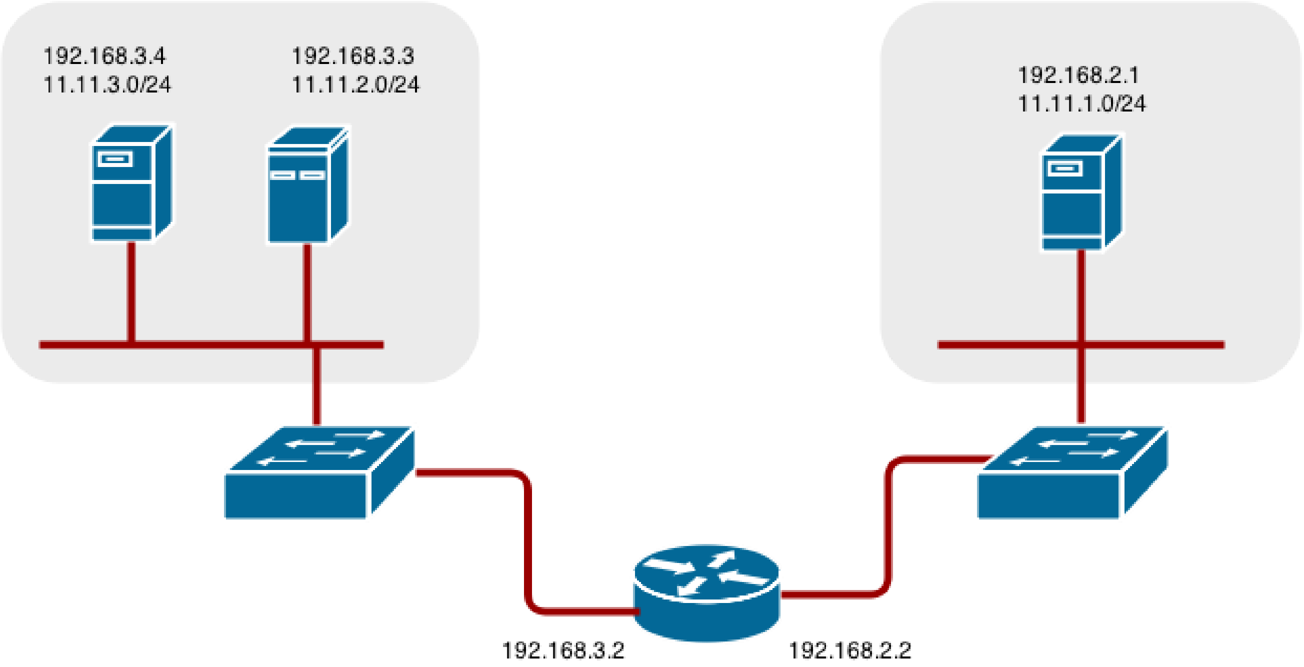
This topic describes how to set up container networking using existing switches and routers and the kernel networking stack in Linux. The setup requires that the network administrator or a script modifies the router or routers when new nodes are added to the cluster.

NOTE

The procedures outlined in this topic can be adapted to any type of router.

17.2. NETWORK LAYOUT

The following diagram shows the container networking setup described in this topic. It uses one Linux node with two network interface cards serving as a router, two switches, and three nodes connected to these switches.



17.3. NETWORK OVERVIEW

The following describes a general network setup:

- 11.11.0.0/16 is the container network.
- The 11.11.x.0/24 subnet is reserved for each node and assigned to the Docker Linux bridge.
- Each node has a route to the router for reaching anything in the 11.11.0.0/16 range, except the local subnet.
- The router has routes for each node, so it can be directed to the right node.
- Existing nodes do not need any changes when new nodes are added, unless the network topology is modified.

- IP forwarding is enabled on each node.

17.4. NODE SETUP

1. Assign an unused 11.11.x.0/24 subnet IP address to the Linux bridge on the node:

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

2. Modify the Docker startup script to use the new bridge. By default, the startup script is the **`/etc/sysconfig/docker`** file:

```
# docker -d -b lbr0 --other-options
```

3. Add a route to the router for the 11.11.0.0/16 network:

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

4. Enable IP forwarding on the node:

```
# sysctl -w net.ipv4.ip_forward=1
```

17.5. ROUTER SETUP

The following procedure assumes a Linux box with multiple NICs is used as a router. Modify the steps as required to use the syntax for a particular router:

1. Enable IP forwarding on the router:

```
# sysctl -w net.ipv4.ip_forward=1
```

2. Add a route for each node added to the cluster:

```
# ip route add <node_subnet> via <node_ip_address> dev <interface
through which node is L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

CHAPTER 18. SECURING BUILDS BY STRATEGY

18.1. OVERVIEW

Builds in OpenShift 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. 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, project administrators (the **admin** role) and project editors (the **edit** role) are granted permission to use all build strategies (Docker, Source-to-Image, and Custom).

You can control who can build with what build strategy using an authorization policy. Each build strategy has a corresponding build subresource. Granting permission to **create** on the build subresource allows the user to create builds of that type.

Table 18.1. Build Strategy Subresources

Strategy	Subresource
Docker	builds/docker
Source-to-Image	builds/source
Custom	builds/custom

18.2. DISABLING A BUILD STRATEGY GLOBALLY

To prevent access to a particular build strategy globally, log in as a user with **cluster-admin** privileges and edit each of the default 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:

Example 18.1. Disable the Docker Build Strategy for admin

```
kind: ClusterRole
metadata:
  name: admin
...

rules:
- attributeRestrictions: null
  resources:
  - builds/custom
  - builds/docker  1
  - builds/source
  - pods/exec
```

```
    - pods/portforward

 ...
```

1 1 Delete this line to disable Docker builds globally for users with the **admin** role.

18.3. RESTRICTING BUILD STRATEGIES TO A USER GLOBALLY

To allow only a set of specific users to create builds with a particular strategy:

1. Remove the build strategy subresource from the default **admin** and **edit** roles.

2. Create a separate role for the build strategy. For example, create a ***dockerstrategy.yaml*** file that defines a separate role for the Docker build strategy:

   ```
   kind: ClusterRole
   apiVersion: v1
   metadata:
     name: dockerbuilder
   rules:
   - resources:
     - builds/docker
     verbs:
     - create
   ```

   As cluster administrator, create the new cluster role:

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

3. Assign the new cluster role to a specific user. For example, to add the new **dockerbuilder** cluster role to the user **devuser**:

   ```
   $ oadm policy add-cluster-role-to-user dockerbuilder 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.

18.4. 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. Remove the build strategy resource from the default **admin** and **edit** roles.

2. Create a separate role for that build strategy.

3. Assign the role to a specific user within a project. For example, to add the new **dockerbuilder** role within the project **devproject** to the user **devuser**:

   ```
   $ oadm policy add-role-to-user dockerbuilder devuser -n devproject
   ```

CHAPTER 19. BUILDING DEPENDENCY TREES

19.1. OVERVIEW

OpenShift uses image change triggers in a build configuration to detect when an image stream tag has been updated. You can use the **`oadm 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.

19.2. USAGE

The following table describes common **`build-chain`** usage and general syntax:

Table 19.1. Common `build-chain` Operations

Description	Syntax
Build the dependency tree for the **latest** tag in **`<image-stream>`**.	`$ oadm build-chain <image-stream>`

Build the dependency tree for the **v2** tag in DOT format, and visualize it using the DOT utility.	`$ oadm build-chain <image-stream>:v2 \`
	`    -o dot \`
	`    | dot -T svg -o deps.svg`

Build the dependency tree across all projects for the specified image stream tag found the **test** project.	`$ oadm build-chain <image-stream>:v1 \`
	`    -n test --all`

NOTE

You may need to install the **graphviz** package to use the **`dot`** command.

CHAPTER 20. CUSTOMIZING THE WEB CONSOLE

20.1. OVERVIEW

Administrators can customize the web console using extensions, which let you run scripts and load custom stylesheets when the web console loads. You can change the look and feel of nearly any aspect of the user interface in this way.

20.2. LOADING CUSTOM SCRIPTS AND STYLESHEETS

To add scripts and stylesheets, edit the master configuration file. The scripts and stylesheet files must exist on the Asset Server and are added with the following options:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/script1.js
    - /path/to/script2.js
    - ...
  extensionStylesheets:
    - /path/to/stylesheet1.css
    - /path/to/stylesheet2.css
    - ...
```

Relative paths are resolved relative to the master configuration file. To pick up configuration changes, restart the server.

Custom scripts and stylesheets are read once at server start time. To make developing extensions easier, you can reload scripts and stylesheets on every request by enabling development mode with the following setting:

```
assetConfig:
  ...
  extensionDevelopment: true
```

When set, the web console reloads any changes to existing extension script or stylesheet files when you refresh the page in your browser. You still must restart the server when adding new extension stylesheets or scripts, however. This setting is only recommended for testing changes and not for production.

The following examples show common ways you can customize the web console.

Customizing the Logo

The following style changes the logo in the web console header:

```
#header-logo {
  background-image: url("https://www.example.com/images/logo.png");
  width: 160px;
  height: 10px;
}
```

Replace the **example.com** URL with a URL to an actual image, and adjust the width and height. The ideal height is **10px**.

Save the style to a file, for example ***logo.css***, and add it to the master configuration file:

```
assetConfig:
  ...
  extensionStylesheets:
    - /path/to/logo.css
```

Changing the Header Color

The following style changes the header color to dark blue:

```
.navbar-header {
  background-color: #2B3856;
}
```

Save the style to a file, for example ***theme.css***, and add it to the master configuration file:

```
assetConfig:
  ...
  extensionStylesheets:
    - /path/to/theme.css
```

Adding a Link to the Header

The following script adds a link into the web console header:

```
$(".navbar-utility").prepend('<li><a
href="http://example.com/status/">System Status</a></li>');
```

Save this script to a file, for example ***nav-link.js***, and add it to the master configuration file:

```
assetConfig:
  ...
  extensionScripts:
    - /path/to/nav-link.js
```

20.3. SERVING STATIC FILES

You can serve other files from the Asset Server as well. For example, you might want to make the CLI executable available for download from the web console or add images to use in a custom stylesheet.

Add the directory with the files you want using the following configuration option:

```
assetConfig:
  ...
  extensions:
    - name: images
      sourceDirectory: /path/to/my_images
```

The files under the ***/path/to/my_images*** directory will be available under the URL */<context>/extensions/images* in the web console.

To reference these files from a stylesheet, you should generally use a relative path. For example:

```
#header-logo {
  background-image: url("../extensions/images/my-logo.png");
}
```

20.3.1. Enabling.adoc5 Mode

The web console has a special mode for supporting certain static web applications that use the.adoc5 history API:

```
assetConfig:
  ...
  extensions:
    - name: my_extension
      sourceDirectory: /path/to/myExtension
     .adoc5Mode: true
```

Setting **`.adoc5Mode*`** to **true** enables two behaviors:

1. Any request for a non-existent file under ***/<context>/extensions/my_extension/*** instead serves ***/path/to/myExtension/index.adoc*** rather than a "404 Not Found" page.

2. The element **`<base href="/">`** will be rewritten in ***/path/to/myExtension/index.adoc*** to use the actual base depending on the asset configuration; only this exact string is rewritten.

This is needed for JavaScript frameworks such as AngularJS that require **`base`** to be set in ***index.adoc***.

20.4. CUSTOMIZING THE LOGIN PAGE

You can also change the login page for the web console. Run the following command to create a template you can modify:

```
$ oadm create-login-template > login-template.adoc
```

Edit the file to change the styles or add content, but be careful not to remove any required parameters inside curly braces.

To use your custom login page, set the following option in the master configuration file:

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.adoc
```

Relative paths are resolved relative to the master configuration file. You must restart the server after changing this configuration.

20.5. CHANGING THE LOGOUT URL

You can change the location a console user is sent to when logging out of the console by modifying the **`logoutURL`** parameter in the ***/etc/openshift/master/master-config.yaml*** file:

```
...
assetConfig:
  logoutURL: "http://www.example.com"
...
```

This can be useful when authenticating with Request Header and OAuth or OpenID identity providers, which require visiting an external URL to destroy single sign-on sessions.

CHAPTER 21. WORKING WITH HTTP PROXIES

21.1. OVERVIEW

Production environments can deny direct access to the Internet and instead have an HTTP or HTTPS proxy available. Configuring OpenShift to use these proxies can be as simple as setting standard environment variables in configuration or JSON files.

21.2. CONFIGURING HOSTS FOR PROXIES

1. Add the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables to each host's ***/etc/sysconfig/openshift-master*** or ***/etc/sysconfig/openshift-node*** file depending on the type of host:

   ```
   HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
   HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
   NO_PROXY=master.hostname.example.com
   ```

2. Restart the master or node host as appropriate:

   ```
   # systemctl restart openshift-master
   # systemctl restart openshift-node
   ```

21.3. PROXYING DOCKER PULL

OpenShift node hosts need to perform push and pull operations to Docker registries. If you have a registry that does not need a proxy for nodes to access, include the **NO_PROXY** parameter with the registry's host name, the registry service's IP address, and service name. This blacklists that registry, leaving the external HTTP proxy as the only option.

1. Edit the ***/etc/sysconfig/docker*** file and add the variables in shell format:

   ```
   HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
   HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
   NO_PROXY=master.hostname.example.com,172.30.123.45,docker-
   registry.default.svc.cluster.local
   ```

2. Restart the Docker service:

   ```
   # systemctl restart docker
   ```

21.4. CONFIGURING S2I BUILDS FOR PROXIES

S2I builds fetch dependencies from various locations. You can use a ***.sti/environment*** file to specify simple shell variables and OpenShift will react accordingly when seeing build images.

The following are the supported proxy environment variables with example values:

```
HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com
```

21.5. CONFIGURING DEFAULT TEMPLATES FOR PROXIES

The example templates available in OpenShift by default do not include settings for HTTP proxies. For existing applications based on these templates, modify the **source** section of the application's build configuration and add proxy settings:

```
...
source:
  type: Git
  git:
    uri: https://github.com/openshift/ruby-hello-world
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
...
```

This is similar to the process for using proxies for Git cloning.

21.6. SETTING PROXY ENVIRONMENT VARIABLES IN PODS

You can set the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables in the **templates.spec.containers** stanza in a deployment configuration to pass proxy connection information. The same can be done for configuring a Pod's proxy at runtime:

```
...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://USER:PASSWORD@IPADDR:PORT"
...
```

You can also use the **oc env** command to update an existing deployment configuration with a new environment variable:

```
$ oc env dc/frontend HTTP_PROXY=http://USER:PASSWORD@IPADDR:PORT
```

If you have a ConfigChange trigger set up in your OpenShift instance, the changes happen automatically. Otherwise, manually redeploy your application for the changes to take effect.

21.7. GIT REPOSITORY ACCESS

If your Git repository can only be accessed using a proxy, you can define the proxy to use in the **source** section of the **BuildConfig**. You can configure both a HTTP and HTTPS proxy to use. Both fields are optional.

NOTE

Your source URI must use the HTTP or HTTPS protocol for this to work.

```
...
source:
  type: Git
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
...
```

CHAPTER 22. REVISION HISTORY: ADMINISTRATION

22.1. THU MAY 19 2016

Affected Topic	Description of Change
Working with HTTP Proxies	Updated the example in the Configuring Default Templates for Proxies section to use **`https`** for GitHub access.

22.2. WED FEB 17 2016

Affected Topic	Description of Change
Persistent Storage Using NFS	Added an Important box explaining that the export values have changed for OpenShift Enterprise 3.1.

22.3. MON FEB 08 2016

Affected Topic	Description of Change
Aggregating Container Logs	Fixed RPM package paths for **`yum localinstall`** commands.

22.4. TUE JUN 23 2015

OpenShift Enterprise 3.0 release.