# OpenShift Container Platform 3.4 Cluster Administration

OpenShift Container Platform 3.4 Cluster Administration

Red Hat OpenShift Documentation Team

OpenShift Container Platform 3.4 Cluster Administration

## Legal Notice

## Abstract

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

# Table of Contents

# CHAPTER 1. OVERVIEW

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

# CHAPTER 2. MANAGING NODES

## 2.1. OVERVIEW

You can manage nodes in your instance using the CLI.

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

## 2.2. LISTING NODES

To list all nodes that are known to the master:

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

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

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

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

**Table 2.1. Node Conditions**

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

**Note**

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

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

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

For example:

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

## 2.3. ADDING NODES

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

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

## 2.4. DELETING NODES

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

To delete a node from the OpenShift Container Platform cluster:

1. Evacuate pods from the node you are preparing to delete.

2. Delete the node object:

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

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

   ```
   $ oc get nodes
   ```

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

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

## 2.5. 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
```

## 2.6. 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]
```

## 2.7. 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.

## 2.8. 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.

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.

## 2.9. REBOOTING NODES

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

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

### 2.9.1. Infrastructure Nodes

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

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

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

### 2.9.2. Using Pod Anti-Affinity for the Docker Registry Pod

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

Using the **`docker-registry`** pod as an example, the first step in enabling this feature is to set the **`scheduler.alpha.kubernetes.io/affinity`** on the pod. Since this pod uses a deployment configuration, the most appropriate place to add the annotation is to the pod template’s metadata.

```
$ oc edit dc/docker-registry -o yaml

...
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/affinity: |
          {
            "podAntiAffinity": {
              "requiredDuringSchedulingIgnoredDuringExecution": [{
                "labelSelector": {
                  "matchExpressions": [{
                    "key": "docker-registry",
                    "operator": "In",
                    "values":["default"]
                  }]
                },
                "topologyKey": "kubernetes.io/hostname"
              }]
            }
          }
```

**Important**

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

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

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

### 2.9.3. Handling Nodes Running Routers

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

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

## 2.10. CONFIGURING NODE RESOURCES

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

```
kubeletArguments:
  max-pods 1
    - "40"
  resolv-conf 2
    - "/etc/resolv.conf"
  image-gc-high-threshold: 3
    - "90"
  image-gc-low-threshold: 4
    - "80"
```

1

Number of pods that can run on this kubelet.

2

Resolver configuration file used as the basis for the container DNS resolution configuration.

3

The percent of disk usage after which image garbage collection is always run. Default: 90%

4

The percent of disk usage before which image garbage collection is never run. Lowest disk usage to garbage collect to. Default: 80%

To view all available kubelet options:

```
$ kubelet -h
```

This can also be set during an advanced installation using the **`openshift_node_kubelet_args`** variable. For example:

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

## 2.11. CHANGING NODE TRAFFIC INTERFACE

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

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

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

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

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

To change the node traffic interface:

1. Set the **`openshift_node_set_node_ip`** Ansible variable to **`true`**.

2. Set the **`openshift_ip`** to the IP address for the node you want to configure.

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

# CHAPTER 3. MANAGING USERS

## 3.1. OVERVIEW

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

## 3.2. ADDING A USER

After new users log in to OpenShift Container Platform, an account is created for that user per the identity provider configured on the master. The cluster administrator can manage the access level of each user.

## 3.3. VIEWING USER AND IDENTITY LISTS

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

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

To get the current list of users:

```
$ oc get user
NAME      UID                                     FULL NAME
IDENTITIES
demo     75e4b80c-dbf1-11e5-8dc6-0e81e52cc949
htpasswd_auth:demo
```

To get the current list of identities:

```
$ oc get identity
NAME                  IDP NAME        IDP USER NAME   USER NAME
USER UID
htpasswd_auth:demo    htpasswd_auth   demo            demo
75e4b80c-dbf1-11e5-8dc6-0e81e52cc949
```

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

## 3.4. MANAGING USER AND GROUP LABELS

To add a label to a user or group:

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

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

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

To remove the label:

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

To show labels for a user or group:

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

## 3.5. DELETING A USER

To delete a user:

1. Delete the user record:

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

2. Delete the user identity.

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

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

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

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

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

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

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

For external identification management like Lightweight Directory Access Protocol (LDAP) or Internet Download Manager (IDM), use the user management tools to remove the user entry.

# CHAPTER 4. MANAGING PROJECTS

## 4.1. OVERVIEW

In OpenShift Container Platform, projects are used to isolate content from groups of developers. As an administrator, you can give developers access to certain projects, allow them to create their own, and give them administrator rights.

## 4.2. SELF-PROVISIONING PROJECTS

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

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

3. Load the template:

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

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

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

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

| Parameter | Description |
|---|---|
| **PROJECT_NAME** | The name of the project. Required. |

| Parameter | Description |
|---|---|
| **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 administrating user. |
| **PROJECT_REQUESTING_USER** | The username of the requesting user. |

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

### 4.2.2. Disabling Self-provisioning

Removing the **`self-provisioners`**cluster role from authenticated user groups will deny permissions for self-provisioning any new projects.

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

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.
```

## 4.3. USING NODE SELECTORS

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

**Note**

Labels can be assigned during an advanced installation, or added to a node after installation.

### 4.3.1. Setting the Cluster-wide Default Node Selector

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

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

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

Restart the OpenShift service for the changes to take effect:

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

### 4.3.2. Setting the Project-wide Node Selector

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

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

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

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

**Note**

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

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

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

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

### 4.3.3. Developer-specified Node Selectors

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

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

## 4.4. LIMITING NUMBER OF SELF-PROVISIONED PROJECTS PER USER

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

**Important**

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

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

```
apiVersion: v1
kind: ProjectRequestLimitConfig
limits:
- selector:
    label1=value1
    label2=value2
  maxProjects: 10
- selector:
    label3=value3
  maxProjects: 5
```

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

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

```
admissionConfig:
  pluginConfig:
    ProjectRequestLimit:
      configuration:
        apiVersion: v1
        kind: ProjectRequestLimitConfig
        limits:
        - selector:
            level: admin 1
        - selector:
            level: advanced 2
          maxProjects: 10
        - maxProjects: 2 3
```

1 For selector **`level=admin`** no **`maxProjects`** is specified. This means that users with this label will not have a maximum of project requests

2 For selector **`level=advanced`** a maximum number of 10 projects will be allowed.

3 For the third entry, no selector is specified. This means that it will be applied to any user that doesn’t satisfy the previous two rules. Because rules are evaluated in order, this rule should be specified last.

**Note**

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

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

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

# CHAPTER 5. MANAGING PODS

## 5.1. OVERVIEW

This topic describes the management of pods, including managing their networks, limiting their run-once duration, and limiting what they can access, and how much bandwidth they can use.

## 5.2. MANAGING POD NETWORKS

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

### 5.2.1. Joining Project Networks

To join projects to an existing project network:

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

In the above example, all the pods and services in **`<project2>`** and **`<project3>`** can now access any pods and services in **`<project1>`** and vice versa.

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

## 5.3. ISOLATING PROJECT NETWORKS

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

```
$ oadm pod-network isolate-projects <project1> <project2>
```

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

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

### 5.3.1. Making Project Networks Global

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

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

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

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

## 5.4. LIMITING RUN-ONCE POD DURATION

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

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

### 5.4.1. Configuring the RunOnceDuration Plug-in

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

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      RunOnceDuration:
        configuration:
          apiVersion: v1
          kind: RunOnceDurationConfig
          activeDeadlineSecondsOverride: 3600 1
```

1

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

### 5.4.2. Specifying a Custom Duration per Project

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

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

1

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

## 5.5. CONTROLLING EGRESS TRAFFIC

As an OpenShift Container Platform cluster administrator, you can control egress traffic using either a router or firewall methods.

### 5.5.1. Limiting Pod Access with an Egress Router

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

#### 5.5.1.1. Important Deployment Considerations

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

**Red Hat OpenStack Platform**

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

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

**Red Hat Enterprise Virtualization**

If you are using Red Hat Enterprise Virtualization, you should set **`EnableMACAntiSpoofingFilterRules`** to **`false`**.

**VMware vSphere**

If you are using VMware vSphere, follow VMware's Securing Virtual Switch Ports and Forged Transmissions guidance.

#### 5.5.1.2. Deploying an Egress Router Pod

**Example 5.1. Example Pod Definition for an Egress Router**

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  containers:
  - name: egress-router
    image: openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 1
      value: 192.168.12.99
    - name: EGRESS_GATEWAY 2
      value: 192.168.12.1
    - name: EGRESS_DESTINATION 3
      value: 203.0.113.25
  nodeSelector:
    site: springfield-1 4
```

1

IP address on the node subnet reserved by the cluster administrator for use by this pod.

2

Same value as the default gateway used by the node itself.

3

Connections to the pod are redirected to 203.0.113.25, with a source IP address of 192.168.12.99

4

The pod will only be deployed to nodes with the label site **springfield-1**.

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

**Note**

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

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

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

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

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

### 5.5.1.3. Deploying an Egress Router Service

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

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

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

### 5.5.2. Limiting Pod Access with Egress Firewall

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

- A pod can only talk to internal hosts, and cannot initiate connections to the public Internet.

  Or,

- A pod can only talk to the public Internet, and cannot initiate connections to internal hosts (outside the cluster).

  Or,

- A pod cannot reach specified internal subnets/hosts that it should have no reason to contact.

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

**Caution**

You must have the **ovs-multitenant** plug-in enabled in order to limit pod access via egress policy.

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

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

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

### 5.5.2.1. Configuring Pod Access Limits

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

To configure pod access limits:

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

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

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

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

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

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

## 5.6. LIMITING THE BANDWIDTH AVAILABLE TO PODS

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

To limit the bandwidth on a pod:

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

   **Example 5.2. Limited Pod Object Definition**

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

2. Create the pod using the object definition:

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

## 5.7. SETTING POD DISRUPTION BUDGETS

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

**Note**

Starting in OpenShift Container Platform 3.4, pod disruption budgets is a feature in Technology Preview, available only for users with **cluster-admin** privileges.

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

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

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

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

```
apiVersion: policy/v1alpha1 (1)
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  selector:  (2)
    matchLabels:
      foo: bar
  minAvailable: 2  (3)
```

1

**PodDisruptionBudget** is part of the **policy/v1alpha** API group.

2

A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined.

3

The minimum number of pods that must be available simultaneously. This can be either an integer or a string specifying a percentage (for example, **20%**).

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

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

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

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

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

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

# CHAPTER 6. CONFIGURING SERVICE ACCOUNTS

## 6.1. OVERVIEW

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

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

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

## 6.2. USER NAMES AND GROUPS

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

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

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

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

Every service account is also a member of two groups:

**system:serviceaccounts**
: Includes all service accounts in the system.

**system:serviceaccounts:<project>**
: Includes all service accounts in the specified project.

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

```
$ oc policy add-role-to-group view system: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
```

## 6.3. ENABLING SERVICE ACCOUNT AUTHENTICATION

Service accounts authenticate to the API using tokens signed by a private RSA key. The authentication layer verifies the signature using a matching public RSA key.

To enable service account token generation, update the **`serviceAccountConfig`** stanza in the ***/etc/origin/master/master-config.yml*** file on the master to specify a **`privateKeyFile`** (for signing), and a matching public key file in the **`publicKeyFiles`** list:

```
serviceAccountConfig:
  ...
  masterCA: ca.crt 1
  privateKeyFile: 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.

## 6.4. MANAGED SERVICE ACCOUNTS

Service accounts are required in each project to run builds, deployments, and other pods. The **`managedNames`** setting in the ***/etc/origin/master/master-config.yml*** file on the master controls which service accounts are automatically created in every project:

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

1

List of service accounts to automatically create in every project.

2

A **builder** service account in each project is required by build pods, and is given the **system:image-builder** role, which allows pushing images to any image stream in the project using the internal container registry.

3

A **deployer** service account in each project is required by deployment pods, and is given the **system:deployer** role, which allows viewing and modifying replication controllers and pods in the project.

4

A **default** service account is used by all other pods unless they specify a different service account.

All service accounts in a project are given the **system:image-puller** role, which allows pulling images from any image stream in the project using the internal container registry.

## 6.5. INFRASTRUCTURE SERVICE ACCOUNTS

Several infrastructure controllers run using service account credentials. The following service accounts are created in the OpenShift Container Platform infrastructure project (**openshift-infra**) at server start, and given the following roles cluster-wide:

| Service Account | Description |
|---|---|
| **replication-controller** | Assigned the **system:replication-controller** role |
| **deployment-controller** | Assigned the **system:deployment-controller** role |
| **build-controller** | Assigned the **system:build-controller** role. Additionally, the **build-controller** service account is included in the privileged security context constraint in order to create privileged build pods. |

To configure the project where those service accounts are created, set the **`openshiftInfrastructureNamespace`** field in in the ***/etc/origin/master/master-config.yml*** file on the master:

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

## 6.6. SERVICE ACCOUNTS AND SECRETS

Set the **`limitSecretReferences`** field in the ***/etc/origin/master/master-config.yml*** file on the master to **`true`** to require pod secret references to be whitelisted by their service accounts. Set its value to **`false`** to allow pods to reference any secret in the project.

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

# CHAPTER 7. MANAGING AUTHORIZATION POLICIES

## 7.1. OVERVIEW

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

## 7.2. 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.

**Note**

Review a full list of verbs in the Evaluating Authorization section.

### 7.2.1. Viewing Cluster Policy

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

```
$ oc describe clusterPolicy default
```

**Example 7.1. Viewing Cluster Roles**

```
$ oc describe clusterPolicy default
Name:     default
Created:    5 days ago
Labels:     <none>
Annotations:    <none>
Last Modified:    2016-03-17 13:25:27 -0400 EDT
admin     Verbs       Non-Resource URLs  Extension   Resource Names
API Groups   Resources
     [create delete deletecollection get list patch update watch] []
[]   []    [configmaps endpoints persistentvolumeclaims pods
pods/attach pods/exec pods/log pods/portforward pods/proxy
replicationcontrollers replicationcontrollers/scale secrets
serviceaccounts services services/proxy]
     [create delete deletecollection get list patch update watch] []
[]   []    [buildconfigs buildconfigs/instantiate
buildconfigs/instantiatebinary buildconfigs/webhooks buildlogs builds
builds/clone builds/custom builds/docker builds/log builds/source
deploymentconfigrollbacks deploymentconfigs deploymentconfigs/log
deploymentconfigs/scale deployments generatedeploymentconfigs
imagestreamimages imagestreamimports imagestreammappings imagestreams
imagestreams/secrets imagestreamtags localresourceaccessreviews
localsubjectaccessreviews processedtemplates projects
resourceaccessreviews rolebindings roles routes subjectaccessreviews
templateconfigs templates]
     [create delete deletecollection get list patch update watch] []
[]   [autoscaling]   [horizontalpodautoscalers]
     [create delete deletecollection get list patch update watch] []
[]   [batch]    [jobs]
     [create delete deletecollection get list patch update watch] []
[]   [extensions]   [daemonsets horizontalpodautoscalers jobs
replicationcontrollers/scale]
     [get list watch]      []    []   []    [bindings configmaps
endpoints events imagestreams/status limitranges minions namespaces
namespaces/status nodes persistentvolumeclaims persistentvolumes
pods pods/log pods/status policies policybindings
replicationcontrollers replicationcontrollers/status resourcequotas
resourcequotas/status resourcequotausages routes/status
securitycontextconstraints serviceaccounts services]
     [get update]       []    []   []    [imagestreams/layers]
     [update]       []    []   []    [routes/status]
basic-user    Verbs        Non-Resource URLs  Extension   Resource
Names  API Groups   Resources
     [get]        []    [~]   []    [users]
     [list]        []    []   []    [projectrequests]
     [get list]       []    []   []    [clusterroles]
     [list]        []    []   []    [projects]
     [create]       []    IsPersonalSubjectAccessReview []   []
[localsubjectaccessreviews subjectaccessreviews]
cluster-admin    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [*]        []    []   [*]    [*]
     [*]        [*]    []   []    []
cluster-reader    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [bindings buildconfigs
buildconfigs/instantiate buildconfigs/instantiatebinary
buildconfigs/webhooks buildlogs builds builds/clone builds/details
builds/log clusternetworks clusterpolicies clusterpolicybindings
clusterrolebindings clusterroles configmaps deploymentconfigrollbacks
deploymentconfigs deploymentconfigs/log deploymentconfigs/scale
deployments endpoints events generatedeploymentconfigs groups
hostsubnets identities images imagestreamimages imagestreamimports
imagestreammappings imagestreams imagestreams/status imagestreamtags
limitranges localresourceaccessreviews localsubjectaccessreviews
minions namespaces netnamespaces nodes oauthclientauthorizations
oauthclients persistentvolumeclaims persistentvolumes pods pods/log
policies policybindings processedtemplates projectrequests projects
replicationcontrollers resourceaccessreviews resourcequotas
resourcequotausages rolebindings roles routes routes/status
securitycontextconstraints serviceaccounts services
subjectaccessreviews templateconfigs templates useridentitymappings
users]
     [get list watch]      []    []   [autoscaling]
[horizontalpodautoscalers]
     [get list watch]      []    []   [batch]    [jobs]
     [get list watch]      []    []   [extensions]   [daemonsets
horizontalpodautoscalers jobs replicationcontrollers/scale]
```

```
     [create]       []    []   []    [resourceaccessreviews
subjectaccessreviews]
     [get]        []    []   []    [nodes/metrics]
     [create get]       []    []   []    [nodes/stats]
     [get]        [*]    []   []    []
cluster-status    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get]        [/api /api/* /apis /apis/* /healthz /healthz/*
/oapi /oapi/* /osapi /osapi/ /version]     []   []  []
edit     Verbs        Non-Resource URLs  Extension   Resource Names
API Groups   Resources
     [create delete deletecollection get list patch update watch] []
[]   []    [configmaps endpoints persistentvolumeclaims pods
pods/attach pods/exec pods/log pods/portforward pods/proxy
replicationcontrollers replicationcontrollers/scale secrets
serviceaccounts services services/proxy]
     [create delete deletecollection get list patch update watch] []
[]   []    [buildconfigs buildconfigs/instantiate
buildconfigs/instantiatebinary buildconfigs/webhooks buildlogs builds
builds/clone builds/custom builds/docker builds/log builds/source
deploymentconfigrollbacks deploymentconfigs deploymentconfigs/log
deploymentconfigs/scale deployments generatedeploymentconfigs
imagestreamimages imagestreamimports imagestreammappings imagestreams
imagestreams/secrets imagestreamtags processedtemplates routes
templateconfigs templates]
     [create delete deletecollection get list patch update watch] []
[]   [autoscaling]   [horizontalpodautoscalers]
     [create delete deletecollection get list patch update watch] []
[]   [batch]    [jobs]
     [create delete deletecollection get list patch update watch] []
[]   [extensions]   [daemonsets horizontalpodautoscalers jobs
replicationcontrollers/scale]
     [get list watch]      []    []   []    [bindings configmaps
endpoints events imagestreams/status limitranges minions namespaces
namespaces/status nodes persistentvolumeclaims persistentvolumes
pods pods/log pods/status projects replicationcontrollers
replicationcontrollers/status resourcequotas resourcequotas/status
resourcequotausages routes/status securitycontextconstraints
serviceaccounts services]
     [get update]       []    []   []    [imagestreams/layers]
registry-admin    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [create delete deletecollection get list patch update watch] []
[]   []    [imagestreamimages imagestreamimports imagestreammappings
imagestreams imagestreams/secrets imagestreamtags]
     [create delete deletecollection get list patch update watch] []
[]   []    [localresourceaccessreviews localsubjectaccessreviews
resourceaccessreviews rolebindings roles subjectaccessreviews]
     [get update]       []    []   []    [imagestreams/layers]
     [get list watch]      []    []   []    [policies
policybindings]
     [get]        []    []   []    [namespaces projects]
registry-editor    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get]        []    []   []    [namespaces projects]
     [create delete deletecollection get list patch update watch] []
```

```
[]   []    [imagestreamimages imagestreamimports imagestreammappings
imagestreams imagestreams/secrets imagestreamtags]
     [get update]       []    []   []    [imagestreams/layers]
registry-viewer    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [imagestreamimages
imagestreamimports imagestreammappings imagestreams imagestreamtags]
     [get]        []    []   []    [imagestreams/layers namespaces
projects]
self-provisioner   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [create]       []    []   []    [projectrequests]
system:build-controller   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [builds]
     [update]       []    []   []    [builds]
     [create]       []    []   []    [builds/custom builds/docker
builds/source]
     [get]        []    []   []    [imagestreams]
     [create delete get list]     []    []   []    [pods]
     [create patch update]      []    []   []    [events]
system:daemonset-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   [extensions]   [daemonsets]
     [list watch]       []    []   []    [pods]
     [list watch]       []    []   []    [nodes]
     [update]       []    []   [extensions]   [daemonsets/status]
     [create delete]       []    []   []    [pods]
     [create]       []    []   []    [pods/binding]
     [create patch update]      []    []   []    [events]
system:deployer    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list]       []    []   []    [replicationcontrollers]
     [get update]       []    []   []    [replicationcontrollers]
     [create get list watch]      []    []   []    [pods]
     [get]        []    []   []    [pods/log]
     [update]       []    []   []    [imagestreamtags]
system:deployment-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   []    [replicationcontrollers]
     [get update]       []    []   []    [replicationcontrollers]
     [create delete get list update]     []    []   []    [pods]
     [create patch update]      []    []   []    [events]
system:discovery   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get]        [/api /api/* /apis /apis/* /oapi /oapi/* /osapi
/osapi/ /version]       []   []   []
system:hpa-controller   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   [extensions autoscaling]
[horizontalpodautoscalers]
     [update]       []    []   [extensions autoscaling]
[horizontalpodautoscalers/status]
     [get update]       []    []   [extensions ]
[replicationcontrollers/scale]
     [get update]       []    []   []    [deploymentconfigs/scale]
     [create patch update]      []    []   []    [events]
     [list]        []    []   []    [pods]
     [proxy]        []    [https:heapster:] []    [services]
system:image-builder   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get update]       []    []   []    [imagestreams/layers]
     [update]       []    []   []    [builds/details]
system:image-pruner   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [delete]       []    []   []    [images]
     [get list]       []    []   []    [buildconfigs builds
deploymentconfigs images imagestreams pods replicationcontrollers]
     [update]       []    []   []    [imagestreams/status]
system:image-puller   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get]        []    []   []    [imagestreams/layers]
system:image-pusher   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get update]       []    []   []    [imagestreams/layers]
system:job-controller   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [list watch]       []    []   [extensions batch]  [jobs]
     [update]       []    []   [extensions batch]  [jobs/status]
     [list watch]       []    []   []    [pods]
     [create delete]       []    []   []    [pods]
     [create patch update]      []    []   []    [events]
system:master    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [*]        []    []   [*]    [*]
system:namespace-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [delete get list watch]      []    []   []    [namespaces]
     [update]       []    []   []    [namespaces/finalize
namespaces/status]
     [delete deletecollection get list]    []    []   [*]    [*]
system:node    Verbs        Non-Resource URLs  Extension   Resource
Names  API Groups   Resources
     [create]       []    []   []    [localsubjectaccessreviews
subjectaccessreviews]
     [get list watch]      []    []   []    [services]
     [create get list watch]      []    []   []    [nodes]
     [update]       []    []   []    [nodes/status]
     [create patch update]      []    []   []    [events]
     [get list watch]      []    []   []    [pods]
     [create delete get]      []    []   []    [pods]
     [update]       []    []   []    [pods/status]
     [get]        []    []   []    [configmaps secrets]
     [get]        []    []   []    [persistentvolumeclaims
persistentvolumes]
     [get]        []    []   []    [endpoints]
system:node-admin   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [nodes]
     [proxy]        []    []   []    [nodes]
     [*]        []    []   []    [nodes/log nodes/metrics
nodes/proxy nodes/stats]
```

```
system:node-proxier   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [list watch]       []    []   []    [endpoints services]
system:node-reader   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [nodes]
     [get]        []    []   []    [nodes/metrics]
     [create get]       []    []   []    [nodes/stats]
system:oauth-token-deleter  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [delete]       []    []   []    [oauthaccesstokens
oauthauthorizetokens]
system:pv-binder-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   []    [persistentvolumes]
     [create delete get update]     []    []   []
[persistentvolumes]
     [update]       []    []   []    [persistentvolumes/status]
     [list watch]       []    []   []    [persistentvolumeclaims]
     [get update]       []    []   []    [persistentvolumeclaims]
     [update]       []    []   []
[persistentvolumeclaims/status]
system:pv-provisioner-controller Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   []    [persistentvolumes]
     [create delete get update]     []    []   []
[persistentvolumes]
     [update]       []    []   []    [persistentvolumes/status]
     [list watch]       []    []   []    [persistentvolumeclaims]
     [get update]       []    []   []    [persistentvolumeclaims]
     [update]       []    []   []
[persistentvolumeclaims/status]
system:pv-recycler-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   []    [persistentvolumes]
     [create delete get update]     []    []   []
[persistentvolumes]
     [update]       []    []   []    [persistentvolumes/status]
     [list watch]       []    []   []    [persistentvolumeclaims]
     [get update]       []    []   []    [persistentvolumeclaims]
     [update]       []    []   []
[persistentvolumeclaims/status]
     [list watch]       []    []   []    [pods]
     [create delete get]      []    []   []    [pods]
     [create patch update]      []    []   []    [events]
system:registry    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [delete get]       []    []   []    [images]
     [get]        []    []   []    [imagestreamimages imagestreams
imagestreams/secrets imagestreamtags]
     [update]       []    []   []    [imagestreams]
     [create]       []    []   []    [imagestreammappings]
     [list]        []    []   []    [resourcequotas]
system:replication-controller  Verbs        Non-Resource URLs
Extension   Resource Names  API Groups   Resources
     [list watch]       []    []   []    [replicationcontrollers]
     [get update]       []    []   []    [replicationcontrollers]
     [update]       []    []   []
[replicationcontrollers/status]
     [list watch]       []    []   []    [pods]
     [create delete]       []    []   []    [pods]
     [create patch update]      []    []   []    [events]
system:router    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [list watch]       []    []   []    [endpoints routes]
     [update]       []    []   []    [routes/status]
system:sdn-manager   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [create delete get list watch]     []    []   []
[hostsubnets]
     [create delete get list watch]     []    []   []
[netnamespaces]
     [get list watch]      []    []   []    [nodes]
     [create get]       []    []   []    [clusternetworks]
system:sdn-reader   Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [get list watch]      []    []   []    [hostsubnets]
     [get list watch]      []    []   []    [netnamespaces]
     [get list watch]      []    []   []    [nodes]
     [get]        []    []   []    [clusternetworks]
     [get list watch]      []    []   []    [namespaces]
system:webhook    Verbs        Non-Resource URLs  Extension
Resource Names  API Groups   Resources
     [create get]       []    []   []    [buildconfigs/webhooks]
view     Verbs        Non-Resource URLs  Extension   Resource Names
API Groups   Resources
     [get list watch]      []    []   []    [bindings buildconfigs
buildconfigs/instantiate buildconfigs/instantiatebinary
buildconfigs/webhooks buildlogs builds builds/clone builds/log
configmaps deploymentconfigrollbacks deploymentconfigs
deploymentconfigs/log deploymentconfigs/scale deployments endpoints
events generatedeploymentconfigs imagestreamimages imagestreamimports
imagestreammappings imagestreams imagestreams/status imagestreamtags
limitranges minions namespaces namespaces/status nodes
persistentvolumeclaims persistentvolumes pods pods/log pods/status
processedtemplates projects replicationcontrollers
replicationcontrollers/status resourcequotas resourcequotas/status
resourcequotausages routes routes/status securitycontextconstraints
serviceaccounts services templateconfigs templates]
     [get list watch]      []    []   [autoscaling]
[horizontalpodautoscalers]
     [get list watch]      []    []   [batch]    [jobs]
     [get list watch]      []    []   [extensions]   [daemonsets
horizontalpodautoscalers jobs]
```

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 7.2. 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]
```

### 7.2.2. 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 7.3. 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.

## 7.3. 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 7.1. 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. |

| Command | Description |
|---|---|
| **$ 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 for these operations because the cluster policy uses non-namespaced resources.

**Table 7.2. 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`**.

## 7.4. GRANTING USERS DAEMONSET PERMISSIONS

By default, project developers do not have the permission to create daemonsets. As a cluster administrator, you can grant them the abilities.

1. Define a ***ClusterRole*** file:

```
- apiVersion: v1
  kind: ClusterRole
  metadata:
    name: daemonset-admin
  rules:
  - resources:
    - daemonsets
    apiGroups:
    - extensions
    verbs:
    - create
    - get
    - list
    - watch
    - delete
    - update
```

2. Create the role:

```
$ oadm policy add-role-to-user system:daemonset-admin <user>
```

## 7.5. CREATING A LOCAL ROLE

To create a local role for a project, you can either copy and modify an existing role or build a new role from scratch. It is recommended that you build it from scratch so that you understand each of the permissions assigned.

To copy the cluster role **view** to use as a local role, run:

```
$ oc get clusterrole view -o yaml > clusterrole_view.yaml
$ cp clusterrole_view.yaml localrole_exampleview.yaml
$ vim localrole_exampleview.yaml
# 1. Update kind: ClusterRole to kind: Role
# 2. Update name: view to name: exampleview
# 3. Remove resourceVersion, selfLink, uid, and creationTimestamp
$ oc create -f path/to/localrole_exampleview.yaml -n
<project_you_want_to_add_the_local_role_exampleview_to>
```

To create a new role from scratch, save this snippet into the file ***role_exampleview.yaml***:

### Example Role Named exampleview

```
apiVersion: v1
kind: Role
metadata:
  name: exampleview
rules:
- apiGroups: null
  attributeRestrictions: null
  resources:
  - pods
  - builds
  verbs:
  - get
 - list
 - watch
```

Then, to use the current project, run:

```
$ oc project <project_you_want_to_add_the_local_role_exampleview_to>
```

Optionally, annotate it with a description.

To use the new role, run:

```
$ oadm policy add-role-to-user exampleview user2
```

**Note**

A **`clusterrolebinding`** is a role binding that exists at the cluster level. A **`rolebinding`** exists at the project level. This can be confusing. The **`clusterrolebinding`** *view* must be assigned to a user within a project for that user to view the project. Local roles are only created if a cluster role does not provide the set of permissions needed for a particular situation, which is unlikely.

Some cluster role names are initially confusing. The **`clusterroleclusteradmin`** can be assigned to a user within a project, making it appear that this user has the privileges of a cluster administrator. This is not the case. The **`clusteradmin`** cluster role bound to a certain project is more like a super administrator for that project, granting the permissions of the cluster role **`admin`**, plus a few additional permissions like the ability to edit rate limits. This can appear especially confusing via the web console UI, which does not list cluster policy (where cluster administrators exist). However, it does list local policy (where a locally bound **`clusteradmin`** may exist).

Within a project, project administrators should be able to see **`rolebindings`**, not **`clusterrolebindings`**.

# CHAPTER 8. IMAGE POLICY

## 8.1. OVERVIEW

You can control which images are allowed to run on your cluster using the ImagePolicy admission plug-in (currently considered beta). It allows you to control:

- **The source of images**: which registries can be used to pull images
- **Image resolution**: force pods to run with immutable digests to ensure the image does not change due to a re-tag
- **Container image label restrictions**: force an image to have or not have particular labels
- **Image annotation restrictions**: force an image in the integrated container registry to have or not have particular annotations

## 8.2. CONFIGURING THE IMAGEPOLICY ADMISSION PLUG-IN

To enable this feature, configure the plug-in in ***master-config.yaml***:

**Example 8.1. Annotated Example File**

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
        kind: ImagePolicyConfig
        apiVersion: v1
        resolveImages: AttemptRewrite 1
        executionRules: 2
        - name: execution-denied
          # Reject all images that have the annotation
images.openshift.io/deny-execution set to true.
          # This annotation may be set by infrastructure that wishes
to flag particular images as dangerous
          onResources: 3
          - resource: pods
          - resource: builds
          reject: true 4
          matchImageAnnotations: 5
          - key: images.openshift.io/deny-execution
            value: "true"
          skipOnResolutionFailure: true 6
        - name: allow-images-from-internal-registry
          # allows images from the internal registry and tries to
resolve them
          onResources:
          - resource: pods
          - resource: builds
          matchIntegratedRegistry: true
        - name: allow-images-from-dockerhub
          onResources:
          - resource: pods
          - resource: builds
          matchRegistries:
          - docker.io
```

1

Try to resolve images to an immutable image digest and update the image pull specification in the pod.

2

Array of rules to evaluate against incoming resources. If you only have reject==true rules, the default is **allow all**. If you have any accept rule, the default is **deny all**.

3

Indicates which resources to enforce rules upon. If nothing is specified, the default is **pods**.

4

Indicates that if this rule matches, the pod should be rejected.

5

List of annotations to match on the image object's metadata.

6

If you are not able to resolve the image, do not fail the pod.

## 8.3. TESTING THE IMAGEPOLICY ADMISSION PLUG-IN

1. Use the **`openshift/image-policy-check`** to test your configuration.

   For example, use the information above, then test like this:

   ```
   oc import-image openshift/image-policy-check:latest --confirm
   ```

2. Create a pod using this YAML. The pod should be created.

   ```
   apiVersion: v1
   kind: Pod
   ```

```
metadata:
  generateName: test-pod
spec:
  containers:
  - image: docker.io/openshift/image-policy-check:latest
    name: first
```

3. Create another pod pointing to a different registry. The pod should be rejected.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: different-registry/openshift/image-policy-check:latest
    name: first
```

4. Create a pod pointing to the internal registry using the imported image. The pod should be created and if you look at the image specification, you should see a digest in place of the tag.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
check:latest
    name: first
```

5. Create a pod pointing to the internal registry using the imported image. The pod should be created and if you look at the image specification, you should see the tag unmodified.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
check:v1
    name: first
```

6. Get the digest from **`oc get istag/image-policy-check:latest`** and use it for **`oc annotate images/<digest> images.openshift.io/deny-execution=true`**. For example:

```
$ oc annotate
images/sha256:09ce3d8b5b63595ffca6636c7daefb1a615a7c0e3f8ea68e5db
044a9340d6ba8 images.openshift.io/deny-execution=true
```

7. Create this pod again, and you should see the pod rejected:

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
check:latest
    name: first
```

# CHAPTER 9. SCOPED TOKENS

## 9.1. OVERVIEW

A user may want to give another entity the power to act as they have, but only in a limited way. For example, a project administrator may want to delegate the power to create pods. One way to do this is to create a scoped token.

A scoped token is a token that identifies as a given user, but is limited to certain actions by its scope. Right now, only a **cluster-admin** can create scoped tokens.

## 9.2. EVALUATION

Scopes are evaluated by converting the set of scopes for a token into a set of **PolicyRules**. Then, the request is matched against those rules. The request attributes must match at least one of the scope rules to be passed to the "normal" authorizer for further authorization checks.

## 9.3. USER SCOPES

User scopes are focused on getting information about a given user. They are intent-based, so the rules are automatically created for you:

- **`user:full`** - Allows full read/write access to the API with all of the user’s permissions.
- **`user:info`** - Allows read-only access to information about the user: name, groups, and so on.
- **`user:check-access`** - Allows access to **self-localsubjectaccessreviews** and **self-subjectaccessreviews**. These are the variables where you pass an empty user and groups in your request object.
- **`user:list-projects`** - Allows read-only access to list the projects the user has access to.

## 9.4. ROLE SCOPE

The role scope allows you to have the same level of access as a given role filtered by namespace.

- **`role:<cluster-role name>:<namespace or * for all>`** - Limits the scope to the rules specified by the cluster-role, but only in the specified namespace .

**Note**

Caveat: This prevents escalating access. Even if the role allows access to resources like secrets, rolebindings, and roles, this scope will deny access to those resources. This helps prevent unexpected escalations. Many people do not think of a role like **edit** as being an escalating role, but with access to a secret it is.

- **`role:<cluster-role name>:<namespace or * for all>:!`** - This is similar to the example above, except that including the bang causes this scope to allow escalating access.

# CHAPTER 10. MONITORING IMAGES

## 10.1. OVERVIEW

You can monitor images in your instance using the CLI.

## 10.2. VIEWING IMAGES STATISTICS

OpenShift Container Platform can display several usage statistics about all the images it manages. In other words, all the images pushed to the internal registry either directly or through a build.

To view the usage statistics:

```
$ oadm top images
NAME                 IMAGESTREAMTAG            PARENTS
USAGE                            METADATA    STORAGE
sha256:80c985739a78b openshift/python (3.5)
yes         303.12MiB
sha256:64461b5111fc7 openshift/ruby (2.2)
yes         234.33MiB
sha256:0e19a0290ddc1 test/ruby-ex (latest)     sha256:64461b5111fc71ec
Deployment: ruby-ex-1/test    yes         150.65MiB
sha256:a968c61adad58 test/django-ex (latest)   sha256:80c985739a78b760
Deployment: django-ex-1/test  yes         186.07MiB
```

The command displays the following information:

- image ID
- project, name, and tag of the accompanying **`ImageStreamTag`**
- potential parents of the image, using their ID
- information about where the image is being used
- flag informing whether the image contains proper Docker metadata information
- size of the image

## 10.3. VIEWING IMAGESTREAMS STATISTICS

OpenShift Container Platform can display several usage statistics about all the **`ImageStreams`**.

To view the usage statistics:

```
$ oadm top imagestreams
NAME                STORAGE     IMAGES   LAYERS
openshift/python    1.21GiB     4        36
openshift/ruby      717.76MiB   3        27
test/ruby-ex        150.65MiB   1        10
test/django-ex      186.07MiB   1        10
```

The command displays the following information:

- project and name of the **`ImageStream`**
- size of the entire **`ImageStream`** stored in the internal Red Hat Container Registry
- number of images this particular **`ImageStream`** is pointing to
- number of layers **`ImageStream`** consists of

## 10.4. PRUNING IMAGES

The information returned from the above commands is helpful when performing image pruning.

# CHAPTER 11. MANAGING SECURITY CONTEXT CONSTRAINTS

## 11.1. OVERVIEW

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

**Note**

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

## 11.2. LISTING SECURITY CONTEXT CONSTRAINTS

To get a current list of SCCs:

```
$ oc get scc

NAME               PRIV      CAPS      SELINUX     RUNASUSER
FSGROUP     SUPGROUP    PRIORITY   READONLYROOTFS   VOLUMES
anyuid             false     []        MustRunAs   RunAsAny
RunAsAny    RunAsAny    10         false            [configMap
downwardAPI emptyDir persistentVolumeClaim secret]
hostaccess         false     []        MustRunAs   MustRunAsRange
MustRunAs   RunAsAny    <none>     false            [configMap
downwardAPI emptyDir hostPath persistentVolumeClaim secret]
hostmount-anyuid   false     []        MustRunAs   RunAsAny
RunAsAny    RunAsAny    <none>     false            [configMap
downwardAPI emptyDir hostPath persistentVolumeClaim secret]
hostnetwork        false     []        MustRunAs   MustRunAsRange
MustRunAs   MustRunAs   <none>     false            [configMap
downwardAPI emptyDir persistentVolumeClaim secret]
nonroot            false     []        MustRunAs   MustRunAsNonRoot
RunAsAny    RunAsAny    <none>     false            [configMap
downwardAPI emptyDir persistentVolumeClaim secret]
privileged         true      []        RunAsAny    RunAsAny
RunAsAny    RunAsAny    <none>     false            [*]
restricted         false     []        MustRunAs   MustRunAsRange
MustRunAs   RunAsAny    <none>     false            [configMap
downwardAPI emptyDir persistentVolumeClaim secret]
```

## 11.3. EXAMINING A SECURITY CONTEXT CONSTRAINTS OBJECT

To examine a particular SCC, use **`oc get`**, **`oc describe`**, **`oc export`**, or **`oc edit`**. For example, to examine the **restricted** SCC:

```
$ oc describe scc restricted
```

```
Name:      restricted
Priority:     <none>
Access:
  Users:     <none>
  Groups:     system:authenticated
Settings:
  Allow Privileged:    false
  Default Add Capabilities:   <none>
  Required Drop Capabilities:   <none>
  Allowed Capabilities:    <none>
  Allowed Volume Types:
awsElasticBlockStore,azureFile,cephFS,cinder,configMap,downwardAPI,empt
yDir,fc,flexVolume,flocker,gcePersistentDisk,gitRepo,glusterfs,iscsi,nf
s,persistentVolumeClaim,rbd,secret
  Allow Host Network:    false
  Allow Host Ports:    false
  Allow Host PID:    false
  Allow Host IPC:    false
  Read Only Root Filesystem:   false
  Run As User Strategy: MustRunAsRange
    UID:     <none>
    UID Range Min:    <none>
    UID Range Max:    <none>
  SELinux Context Strategy: MustRunAs
    User:     <none>
    Role:     <none>
    Type:     <none>
    Level:     <none>
  FSGroup Strategy: RunAsAny
    Ranges:     <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:     <none>
```

**Note**

In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs other than priority, users, and groups.

## 11.4. CREATING NEW SECURITY CONTEXT CONSTRAINTS

To create a new SCC:

1. Define the SCC in a JSON or YAML file:

**Example 11.1. Security Context Constraint Object Definition**

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: scc-admin
```

```
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
- my-admin-user
groups:
- my-admin-group
```

Optionally, you can add drop capabilities to an SCC by setting the **`requiredDropCapabilities:`** field with the desired values. Any specified capabilities will be dropped from the container. For example, to create an SCC with the **`KILL`**, **`MKNOD`**, and **`SYS_CHROOT`** required drop capabilities, add the following to the SCC object:

```
requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT
```

You can see the list of possible values in the Docker documentation.

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

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

3. Verify that the SCC was created:

```
$ 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
```

## 11.5. 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.

## 11.6. UPDATING SECURITY CONTEXT CONSTRAINTS

To update an existing SCC:

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

**Note**

In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs other than priority, users, and groups.

## 11.7. UPDATING THE DEFAULT SECURITY CONTEXT CONSTRAINTS

Default SCCs will be created when the master is started if they are missing. To reset SCCs to defaults, or update existing SCCs to new default definitions after an upgrade you may:

1. Delete any SCC you would like to be reset and let it be recreated by restarting the master
2. Use the **`oadm policy reconcile-sccs`** command

The **`oadm policy reconcile-sccs`** command will set all SCC policies to the default values but retain any additional users and groups as well as priorities you may have already set. To view which SCCs will be changed you may run the command with no options or by specifying your preferred output with the **`-o <format>`** option.

After reviewing it is recommended that you back up your existing SCCs and then use the **`--confirm`** option to persist the data.

**Note**

If you would like to reset priorities and grants, use the **`--additive-only=false`** option.

**Note**

If you have customized settings other than priority, users, or groups in an SCC, you will lose those settings when you reconcile.

## 11.8. HOW DO I?

The following describe common scenarios and procedures using SCCs.

### 11.8.1. 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:

```
$ oadm policy add-scc-to-user <scc_name> <user_name>
$ oadm policy add-scc-to-group <scc_name> <group_name>
```

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

```
$ oadm policy add-scc-to-user privileged e2e-user
```

### 11.8.2. Grant a Service Account Access to the Privileged SCC

First, create a service account. For example, to create service account **`mysvcacct`** in project **`myproject`**:

```
$ oc create serviceaccount mysvcacct -n myproject
```

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

```
$ oadm policy add-scc-to-user privileged
system:serviceaccount:myproject:mysvcacct
```

### 11.8.3. Enable Images to Run with USER in the Dockerfile

To relax the security in your cluster so that images are not forced to run as a pre-allocated UID, without granting everyone access to the **privileged** SCC:

1. Grant all authenticated users access to the **anyuid** SCC:

```
$ oadm policy add-scc-to-group anyuid system:authenticated
```

**Warning**

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

### 11.8.4. Enable Container Images that Require Root

Some container images (examples: **`postgres`** and **`redis`**) require root access and have certain expectations about how volumes are owned. For these images, add the service account to the **`anyuid`** SCC.

```
$ oadm policy add-scc-to-user anyuid
system:serviceaccount:myproject:mysvcacct
```

### 11.8.5. Use --mount-host on the Registry

It is recommended that persistent storage using **`PersistentVolume`** and **`PersistentVolumeClaim`** objects be used for registry deployments. If you are testing and would like to instead use the **`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.

### 11.8.6. Provide Additional Capabilities

In some cases, an image may require capabilities that Docker does not provide out of the box. You can provide the ability to request additional capabilities in the pod specification which will be validated against an SCC.

**Important**

This allows images to run with elevated capabilities and should be used only if necessary. You should not edit the default **restricted** SCC to enable additional capabilities.

When used in conjunction with a non-root user, you must also ensure that the file that requires the additional capability is granted the capabilities using the **`setcap`** command. For example, in the ***Dockerfile*** of the image:

```
setcap cap_net_raw,cap_net_admin+p /usr/bin/ping
```

Further, if a capability is provided by default in Docker, you do not need to modify the pod specification to request it. For example, **`NET_RAW`** is provided by default and capabilities should already be set on **`ping`**, therefore no special steps should be required to run **`ping`**.

To provide additional capabilities:

1. Create a new SCC
2. Add the allowed capability using the **`allowedCapabilities`** field.
3. When creating the pod, request the capability in the **`securityContext.capabilities.add`** field.

### 11.8.7. 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:

**Note**

In order to preserve customized SCCs during upgrades, do not edit settings on the default SCCs other than priority, users, and groups.

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.

### 11.8.8. Use the hostPath Volume Plug-in

To relax the security in your cluster so that pods are allowed to use the **`hostPath`** volume plug-in without granting everyone access to the **privileged** SCC:

1. Edit the **restricted** SCC:

   ```
   $ oc edit scc restricted
   ```

2. Add **`allowHostDirVolumePlugin: true`**.
3. Save the changes.

### 11.8.9. Ensure That Admission Attempts to Use a Specific SCC First

You may control the sort ordering of SCCs in admission by setting the **`Priority`** field of the SCCs. Please see the SCC Prioritization section for more information on sorting.

### 11.8.10. Add an SCC to a User or Group

To add an SCC to a user:

```
$ oadm policy add-scc-to-user <scc_name> <user_name>
```

To add an SCC to a service account:

```
$ oadm policy add-scc-to-user <scc_name>  \
    system:serviceaccount:<serviceaccount_namespace>:
<serviceaccount_name>
```

To add an SCC to a group:

–

```
$ oadm policy add-scc-to-group <scc_name> <group_name>
```

To add an SCC to all service accounts in a namespace:

```
$ oadm policy add-scc-to-group <scc_name>  \
    system:serviceaccounts:<serviceaccount_namespace>
```

# CHAPTER 12. SETTING QUOTAS

## 12.1. OVERVIEW

A resource quota, defined by a **`ResourceQuota`** object, provides constraints that limit aggregate resource consumption per project. It can limit the quantity of objects that can be created in a project by type, as well as the total amount of compute resources and storage that may be consumed by resources in that project.

**Note**

See the Developer Guide for more on compute resources.

## 12.2. RESOURCES MANAGED BY QUOTA

The following describes the set of compute resources and object types that may be managed by a quota.

**Table 12.1. Compute Resources Managed by Quota**

| Resource Name | Description |
|---|---|
| **`cpu`** | Across all pods in a non-terminal state, the sum of CPU requests cannot exceed this value. **`cpu`** and `**requests.cpu**`are the same value and can be used interchangeably. |
| **`memory`** | Across all pods in a non-terminal state, the sum of memory requests cannot exceed this value. **`memory`** and **`requests.memory`** are the same value and can be used interchangeably. |
| **`requests.cpu`** | Across all pods in a non-terminal state, the sum of CPU requests cannot exceed this value. **`cpu`** and `**requests.cpu**`are the same value and can be used interchangeably. |
| **`requests.memory`** | Across all pods in a non-terminal state, the sum of memory requests cannot exceed this value. **`memory`** and **`requests.memory`** are the same value and can be used interchangeably. |
| **`requests.storage`** | Across all persistent volume claim in any state, the sum of storage requests cannot exceed this value. **`storage`** and **`requests.storage`** are the same value and can be used interchangeably. |

| Resource Name | Description |
|---|---|
| `limits.cpu` | Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value. |
| `limits.memory` | Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value. |
| `limits.storage` | Across all persistent volume claims, the sum of storage limits cannot exceed this value. |

**Table 12.2. Object Counts Managed by Quota**

| Resource Name | Description |
|---|---|
| `pods` | The total number of pods in a non-terminal state that can exist in the project. A pod is in a terminal state if `status.phase in (Failed, Succeeded)` is true. |
| `replicationcontrollers` | The total number of replication controllers that can exist in the project. |
| `resourcequotas` | The total number of resource quotas that can exist in the project. |
| `services` | The total number of services that can exist in the project. |
| `secrets` | The total number of secrets that can exist in the project. |
| `configmaps` | The total number of `ConfigMap` objects that can exist in the project. |
| `persistentvolumeclaims` | The total number of persistent volume claims that can exist in the project. |
| `openshift.io/imagestreams` | The total number of image streams that can exist in the project. |

## 12.3. QUOTA SCOPES

Each quota can have an associated set of *scopes*. A quota will only measure usage for a resource if it matches the intersection of enumerated scopes.

When a scope is added to a quota, it limits the number of resources it supports to those that pertain to the scope. Resources specified on the quota outside of the allowed set results in a validation error.

| Scope | Description |
|---|---|
| **Terminating** | Match pods where **`spec.activeDeadlineSeconds >= 0`** |
| **NotTerminating** | Match pods where **`spec.activeDeadlineSeconds is nil`** |
| **BestEffort** | Match pods that have best effort quality of service for either **`cpu`** or **`memory`**. |
| **NotBestEffort** | Match pods that do not have best effort quality of service for **`cpu`** and **`memory`**. |

A **BestEffort** scope restricts a quota to limit the following resources:

- **`pods`**

A **Terminating**, **NotTerminating**, and **NotBestEffort** scope restricts a quota to tracking the following resources:

- **`pods`**
- **`memory`**
- **`requests.memory`**
- **`limits.memory`**
- **`cpu`**
- **`requests.cpu`**
- **`limits.cpu`**

## 12.4. QUOTA ENFORCEMENT

After a resource quota for a project is first created, the project restricts the ability to create any new resources that may violate a quota constraint until it has calculated updated usage statistics.

After a quota is created and usage statistics are updated, the project accepts the creation of new content. When you create or modify resources, your quota usage is incremented immediately upon the request to create or modify the resource.

When you delete a resource, your quota use is decremented during the next full recalculation of quota statistics for the project. A configurable amount of time determines how long it takes to reduce quota usage statistics to their current observed system value.

If project modifications exceed a quota usage limit, the server denies the action, and an appropriate error message is returned to the user explaining the quota constraint violated, and what their currently observed usage stats are in the system.

## 12.5. REQUESTS VS LIMITS

When allocating compute resources, each container may specify a request and a limit value for either CPU or memory. The quota can be configured to quota either value.

If the quota has a value specified for **`requests.cpu`** or **`requests.memory`**, then it requires that every incoming container makes an explicit request for those resources. If the quota has a value specified for **`limits.cpu`** or **`limits.memory`**, then it requires that every incoming container specifies an explicit limit for those resources.

## 12.6. SAMPLE RESOURCE QUOTA DEFINITIONS

**Example 12.1. *object-counts.yaml***

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" 1
    persistentvolumeclaims: "4" 2
    replicationcontrollers: "20" 3
    secrets: "10" 4
    services: "10" 5
```

1 The total number of **`ConfigMap`** objects that can exist in the project.

2 The total number of persistent volume claims (PVCs) that can exist in the project.

3 The total number of replication controllers that can exist in the project.

4

The total number of secrets that can exist in the project.

5

The total number of services that can exist in the project.

**Example 12.2.** ***openshift-object-counts.yaml***

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" 1
```

1

The total number of image streams that can exist in the project.

**Example 12.3.** ***compute-resources.yaml***

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

1

The total number of pods in a non-terminal state that can exist in the project.

2

Across all pods in a non-terminal state, the sum of CPU requests cannot exceed 1 core.

3

Across all pods in a non-terminal state, the sum of memory requests cannot exceed 1Gi.

4

Across all pods in a non-terminal state, the sum of CPU limits cannot exceed 2 cores.

5

Across all pods in a non-terminal state, the sum of memory limits cannot exceed 2Gi.

**Example 12.4. *besteffort.yaml***

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
  hard:
    pods: "1" 1
  scopes:
  - BestEffort 2
```

1

The total number of pods in a non-terminal state with **BestEffort** quality of service that can exist in the project.

2

Restricts the quota to only matching pods that have **BestEffort** quality of service for either memory or CPU.

**Example 12.5. *compute-resources-long-running.yaml***

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" (1)
    limits.cpu: "4" (2)
    limits.memory: "2Gi" (3)
  scopes:
  - NotTerminating (4)
```

1. The total number of pods in a non-terminal state.

2. Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.

3. Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.

4. Restricts the quota to only matching pods where **`spec.activeDeadlineSeconds is nil`**. For example, this quota would not charge for build or deployer pods.

**Example 12.6. *compute-resources-time-bound.yaml***

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" (1)
    limits.cpu: "1" (2)
    limits.memory: "1Gi" (3)
  scopes:
  - Terminating (4)
```

1. The total number of pods in a non-terminal state.

2

Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.

3

Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.

4

Restricts the quota to only matching pods where **`spec.activeDeadlineSeconds >=0`**. For example, this quota would charge for build or deployer pods, but not long running pods like a web server or database.

**Example 12.7. storage-consumption.yaml**

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" 1
    requests.storage: "50Gi" 2
```

1

The total number of persistent volume claims in a project

2

Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.

## 12.7. CREATING A QUOTA

To create a quota, first define the quota to your specifications in a file, for example as seen in Sample Resource Quota Definitions. Then, create using that file to apply it to a project:

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

For example:

```
$ oc create -f resource-quota.json -n demoproject
```

## 12.8. VIEWING A QUOTA

You can view usage statistics related to any hard limits defined in a project's quota by navigating in the web console to the project's **Settings** tab.

You can also use the CLI to view quota details:

1. First, get the list of quotas defined in the project. For example, for a project called **demoproject**:

   ```
   $ oc get quota -n demoproject
   NAME                AGE
   besteffort          11m
   compute-resources   2m
   core-object-counts  29m
   ```

2. Then, describe the quota you are interested in, for example the **core-object-counts** quota:

   ```
   $ oc describe quota core-object-counts -n demoproject
   Name:   core-object-counts
   Namespace:  demoproject
   Resource  Used Hard
   --------  ---- ----
   configmaps  3 10
   persistentvolumeclaims 0 4
   replicationcontrollers 3 20
   secrets   9 10
   services  2 10
   ```

## 12.9. CONFIGURING QUOTA SYNCHRONIZATION PERIOD

When a set of resources are deleted, the synchronization time frame of resources is determined by the **`resource-quota-sync-period`** setting in the ***/etc/origin/master/master-config.yaml*** file.

Before quota usage is restored, a user may encounter problems when attempting to reuse the resources. You can change the **`resource-quota-sync-period`** setting to have the set of resources regenerate at the desired amount of time (in seconds) and for the resources to be available again:

```
kubernetesMasterConfig:
  apiLevels:
  - v1beta3
  - v1
  apiServerArguments: null
  controllerArguments:
    resource-quota-sync-period:
      - "10s"
```

After making any changes, restart the master service to apply them.

Adjusting the regeneration time can be helpful for creating resources and determining resource usage when automation is used.

**Note**

The **`resource-quota-sync-period`** setting is designed to balance system performance. Reducing the sync period can result in a heavy load on the master.

## 12.10. ACCOUNTING FOR QUOTA IN DEPLOYMENT CONFIGURATIONS

If a quota has been defined for your project, see Deployment Resources for considerations on any deployment configurations.

# CHAPTER 13. SETTING MULTI-PROJECT QUOTAS

## 13.1. OVERVIEW

A multi-project quota, defined by a **`ClusterResourceQuota`** object, allows quotas to be shared across multiple projects. Resources used in each selected project will be aggregated and that aggregate will be used to limit resources across all the selected projects.

## 13.2. SELECTING PROJECTS

Projects can be selected based on either annotation selection, label selection, or both. For example:

```
$ oc create clusterquota for-user \
     --project-annotation-selector openshift.io/requester=<user-name> \
     --hard pods=10 \
     --hard secrets=20
```

creates:

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: 1
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: 2
      openshift.io/requester: <user-name>
    labels: null 3
status:
  namespaces: 4
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: 5
    hard:
      pods: "10"
      secrets: "20"
    used:
      pods: "1"
      secrets: "9"
```

1

The **ResourceQuotaSpec** object that will be enforced over the selected projects.

2

A simple key/value selector for annotations.

3

A label selector that can be used to select projects.

4

A per-namespace map that describes current quota usage in each selected project.

5

The aggregate usage across all selected projects.

This multi-project quota document controls all projects requested by **<user-name>** using the default project request endpoint. You are limited to 10 pods and 20 secrets.

## 13.3. VIEWING APPLICABLE CLUSTERRESOURCEQUOTAS

A project administrator is not allowed to create or modify the multi-project quota that limits his or her project, but the administrator is allowed to view the multi-project quota documents that are applied to his or her project. The project administrator can do this via the **AppliedClusterResourceQuota** resource.

```
$ oc describe AppliedClusterResourceQuota
```

produces:

```
Name:   for-user
Namespace:  <none>
Created:  19 hours ago
Labels:   <none>
Annotations:  <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource  Used  Hard
--------  ----  ----
pods    1 10
secrets   9 20
```

## 13.4. SELECTION GRANULARITY

Due to the locking consideration when claiming quota allocations, the number of active projects selected by a multi-project quota is an important consideration. Selecting more than 100 projects under a single multi-project quota may have detrimental effects on API server responsiveness in those projects.

# CHAPTER 14. SETTING LIMIT RANGES

## 14.1. OVERVIEW

A limit range, defined by a **`LimitRange`** object, enumerates compute resource constraints in a project at the pod, container, image, image stream, and persistent volume claim level, and specifies the amount of resources that a pod, container, image, image stream, or persistent volume claim can consume.

All resource create and modification requests are evaluated against each **`LimitRange`** object in the project. If the resource violates any of the enumerated constraints, then the resource is rejected. If the resource does not set an explicit value, and if the constraint supports a default value, then the default value is applied to the resource.

**Example 14.1. Limit Range Object Definition**

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

1

The name of the limit range document.

2

The maximum amount of CPU that a pod can request on a node across all containers.

3

The maximum amount of memory that a pod can request on a node across all containers.

4

The minimum amount of CPU that a pod can request on a node across all containers.

5

The minimum amount of memory that a pod can request on a node across all containers.

6

The maximum amount of CPU that a single container in a pod can request.

7

The maximum amount of memory that a single container in a pod can request.

8

The minimum amount of CPU that a single container in a pod can request.

9

The minimum amount of memory that a single container in a pod can request.

10

The default amount of CPU that a container will be limited to use if not specified.

11

The default amount of memory that a container will be limited to use if not specified.

12

The default amount of CPU that a container will request to use if not specified.

13

The default amount of memory that a container will request to use if not specified.

14

The maximum amount of CPU burst that a container can make as a ratio of its limit over request.

**Example 14.2. OpenShift Container Platform Limit Range Object Definition**

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "openshift-resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi 1
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 2
        openshift.io/images: 30 3
```

1

The maximum size of an image that can be pushed to an internal registry.

2

The maximum number of unique image tags per image stream’s spec.

3

The maximum number of unique image references per image stream’s status.

Both core and OpenShift Container Platform resources can be specified in just one limit range object. They are separated here into two examples for clarity.

### 14.1.1. Container Limits

**Supported Resources:**

- CPU
- Memory

**Supported Constraints:**

Per container, the following must hold true if specified:

**Table 14.1. Container**

| Constraint | Behavior |
|---|---|
| `Min` | `Min[resource]` less than or equal to `container.resources.requests[resource]` (required) less than or equal to `container/resources.limits[resource]` (optional)<br><br>If the configuration defines a `min` CPU, then the request value must be greater than the CPU value. A limit value does not need to be specified. |
| `Max` | `container.resources.limits[resource]` (required) less than or equal to `Max[resource]`<br><br>If the configuration defines a `max` CPU, then you do not need to define a request value, but a limit value does need to be set that satisfies the maximum CPU constraint. |
| `MaxLimitRequestRatio` | `MaxLimitRequestRatio[resource]` less than or equal to ( `container.resources.limits[resource]` / `container.resources.requests[resource]`)<br><br>If a configuration defines a `maxLimitRequestRatio` value, then any new containers must have both a request and limit value. Additionally, OpenShift Container Platform calculates a limit to request ratio by dividing the limit by the request.<br><br>For example, if a container has `cpu: 500` in the `limit` value, and `cpu: 100` in the `request` value, then its limit to request ratio for `cpu` is `5`. This ratio must be less than or equal to the `maxLimitRequestRatio`. |

**Supported Defaults:**

**`Default[resource]`**

Defaults `container.resources.limit[resource]` to specified value if none.

**`Default Requests[resource]`**

Defaults `container.resources.requests[resource]` to specified value if none.

### 14.1.2. Pod Limits

**Supported Resources:**

- CPU
- Memory

**Supported Constraints:**

Across all containers in a pod, the following must hold true:

Table 14.2. Pod

| Constraint | Enforced Behavior |
|---|---|
| **`Min`** | **`Min[resource]`** less than or equal to **`container.resources.requests[resource]`** (required) less than or equal to **`container.resources.limits[resource]`** (optional) |
| **`Max`** | **`container.resources.limits[resource]`** (required) less than or equal to **`Max[resource]`** |
| **`MaxLimitRequestRatio`** | **`MaxLimitRequestRatio[resource]`** less than or equal to ( **`container.resources.limits[resource]`** / **`container.resources.requests[resource]`**) |

### 14.1.3. Image Limits

**Supported Resources:**

- Storage

**Resource type name:**

- **`openshift.io/Image`**

Per image, the following must hold true if specified:

Table 14.3. Image

| Constraint | Behavior |
|---|---|
| **`Max`** | **`image.dockerimagemetadata.size`** less than or equal to **`Max[resource]`** |

**Note**

To prevent blobs exceeding the limit from being uploaded to the registry, the registry must be configured to enforce quota. An environment variable **`REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA`** must be set to **`true`** which is done by default for new deployments. To update older deployment configuration, refer to Enforcing quota in the Registry.

**Warning**

The image size is not always available in the manifest of an uploaded image. This is especially the case for images built with Docker 1.10 or higher and pushed to a v2 registry. If such an image is pulled with an older Docker daemon, the image manifest will be converted by the registry to schema v1 lacking all the size information. No storage limit set on images will prevent it from being uploaded.

The issue is being addressed.

### 14.1.4. Image Stream Limits

**Supported Resources:**

- **`openshift.io/image-tags`**
- **`openshift.io/images`**

**Resource type name:**

- **`openshift.io/ImageStream`**

Per image stream, the following must hold true if specified:

**Table 14.4. ImageStream**

| Constraint | Behavior |
|---|---|
| **`Max[openshift.io/image-tags]`** | **`length( uniqueimagetags( imagestream.spec.tags ) )`** less than or equal to **`Max[openshift.io/image-tags]`**<br><br>**`uniqueimagetags`** returns unique references to images of given spec tags. |
| **`Max[openshift.io/images]`** | **`length( uniqueimages( imagestream.status.tags ) )`** less than or equal to **`Max[openshift.io/images]`**<br><br>**`uniqueimages`** returns unique image names found in status tags. The name equals image’s digest. |

#### 14.1.4.1. Counting of Image References

Resource **`openshift.io/image-tags`** represents unique image references. Possible references are an **`ImageStreamTag`**, an **`ImageStreamImage`** and a **`DockerImage`**. They may be created using commands **`oc tag`** and **`oc import-image`** or by using tag tracking. No distinction is made between internal and external references. However, each unique reference tagged in the image stream's specification is counted just once. It does not restrict pushes to an internal container registry in any way, but is useful for tag restriction.

Resource **`openshift.io/images`** represents unique image names recorded in image stream status. It allows for restriction of a number of images that can be pushed to the internal registry. Internal and external references are not distinguished.

### 14.1.5. PersistentVolumeClaim Limits

**Supported Resources:**

- Storage

**Supported Constraints:**

Across all persistent volume claims in a project, the following must hold true:

**Table 14.5. Pod**

| Constraint | Enforced Behavior |
|---|---|
| **`Min`** | Min[resource] <= claim.spec.resources.requests[resource] (required) |
| **`Max`** | claim.spec.resources.requests[resource] (required) <= Max[resource] |

**Example 14.3. Limit Range Object Definition**

```
{
  "apiVersion": "v1",
  "kind": "LimitRange",
  "metadata": {
    "name": "pvcs" (1)
  },
  "spec": {
    "limits": [{
        "type": "PersistentVolumeClaim",
        "min": {
          "storage": "2Gi" (2)
        },
        "max": {
          "storage": "50Gi" (3)
        }
        }
      ]
    }
  }
```

1

The name of the limit range document.

2

The minimum amount of storage that can be requested in a persistent volume claim

3

The maximum amount of storage that can be requested in a persistent volume claim

## 14.2. CREATING A LIMIT RANGE

To apply a limit range to a project, create a limit range object definition on your file system to your desired specifications, then run:

```
$ oc create -f <limit_range_file> -n <project>
```

## 14.3. VIEWING LIMITS

You can view any limit ranges defined in a project by navigating in the web console to the project’s **Settings** tab.

You can also use the CLI to view limit range details:

1. First, get the list of limit ranges defined in the project. For example, for a project called **demoproject**:

```
$ oc get limits -n demoproject
NAME              AGE
resource-limits   6d
```

2. Then, describe the limit range you are interested in, for example the **resource-limits** limit range:

```
$ oc describe limits resource-limits
Name:                          limits
Namespace:                     default
Type                           Resource                 Min   Max
Request Limit Limit/Request
```

```
----                      --------                 ---  --- --
----- ----- ------------
Pod                       memory                   6Mi  1Gi -
-     -
Pod                       cpu                      200m  2  -
-     -
Container                 cpu                      100m  2
200m    300m  10
Container                 memory                   4Mi  1Gi
100Mi   200Mi -
openshift.io/Image        storage                  -    1Gi -
-     -
openshift.io/ImageStream  openshift.io/image-tags  -    10  -
-     -
openshift.io/ImageStream  openshift.io/images      -    12  -
-     -
```

## 14.4. DELETING LIMITS

Remove any active limit range to no longer enforce the limits of a project:

```
$ oc delete limits <limit_name>
```

# CHAPTER 15. PRUNING OBJECTS

## 15.1. OVERVIEW

Over time, API objects created in OpenShift Container Platform can accumulate in the etcd data store through normal user operations, such as when building and deploying applications.

As an administrator, you can periodically prune older versions of objects from your OpenShift Container Platform instance that are no longer needed. For example, by pruning images you can delete older images and layers that are no longer in use, but are still taking up disk space.

## 15.2. 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.

## 15.3. 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 15.1. 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`**) |

| Option | Description |
|---|---|
| **`--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
```

## 15.4. 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 15.2. 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
```

## 15.5. PRUNING IMAGES

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

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

**Note**

Pruning images removes data from the integrated registry. For this operation to work properly, ensure your registry is configured with **`storage:delete:enabled`** set to **true**.

**Note**

Pruning images with the **`--namespace`** flag does not remove images, only image streams. Images are non-namespaced resources. Therefore, limiting pruning to a particular namespace makes it impossible to calculate their current usage.

**Table 15.3. Prune Images CLI Configuration Options**

| Option | Description |
|---|---|
| **`--certificate-authority`** | The path to a certificate authority file to use when communicating with the OpenShift Container Platform-managed registries. Defaults to the certificate authority data from the current user's config file. |
| **`--confirm`** | Indicate that pruning should occur, instead of performing a dry-run. |

| Option | Description |
|---|---|
| `--keep-tag-revisions=<N>` | For each image stream, keep up to at most N image revisions per tag. (default **3**) |
| `--keep-younger-than=<duration>` | Do not prune any image that is younger than `<duration>` relative to the current time. Do not prune any image that is referenced by any other object that is younger than `<duration>` relative to the current time. (default **60m**) |
| `--prune-over-size-limit` | Prune each image that exceeds the smallest limit defined in the same project. This flag cannot be combined with `--keep-tag-revisions` nor `--keep-younger-than`. |

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

**Image Prune Conditions**

- Remove any image "managed by OpenShift Container Platform" (images with the annotation `openshift.io/image.managed`) that was created at least `--keep-younger-than` minutes ago and is not currently referenced by:
  - any pod created less than `--keep-younger-than` minutes ago.
  - any image stream created less than `--keep-younger-than` minutes ago.
  - any running pods.
  - any pending pods.
  - any replication controllers.
  - any deployment configurations.
  - any build configurations.
  - any builds.
  - the `--keep-tag-revisions` most recent items in `stream.status.tags[].items`.
- Remove any image "managed by OpenShift Container Platform" (images with the annotation `openshift.io/image.managed`) that is exceeding the smallest limit defined in the same project and is not currently referenced by:
  - any running pods.
  - any pending pods.
  - any replication controllers.
  - any deployment configurations.
  - any build configurations.

  - any builds.
- There is no support for pruning from external registries.
- When an image is pruned, all references to the image are removed from all image streams that have a reference to the image in **`status.tags`**.
- Image layers that are no longer referenced by any images are removed as well.

**Note**

**`--prune-over-size-limit`** cannot be combined with **`--keep-tag-revisions`** nor **`--keep-younger-than`** flags. Doing so will return an information that this operation is not allowed.

To see what a pruning operation would delete:

1. Keeping up to three tag revisions, and keeping resources (images, image streams and pods) younger than sixty minutes:

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

2. Pruning every image that exceeds defined limits:

   ```
   $ oadm prune images --prune-over-size-limit
   ```

To actually perform the prune operation for the previously mentioned options accordingly:

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

$ oadm prune images --prune-over-size-limit --confirm
```

### 15.5.1. Image Pruning Problems

If your images keep accumulating and the **`prune`** command removes just a small portion of what you expect, ensure that you understand the conditions that must apply for an image to be considered a candidate for pruning.

Especially ensure that images you want removed occur at higher positions in each tag history than your chosen tag revisions threshold. For example, consider an old and obsolete image named **`sha:abz`**. By running the following command in namespace **`N`**, where the image is tagged, you will see the image is tagged three times in a single image stream named **`myapp`**:

```
$ image_name="sha:abz"
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range
$ti, $tag := $is.status.tags}}'\
  '{{range $ii, $item := $tag.items}}{{if eq $item.image
"'"${image_name}"'\
  $'"}}{{$is.metadata.name}}:{{$tag.tag}} at position {{$ii}} out of
```

```
{{len $tag.items}}\n'\
  '{{end}}{{end}}{{end}}{{end}}'
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

When default options are used, the image will not ever be pruned because it occurs at position **`0`** in a history of **`myapp:v2.1-may-2016`** tag. For an image to be considered for pruning, the administrator must either:

1. Specify **`--keep-tag-revisions=0`** with the **`oadm prune images`** command.

   **Caution**

   This action will effectively remove all the tags from all the namespaces with underlying images, unless they are younger or they are referenced by objects younger than the specified threshold.

2. Delete all the *istags* where the position is below the revision threshold, which means **`myapp:v2.1`** and **`myapp:v2.1-may-2016`**.
3. Move the image further in the history, either by running new builds pushing to the same *istag*, or by tagging other image. Unfortunately, this is not always desirable for old release tags.

Tags having a date or time of a particular image's build in their names should be avoided, unless the image needs to be preserved for undefined amount of time. Such tags tend to have just one image in its history, which effectively prevents them from ever being pruned. Learn more about *istag* naming.

# CHAPTER 16. GARBAGE COLLECTION

## 16.1. OVERVIEW

The OpenShift Container Platform node performs two types of garbage collection:

- Container garbage collection: Removes terminated containers. Typically run every minute.
- Image garbage collection: Removes images not referenced by any running pods. Typically run every five minutes.

## 16.2. CONTAINER GARBAGE COLLECTION

The policy for container garbage collection is based on three node settings:

| Setting | Description |
|---|---|
| **`minimum-container-ttl-duration`** | The minimum age that a container is eligible for garbage collection. The default is **`1m`** (one minute). Use **`0`** for no limit. Values for this setting can be specified using unit suffixes such as **`h`** for hour, **`m`** for minutes, **`s`** for seconds. |
| **`maximum-dead-containers-per-container`** | The number of instances to retain per pod container. The default is **`2`**. |
| **`maximum-dead-containers`** | The maximum number of total dead containers in the node. The default is **`100`**. |

The **`maximum-dead-containers`** setting takes precedence over the **`maximum-dead-containers-per-container`** setting when there is a conflict. For example, if retaining the number of **`maximum-dead-containers-per-container`** would result in a total number of containers that is greater than **`maximum-dead-containers`**, the oldest containers will be removed to satisfy the **`maximum-dead-containers`** limit.

When the node removes the dead containers, all files inside those containers are removed as well. Only containers created by the node will be garbage collected.

You can specify values for these settings in the **`kubeletArguments`** section of the ***/etc/origin/node/node-config.yaml*** file on node hosts. Add the section if it does not already exist:

### Container Garbage Collection Settings

```
kubeletArguments:
  minimum-container-ttl-duration:
    - "10s"
  maximum-dead-containers-per-container:
    - "2"
  maximum-dead-containers:
    - "100"
```

### 16.2.1. Detecting Containers for Deletion

Each spin of the garbage collector loop goes through the following steps:

1. Retrieve a list of available containers.
2. Filter out all containers that are running or are not alive longer than the **`minimum-container-ttl-duration`** parameter.
3. Classify all remaining containers into equivalence classes based on pod and image name membership.
4. Remove all unidentified containers (containers that are managed by kubelet but their name is malformed).
5. For each class that contains more containers than the **`maximum-dead-containers-per-container`** parameter, sort containers in the class by creation time.
6. Start removing containers from the oldest first until the **`maximum-dead-containers-per-container`** parameter is met.
7. If there are still more containers in the list than the **`maximum-dead-containers`** parameter, the collector starts removing containers from each class so the number of containers in each one is not greater than the average number of containers per class, or **`<all_remaining_containers>/<number_of_classes>`**.
8. If this is still not enough, sort all containers in the list and start removing containers from the oldest first until the **`maximum-dead-containers`** criterion is met.

## 16.3. IMAGE GARBAGE COLLECTION

Image garbage collection relies on disk usage as reported by **cAdvisor** on the node to decide which images to remove from the node. It takes the following settings into consideration:

| Setting | Description |
|---|---|
| **`image-gc-high-threshold`** | The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is **90**. |
| **`image-gc-low-threshold`** | The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is **80**. |

You can specify values for these settings in the **`kubeletArguments`** section of the ***/etc/origin/node/node-config.yaml*** file on node hosts. Add the section if it does not already exist:

**Image Garbage Collection Settings**

```
kubeletArguments:
  image-gc-high-threshold:
    - "90"
  image-gc-low-threshold:
    - "80"
```

### 16.3.1. Detecting Images for Deletion

Two lists of images are retrieved in each garbage collector run:

1. A list of images currently running in at least one pod
2. A list of images available on a host

As new containers are run, new images appear. All images are marked with a time stamp. If the image is running (the first list above) or is newly detected (the second list above), it is marked with the current time. The remaining images are already marked from the previous spins. All images are then sorted by the time stamp.

Once the collection starts, the oldest images get deleted first until the stopping criterion is met.

# CHAPTER 17. SCHEDULER

## 17.1. 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.

## 17.2. 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

### 17.2.1. 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'.

### 17.2.2. 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.

### 17.2.3. 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.

## 17.3. 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.

### 17.3.1. 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"}
```

### 17.3.2. 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.

## 17.4. 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.

### 17.4.1. 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}
```

### 17.4.2. 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"}}}
```

## 17.5. 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.

### 17.5.1. 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

### 17.5.2. Modifying Scheduler Policy

The scheduler policy is defined in a file on the master, named ***/etc/origin/master/scheduler.json*** by default, unless overridden by the **`kubernetesMasterConfig.schedulerConfigFile`** field in the master configuration file.

To modify the scheduler policy:

1. Edit the scheduler configuration file to set the desired predicates and priority functions. You can create a custom configuration, or modify one of the sample policy configurations.
2. Restart the OpenShift Container Platform master services for the changes to take effect.

## 17.6. USE CASES

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

### 17.6.1. Infrastructure Topological Levels

Administrators can define multiple topological levels for their infrastructure (nodes). This is done by specifying labels on nodes (e.g., **`region=r1`**, **`zone=z1`**, **`rack=s1`**). These label names have no particular meaning and administrators are free to name their infrastructure levels anything (eg, city/building/room). Also, administrators can define any number of levels for their infrastructure topology, with three levels usually being adequate (eg. regions → zones → racks). Lastly, administrators can specify affinity and anti-affinity rules at each of these levels in any combination.

### 17.6.2. 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.

### 17.6.3. 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.

## 17.7. 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
```

## 17.8. SCHEDULER EXTENSIBILITY

As is the case with almost everything else in Kubernetes/OpenShift Container Platform, 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

### 17.8.1. 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.

### 17.8.2. 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.

## 17.9. CONTROLLING POD PLACEMENT

As a cluster administrator, you can set a policy to prevent application developers with certain roles from targeting specific nodes when scheduling pods.

**Important**

This process involves the **pods/binding** permission role, which is needed to target particular nodes. The constraint on the use of the **`nodeSelector`** field of a pod configuration is based on the **pods/binding** permission and the **`nodeSelectorLabelBlacklist`** configuration option.

The **`nodeSelectorLabelBlacklist`** field of a master configuration file gives you control over the labels that certain roles can specify in a pod configuration’s **`nodeSelector`** field. Users, service accounts, and groups that have the **pods/binding** permission can specify any node selector. Those without the **pods/binding** permission are prohibited from setting a **`nodeSelector`** for any label that appears in **`nodeSelectorLabelBlacklist`**.

As a hypothetical example, an OpenShift Container Platform cluster might consist of five data centers spread across two regions. In the U.S., "us-east", "us-central", and "us-west"; and in the Asia-Pacific region (APAC), "apac-east" and "apac-west". Each node in each geographical region is labeled accordingly. For example, **`region: us-east`**.

**Note**

See Updating Labels on Nodes for details on assigning labels.

As a cluster administrator, you can create an infrastructure where application developers should be deploying pods only onto the nodes closest to their geographical location. You can create a node selector, grouping the U.S. data centers into **`superregion: us`** and the APAC data centers into **`superregion: apac`**.

To maintain an even loading of resources per data center, you can add the desired **`region`** to the **`nodeSelectorLabelBlacklist`** section of a master configuration. Then, whenever a developer located in the U.S. creates a pod, it is deployed onto a node in one of the regions with the **`superregion: us`** label. If the developer tries to target a specific region for their pod (for example, **`region: us-east`**), they will receive an error. If they try again, without the node selector on their pod, it can still be deployed onto the region they tried to target, because **`superregion: us`** is set as the project-level node selector, and nodes labeled **`region: us-east`** are also labeled **`superregion: us`**.

### 17.9.1. Constraining Pod Placement Using Node Name

Ensure a pod is deployed onto only a specified node host by assigning it a label and specifying this in the **`nodeName`** setting in a pod configuration.

1. Ensure you have the desired labels and node selector set up in your environment.

   For example, make sure that your pod configuration features the **`nodeName`** value indicating the desired label:

   ```
   apiVersion: v1
   kind: Pod
   spec:
     nodeName: <key: value>
   ```

2. Modify the master configuration file (***/etc/origin/master/master-config.yaml***) in two places:

   a. Add **`nodeSelectorLabelBlacklist`** to the **`admissionConfig`** section:

      ```
      ...
      admissionConfig:
        pluginConfig:
          PodNodeConstraints:
            configuration:
              apiversion: v1
              kind: PodNodeConstraintsConfig
      ...
      ```

   b. Then, add the same to the **`kubernetesMasterConfig`** section to restrict direct pod creation:

      ```
      ...
      ```

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
...
```

3. Restart OpenShift Container Platform for the changes to take effect.

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

### 17.9.2. Constraining Pod Placement Using a Node Selector

Using **`nodeSelector`** in a pod configuration, you can ensure that pods are only placed onto nodes with specific labels.

1. Ensure you have the desired labels (see Updating Labels on Nodes for details) and node selector set up in your environment.

   For example, make sure that your pod configuration features the **`nodeSelector`** value indicating the desired label:

   ```
   apiVersion: v1
   kind: Pod
   spec:
     nodeSelector:
       <key>: <value>
   ...
   ```

2. Modify the master configuration file (***/etc/origin/master/master-config.yaml***) in two places:

   a. Add **`nodeSelectorLabelBlacklist`** to the **`admissionConfig`** section with the labels that are assigned to the node hosts you want to deny pod placement:

   ```
   ...
   admissionConfig:
     pluginConfig:
       PodNodeConstraints:
         configuration:
           apiversion: v1
           kind: PodNodeConstraintsConfig
           nodeSelectorLabelBlacklist:
             - kubernetes.io/hostname
             - <label>
   ...
   ```

   b. Then, add the same to the **`kubernetesMasterConfig`** section to restrict direct pod creation:

   ```
   ...
   ```

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
          nodeSelectorLabelBlacklist:
            - kubernetes.io/hostname
            - <label_1>
...
```

3. Restart OpenShift Container Platform for the changes to take effect.

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

# CHAPTER 18. ALLOCATING NODE RESOURCES

## 18.1. OVERVIEW

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by all underlying node components (e.g., kubelet, kube-proxy, Docker) and the remaining system components (e.g., **sshd**, **NetworkManager**) on the host. Once specified, the scheduler has more information about the resources (e.g., memory, CPU) a node has allocated for pods.

## 18.2. CONFIGURING NODES FOR ALLOCATED RESOURCES

Resources reserved for node components are based on two node settings:

| Setting | Description |
|---|---|
| **`kube-reserved`** | Resources reserved for node components. Default is none. |
| **`system-reserved`** | Resources reserved for the remaining system components. Default is none. |

You can set these in the **`kubeletArguments`** section of the node configuration file (the ***/etc/origin/node/node-config.yaml*** file by default) using a set of **`<resource_type>=<resource_quantity>`** pairs (e.g., **cpu=200m,memory=30G**). Add the section if it does not already exist:

**Example 18.1. Node Allocatable Resources Settings**

```
kubeletArguments:
  kube-reserved:
    - "cpu=200m,memory=30G"
  system-reserved:
    - "cpu=200m,memory=30G"
```

Currently, the **`cpu`** and **`memory`** resource types are supported. For **`cpu`**, the resource quantity is specified in units of cores (e.g., 200m, 100Ki, 50M). For **`memory`**, it is specified in units of bytes (e.g., 200Ki, 100M, 50Gi).

See Compute Resources for more details.

If a flag is not set, it defaults to **`0`**. If none of the flags are set, the allocated resource is set to the node's capacity as it was before the introduction of allocatable resources.

## 18.3. COMPUTING ALLOCATED RESOURCES

An allocated amount of a resource is computed based on the following formula:

```
[Allocatable] = [Node Capacity] - [kube-reserved] - [system-reserved]
```

If **`[Allocatable]`** is negative, it is set to **`0`**.

## 18.4. VIEWING NODE ALLOCATABLE RESOURCES AND CAPACITY

To see a node’s current capacity and allocatable resources, you can run:

```
$ oc get node/<node_name> -o yaml
...
status:
...
  allocatable:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
  capacity:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
...
```

## 18.5. SYSTEM RESOURCES REPORTED BY NODE

Starting with OpenShift Container Platform 3.3, each node reports system resources utilized by the container runtime and kubelet. To better aid your ability to configure **`--system-reserved`** and **`--kube-reserved`**, you can introspect corresponding node’s resource usage using the node summary API, which is accessible at ***<master>/api/v1/nodes/<node>/proxy/stats/summary***.

For instance, to access the resources from **cluster.node22** node, you can run:

```
$ curl <certificate details>
https://<master>/api/v1/nodes/cluster.node22/proxy/stats/summary
{
    "node": {
        "nodeName": "cluster.node22",
        "systemContainers": [
            {
                "cpu": {
                    "usageCoreNanoSeconds": 929684480915,
                    "usageNanoCores": 190998084
                },
                "memory": {
                    "rssBytes": 176726016,
                    "usageBytes": 1397895168,
                    "workingSetBytes": 1050509312
                },
                "name": "kubelet"
            },
            {
                "cpu": {
                    "usageCoreNanoSeconds": 128521955903,
                    "usageNanoCores": 5928600
                },
                "memory": {
                    "rssBytes": 35958784,
                    "usageBytes": 129671168,
                    "workingSetBytes": 102416384
                },
                "name": "runtime"
            }
        ]
    }
}
```

See REST API Overview for more details about certificate details.

## 18.6. SCHEDULER

The scheduler now uses the value of **`node.Status.Allocatable`** instead of **`node.Status.Capacity`** to decide if a node will become a candidate for pod scheduling.

By default, the node will report its machine capacity as fully schedulable by the cluster.

# CHAPTER 19. OVERCOMMITTING

## 19.1. OVERVIEW

Containers can specify compute resource requests and limits. Requests are used for scheduling your container and provide a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

The scheduler attempts to improve the utilization of compute resources across all nodes in the cluster. It places pods on nodes relative to the pods' compute resource requests to find a node that provides the best fit.

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node, which may be desirable in development environments where performance is not a concern.

## 19.2. REQUESTS AND LIMITS

For each compute resource, a container may specify a resource request and limit. Scheduling decisions are made based on the request to ensure that a node has enough capacity available to meet the requested value. If a container specifies limits, but omits requests, the requests are defaulted to the limits. A container is not able to exceed the specified limit on the node.

The enforcement of limits is dependent upon the compute resource type. If a container makes no request or limit, the container is scheduled to a node with no resource guarantees. In practice, the container is able to consume as much of the specified resource as is available with the lowest local priority. In low resource situations, containers that specify no resource requests are given the lowest quality of service.

## 19.3. COMPUTE RESOURCES

The node-enforced behavior for compute resources is specific to the resource type.

### 19.3.1. CPU

A container is guaranteed the amount of CPU it requests, but it may or may not get more CPU time based on local node conditions. If a container does not specify a corresponding limit, it is able to consume excess CPU available on the node. If multiple containers are attempting to use excess CPU, CPU time is distributed based on the amount of CPU requested by each container.

For example, if one container requested 500m of CPU time, and another container requested 250m of CPU time, any extra CPU time available on the node is distributed among the containers in a 2:1 ratio. If a container specified a limit, it will be throttled to not use more CPU than the specified limit.

CPU requests are enforced using the CFS shares support in the Linux kernel. By default, CPU limits are enforced using the CFS quota support in the Linux kernel over a 100ms measuring interval, though this can be disabled.

### 19.3.2. Memory

A container is guaranteed the amount of memory it requests. A container may use more memory than requested, but once it exceeds its requested amount, it could be killed in a low memory situation on the node.

If a container uses less memory than requested, it will not be killed unless system tasks or daemons need more memory than was accounted for in the node’s resource reservation. If a container specifies a limit on memory, it is immediately killed if it exceeds the limited amount.

## 19.4. QUALITY OF SERVICE CLASSES

A node is *overcommitted* when it has a pod scheduled that makes no request, or when the sum of limits across all pods on that node exceeds available machine capacity.

In an overcommitted environment, it is possible that the pods on the node will attempt to use more compute resource than is available at any given point in time. When this occurs, the node must give priority to one pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) Class.

For each compute resource, a container is divided into one of three QoS classes with decreasing order of priority:

**Table 19.1. Quality of Service Classes**

| Priority | Class Name | Description |
|---|---|---|
| 1 (highest) | **Guaranteed** | If limits and optionally requests are set (not equal to 0) for all resources and they are equal, then the container is classified as **Guaranteed**. |
| 2 | **Burstable** | If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the container is classified as **Burstable**. |
| 3 (lowest) | **BestEffort** | If requests and limits are not set for any of the resources, then the container is classified as **BestEffort**. |

Memory is an incompressible resource, so in low memory situations, containers are killed that have the lowest priority:

- **Guaranteed** containers are considered top priority, and are guaranteed to only be killed if they exceed their limits, or if the system is under memory pressure and there are no lower priority containers that can be evicted.
- **Burstable** containers under system memory pressure are more likely to be killed once they exceed their requests and no other **BestEffort** containers exist.
- **BestEffort** containers are treated with the lowest priority. Processes in these containers are first to be killed if the system runs out of memory.

## 19.5. CONFIGURING MASTERS FOR OVERCOMMITMENT

Scheduling is based on resources requested, while quota and hard limits refer to resource limits, which can be set higher than requested resources. The difference between request and limit determines the level of overcommit; for instance, if a container is given a memory request of 1Gi and a memory limit of 2Gi, it is scheduled based on the 1Gi request being available on the node, but could use up to 2Gi; so it is 200% overcommitted.

If OpenShift Container Platform administrators would like to control the level of overcommit and manage container density on nodes, masters can be configured to override the ratio between request and limit set on developer containers. In conjunction with a per-project LimitRange specifying limits and defaults, this adjusts the container limit and request to achieve the desired level of overcommit.

This requires configuring the **`ClusterResourceOverride`** admission controller in the ***master-config.yaml*** as in the following example (reuse the existing configuration tree if it exists, or introduce absent elements as needed):

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      ClusterResourceOverride:    1
        configuration:
          apiVersion: v1
          kind: ClusterResourceOverrideConfig
          memoryRequestToLimitPercent: 25  2
          cpuRequestToLimitPercent: 25     3
          limitCPUToMemoryPercent: 200     4
```

1

This is the plug-in name; case matters and anything but an exact match for a plug-in name is ignored.

2

(optional, 1-100) If a container memory limit has been specified or defaulted, the memory request is overridden to this percentage of the limit.

3

(optional, 1-100) If a container CPU limit has been specified or defaulted, the CPU request is overridden to this percentage of the limit.

4

(optional, positive integer) If a container memory limit has been specified or defaulted, the CPU limit is overridden to a percentage of the memory limit, with a 100 percentage scaling 1Gi of RAM to equal 1 CPU core. This is processed prior to overriding CPU request (if configured).

After changing the master configuration, a master restart is required.

Note that these overrides have no effect if no limits have been set on containers. Create a LimitRange object with default limits (per individual project, or in the project template) in order to ensure that the overrides apply.

Note also that after overrides, the container limits and requests must still be validated by any LimitRange objects in the project. It is possible, for example, for developers to specify a limit close to the minimum limit, and have the request then be overridden below the minimum limit, causing the pod to be forbidden. This unfortunate user experience should be addressed with future work, but for now, configure this capability and LimitRanges with caution.

When configured, overrides can be disabled per-project (for example, to allow infrastructure components to be configured independently of overrides) by editing the project and adding the following annotation:

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

## 19.6. CONFIGURING NODES FOR OVERCOMMITMENT

In an overcommitted environment, it is important to properly configure your node to provide best system behavior.

### 19.6.1. Enforcing CPU Limits

Nodes by default enforce specified CPU limits using the CPU CFS quota support in the Linux kernel. If you do not want to enforce CPU limits on the node, you can disable its enforcement by modifying the node configuration file (the ***node-config.yaml*** file) to include the following:

```
kubeletArguments:
  cpu-cfs-quota:
    - "false"
```

If CPU limit enforcement is disabled, it is important to understand the impact that will have on your node:

- If a container makes a request for CPU, it will continue to be enforced by CFS shares in the Linux kernel.
- If a container makes no explicit request for CPU, but it does specify a limit, the request will default to the specified limit, and be enforced by CFS shares in the Linux kernel.
- If a container specifies both a request and a limit for CPU, the request will be enforced by CFS shares in the Linux kernel, and the limit will have no impact on the node.

### 19.6.2. Reserving Resources for System Processes

The scheduler ensures that there are enough resources for all pods on a node based on the pod requests. It verifies that the sum of requests of containers on the node is no greater than the node capacity. It includes all containers started by the node, but not containers or processes started outside the knowledge of the cluster.

It is recommended that you reserve some portion of the node capacity to allow for the system daemons that are required to run on your node for your cluster to function (**sshd**, **docker**, etc.). In particular, it is recommended that you reserve resources for incompressible resources such as memory.

If you want to explicitly reserve resources for non-pod processes, there are two ways to do so:

- The preferred method is to allocate node resources by specifying resources available for scheduling. See Allocating Node Resources for more details.
- Alternatively, you can create a **resource-reserver** pod that does nothing but reserve capacity from being scheduled on the node by the cluster. For example:

**Example 19.1. resource-reserver Pod Definition**

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-reserver
spec:
  containers:
  - name: sleep-forever
    image: gcr.io/google_containers/pause:0.8.0
    resources:
      limits:
        cpu: 100m (1)
        memory: 150Mi (2)
```

1. The amount of CPU to reserve on a node for host-level daemons unknown to the cluster.
2. The amount of memory to reserve on a node for host-level daemons unknown to the cluster.

You can save your definition to a file, for example ***resource-reserver.yaml***, then place the file in the node configuration directory, for example ***/etc/origin/node/*** or the **`--config=<dir>`** location if otherwise specified.

Additionally, the node server needs to be configured to read the definition from the node configuration directory, by naming the directory in the **`kubeletArguments.config`** field of the node configuration file (usually named ***node-config.yaml***):

```
kubeletArguments:
  config:
    - "/etc/origin/node" (1)
```

1.

If **`--config=<dir>`** is specified, use **`<dir>`** here.

With the ***resource-reserver.yaml*** file in place, starting the node server also launches the **sleep-forever** container. The scheduler takes into account the remaining capacity of the node, adjusting where to place cluster pods accordingly.

To remove the **resource-reserver** pod, you can delete or move the ***resource-reserver.yaml*** file from the node configuration directory.

### 19.6.3. Kernel Tunable Flags

When the node starts, it ensures that the kernel tunable flags for memory management are set properly. The kernel should never fail memory allocations unless it runs out of physical memory.

To ensure this behavior, the node instructs the kernel to always overcommit memory:

```
$ sysctl -w vm.overcommit_memory=1
```

The node also instructs the kernel not to panic when it runs out of memory. Instead, the kernel OOM killer should kill processes based on priority:

```
$ sysctl -w vm.panic_on_oom=0
```

**Note**

The above flags should already be set on nodes, and no further action is required.

### 19.6.4. Disabling Swap Memory

You can disable swap by default on your nodes in order to preserve quality of service guarantees. Otherwise, physical resources on a node can oversubscribe, affecting the resource guarantees the Kubernetes scheduler makes during pod placement.

For example, if two guaranteed pods have reached their memory limit, each container could start using swap memory. Eventually, if there is not enough swap space, processes in the pods can be terminated due to the system being oversubscribed.

To disable swap:

```
$ swapoff -a
```

Failing to disable swap results in nodes not recognizing that they are experiencing **MemoryPressure**, resulting in pods not receiving the memory they made in their scheduling request. As a result, additional pods are placed on the node to further increase memory pressure, ultimately increasing your risk of experiencing a system out of memory (OOM) event.

**Important**

If swap is enabled, any out of resource handling eviction thresholds for available memory will not work as expected. Take advantage of out of resource handling to allow pods to be evicted from a node when it is under memory pressure, and rescheduled on an alternative node that has no such pressure.

# CHAPTER 20. ASSIGNING UNIQUE EXTERNAL IPS FOR INGRESS TRAFFIC

## 20.1. OVERVIEW

**Note**

This feature is only supported in non-cloud deployments. For cloud (GCE, AWS, and OpenStack) deployments, load Balancer services can be used to automatically deploy a cloud load balancer to target the service's endpoints.

Cluster administrators can assign a unique external IP address to a service. If routed correctly, external traffic can reach that service's endpoints via any TCP/UDP port the service exposes. This can be simpler than having to manage the port space of a limited number of shared IP addresses when manually assigning external IPs to services.

There is support for both automatic and manual assignment of IP addresses, and each address is guaranteed to be assigned to a maximum of one service. This ensures that each service can simply expose its chosen ports regardless of the ports exposed by other services.

## 20.2. RESTRICTIONS

To use an **ExternalIP**, you can:

- Select an IP address from the **ExternalIPNetworkCIDRs** range.
- Have an IP address assigned from a pool. In this case, OpenShift Container Platform implements a non-cloud version of the LoadBalancer service type and assigns IP addresses to the services.

**Caution**

You must ensure that the IP address pool you assign terminates at one or more nodes in your cluster. You can use the existing `oadm ipfailover` to ensure that the external IPs are highly available.

For manually-configured external IPs, potential port clashes are handled on a first-come, first-served basis. If you request a port, it is only available if it has not yet been assigned for that IP address. For example:

**Example 20.1. Port clash example for manually-configured external IPs**

Two services have been manually configured with the same external IP address of 172.7.7.7.

**`MongoDB service A`** requests port 27017, and then **`MongoDB service B`** requests the same port; the first request gets the port.

However, port clashes are not an issue for external IPs assigned by the ingress controller, because the controller assigns each service a unique address.

**Note**

Ingress IPs can only be assigned if the cluster is not running in the cloud. In cloud environments, LoadBalancer-type services configure cloud-specific load balancers.

## 20.3. CONFIGURING THE CLUSTER TO USE UNIQUE EXTERNAL IPS

In non-cloud clusters, **`ingressIPNetworkCIDR`** is set by default to **`172.29.0.0/16`**. If your cluster environment is not already using this private range, you can use the default. However, if you want to use a different range, then you must set **`ingressIPNetworkCIDR`** in the ***/etc/origin/master/master-config.yaml*** file before you assign an ingress IP. Then, restart the master service.

**Caution**

External IPs assigned to services of type **`LoadBalancer`** will always be in the range of **`ingressIPNetworkCIDR`**. If **`ingressIPNetworkCIDR`** is changed such that the assigned external IPs are no longer in range, the affected services will be assigned new external IPs compatible with the new range.

**Example 20.2. Sample /etc/origin/master/master-config.yaml**

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

## 20.4. CONFIGURING AN INGRESS IP FOR A SERVICE

To assign an ingress IP:

1. Create a YAML file for a LoadBalancer service that requests a specific IP via the **`loadBalancerIP`** setting:

   **Example 20.3. Sample LoadBalancer Configuration**

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-1
   spec:
     ports:
     - name: db
       port: 3306
   ```

```
    loadBalancerIP: 172.29.0.1
    type: LoadBalancer
    selector:
      name: my-db-selector
```

2. Create a LoadBalancer service on your pod:

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

3. Check the service for an external IP. For example, for a service named **myservice**:

```
$ oc get svc myservice
```

When your LoadBalancer-type service has an external IP assigned, the output displays the IP:

```
NAME          CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
myservice     172.30.74.106   172.29.0.1    3306/TCP    30s
```

## 20.5. ROUTING THE INGRESS CIDR FOR DEVELOPMENT OR TESTING

Add a static route directing traffic for the ingress CIDR to a node in the cluster. For example:

```
# route add -net 172.29.0.0/16 gw 10.66.140.17 eth0
```

In the example above, **172.29.0.0/16** is the **ingressIPNetworkCIDR**, and **10.66.140.17** is the node IP.

# CHAPTER 21. HANDLING OUT OF RESOURCE ERRORS

## 21.1. OVERVIEW

The node must preserve node stability when available compute resources are low. This is especially important when dealing with incompressible resources such as memory or disk. If either resource is exhausted, the node becomes unstable.

**Warning**

Failure to disable swap memory makes the node not recognize it is under **MemoryPressure**.

To take advantage of memory based evictions, operators must disable swap.

## 21.2. EVICTION POLICY

Using eviction policies, a node can proactively monitor for and prevent against total starvation of a compute resource.

In cases where a node is running low on available resources, it can proactively fail one or more pods in order to reclaim the starved resource using an eviction policy. When the node fails a pod, it terminates all containers in the pod, and the **`PodPhase`** is transitioned to **Failed**.

Platform administrators can configure eviction settings within the ***node-config.yaml*** file.

### 21.2.1. Eviction Signals

The node can be configured to trigger eviction decisions on the signals described in the table below. The value of each signal is described in the description column based on the node summary API.

To view the signals:

```
curl <certificate details> \
  https://<master>/api/v1/nodes/<node>/proxy/stats/summary
```

**Table 21.1. Supported Eviction Signals**

| Eviction Signal | Description |
|---|---|
| **`memory.available`** | **`memory.available`** = **`node.status.capacity[memory]`** - **`node.stats.memory.workingSet`** |

| Eviction Signal | Description |
|---|---|
| **`nodefs.available`** | **`nodefs.available`** = **`node.stats.fs.available`** |
| **`nodefs.inodesFree`** | **`nodefs.inodesFree`** = **`node.stats.fs.inodesFree`** |
| **`imagefs.available`** | **`imagefs.available`** = **`node.stats.runtime.imagefs.available`** |
| **`imagefs.inodesFree`** | **`imagefs.inodesFree`** = **`node.stats.runtime.imagefs.inodesFree`** |

The node supports two file system partitions when detecting disk pressure.

- The **`nodefs`** file system that the node uses for local disk volumes, daemon logs, and so on (for example, the file system that provides **`/`**).
- The **`imagefs`** file system that the container runtime uses for storing images and individual container writable layers.

The node auto-discovers these file systems using **`cAdvisor`**.

If you store volumes and logs in a dedicated file system, the node will not monitor that file system at this time.

**Note**

As of OpenShift Container Platform 3.4, the node supports the ability to trigger eviction decisions based on disk pressure. Operators must opt in to enable disk-based evictions. Prior to evicting pods due to disk pressure, the node will also perform container and image garbage collection. In future releases, garbage collection will be deprecated in favor of a pure disk eviction based configuration.

### 21.2.2. Eviction Thresholds

You can configure a node to specify eviction thresholds, which trigger the node to reclaim resources.

Eviction thresholds can be soft, for when you allow a grace period before reclaiming resources, and hard, for when the node takes immediate action when a threshold is met.

Thresholds are configured in the following form:

```
<eviction_signal><operator><quantity>
```

- Valid **`eviction-signal`** tokens as defined by eviction signals.

- Valid **`operator`** tokens are **`<`**.
- Valid **`quantity`** tokens must match the quantity representation used by Kubernetes.
- an eviction threshold can be expressed as a percentage if it ends with the **`%`** token.

For example, if an operator has a node with 10Gi of memory, and that operator wants to induce eviction if available memory falls below 1Gi, an eviction threshold for memory can be specified as either of the following:

memory.available<1Gi memory.available<10% ---

#### 21.2.2.1. Soft Eviction Thresholds

A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period. The node does not reclaim resources associated with the eviction signal until that grace period is exceeded. If no grace period is provided, the node errors on startup.

In addition, if a soft eviction threshold is met, an operator can specify a maximum allowed pod termination grace period to use when evicting pods from the node. If specified, the node uses the lesser value among the **`pod.Spec.TerminationGracePeriodSeconds`** and the maximum-allowed grace period. If not specified, the node kills pods immediately with no graceful termination.

To configure soft eviction thresholds, the following flags are supported:

- **`eviction-soft`**: a set of eviction thresholds (for example, **`memory.available<1.5Gi`**) that, if met over a corresponding grace period, triggers a pod eviction.
- **`eviction-soft-grace-period`**: a set of eviction grace periods (for example, **`memory.available=1m30s`**) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction.
- **`eviction-max-pod-grace-period`**: the maximum-allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met.

#### 21.2.2.2. Hard Eviction Thresholds

A hard eviction threshold has no grace period and, if observed, the node takes immediate action to reclaim the associated starved resource. If a hard eviction threshold is met, the node kills the pod immediately with no graceful termination.

To configure hard eviction thresholds, the following flag is supported:

- **`eviction-hard`**: a set of eviction thresholds (for example, **`memory.available<1Gi`**) that, if met, triggers a pod eviction.

### 21.2.3. Oscillation of Node Conditions

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node condition oscillates between **true** and **false**, which can confuse the scheduler.

To protect this, set the following flag to control how long the node must wait before transitioning out of a pressure condition:

- **`eviction-pressure-transition-period`**: the duration that the node has to wait before transitioning out of an eviction pressure condition.

Before toggling the condition back to **false**, the node ensures that it has not observed a met eviction threshold for the specified pressure condition for the period specified.

### 21.2.4. Eviction Monitoring Interval

The node evaluates and monitors eviction thresholds every 10 seconds and the value can not be modified. This is the housekeeping interval.

### 21.2.5. Mapping Eviction Signals to Node Conditions

The node can map one or more eviction signals to a corresponding node condition.

If an eviction threshold is met, independent of its associated grace period, the node reports a condition indicating that the node is under pressure.

The following node conditions are defined that correspond to the specified eviction signal.

**Table 21.2. Node Conditions Related to Low Resources**

| Node Condition | Eviction Signal | Description |
|---|---|---|
| **`MemoryPressure`** | **`memory.available`** | Available memory on the node has satisfied an eviction threshold. |
| **`DiskPressure`** | **`nodefs.available`**, **`nodefs.inodesFree`**, **`imagefs.available`**, or **`imagefs.inodesFree`** | Available disk space and inodes on either the node's root file system or image file system has satisfied an eviction threshold. |

When the above is set the node continues to report node status updates at the frequency specified by the **`node-status-update-frequency`** argument, which defaults to **`10s`**.

### 21.2.6. Reclaiming Node-level Resources

If an eviction criteria is satisfied, the node initiates the process of reclaiming the pressured resource until it observes that the signal has gone below its defined threshold. During this time, the node does not support scheduling any new pods.

The node attempts to reclaim node-level resources prior to evicting end-user pods. If disk pressure is observed, the node reclaims node-level resources differently if the machine has a dedicated **`imagefs`** configured for the container runtime.

#### 21.2.6.1. With Imagefs

If the **`nodefs`** file system meets eviction thresholds, the node frees up disk space in the following order:

- Delete dead pods/containers

If the **`imagefs`** file system meets eviction thresholds, the node frees up disk space in the following order:

- Delete all unused images

#### 21.2.6.2. Without Imagefs

If the **`nodefs`** file system meets eviction thresholds, the node frees up disk space in the following order:

- Delete dead pods/containers
- Delete all unused images

### 21.2.7. Eviction of Pods

If an eviction threshold is met and the grace period is passed, the node initiates the process of evicting pods until it observes the signal going below its defined threshold.

The node ranks pods for eviction by their quality of service, and, among those with the same quality of service, by the consumption of the starved compute resource relative to the pod's scheduling request.

- **`BestEffort`**: pods that consume the most of the starved resource are failed first.
- **`Burstable`**: pods that consume the most of the starved resource relative to their request for that resource are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.
- **`Guaranteed`**: pods that consume the most of the starved resource relative to their request are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the starved resource.

A **`Guaranteed`** pod will never be evicted because of another pod's resource consumption unless a system daemon (node, **docker**, **journald**, etc) is consuming more resources than were reserved via **system-reserved**, or **kube-reserved** allocations or if the node has only **`Guaranteed`** pods remaining.

If the latter, the node evicts a **`Guaranteed`** pod that least impacts node stability and limits the impact of the unexpected consumption to other **`Guaranteed`** pods.

Local disk is a **`BestEffort`** resource. If necessary, the node will evict pods one at a time to reclaim disk when **`DiskPressure`** is encountered. The node ranks pods by quality of service. If the node is responding to inode starvation, it will reclaim inodes by evicting pods with the lowest quality of service first. If the node is responding to lack of available disk, it will rank pods within a quality of service that consumes the largest amount of local disk, and evict those pods first.

**Note**

At this time, volumes that are backed by local disk are only deleted when a pod is deleted from the API server instead of when the pod is terminated.

As a result, if a pod is evicted as a consequence of consuming too much disk in an **`EmptyDir`** volume, the pod will be evicted, but the local volume usage will not be reclaimed by the node. The node will keep evicting pods on the node to prevent total exhaustion of disk. Operators can reclaim the disk by manually deleting the evicted pods from the node once terminated.

This will be remedied in a future release.

### 21.2.8. Scheduler

The scheduler views node conditions when placing additional pods on the node. For example, if the node has an eviction threshold like the following:

```
eviction-hard is "memory.available<500Mi"
```

and available memory falls below 500Mi, the node reports a value in **`Node.Status.Conditions`** as **`MemoryPressure`** as true.

**Table 21.3. Node Conditions and Scheduler Behavior**

| Node Condition | Scheduler Behavior |
|---|---|
| **`MemoryPressure`** | If a node reports this condition, the scheduler will not place **`BestEffort`** pods on that node. |
| **`DiskPressure`** | If a node reports this condition, the scheduler will not place any additional pods on that node. |

### 21.2.9. Example Scenario

Consider the following scenario:

- Node memory capacity of **`10Gi`**.
- The operator wants to reserve 10% of memory capacity for system daemons (kernel, node, etc.).
- The operator wants to evict pods at 95% memory utilization to reduce thrashing and incidence of system OOM.

A node reports two values:

- **`Capacity`**: How much resource is on the machine
- **`Allocatable`**: How much resource is made available for scheduling.

The goal is to allow the scheduler to fully allocate a node and to not have evictions occur.

Evictions should only occur if pods use more than their requested amount of resource.

To facilitate this scenario, the node configuration file (the ***node-config.yaml*** file) is modified as follows:

```
kubeletArguments:
  eviction-hard: 1
    - "memory.available<500Mi"
  system-reserved:
    - "1.5Gi"
```

1

This threshold can either be **`eviction-hard`** or **`eviction-soft`**.

**Note**

Soft eviction usage is more common when you are targeting a certain level of utilization, but can tolerate temporary spikes. It is recommended that the soft eviction threshold is always less than the hard eviction threshold, but the time period is operator specific. The system reservation should also cover the soft eviction threshold.

Implicit in this configuration is the understanding that **`system-reserved`** should include the amount of memory covered by the eviction threshold.

To reach that capacity, either some pod is using more than its request, or the system is using more than **`1Gi`**.

If a node has 10 Gi of capacity, and you want to reserve 10% of that capacity for the system daemons, do the following:

```
capacity = 10 Gi
system-reserved = 10 Gi * .01 = 1 Gi
```

The node allocatable value in this setting becomes:

```
allocatable = capacity - system-reserved = 9 Gi
```

This means by default, the scheduler will schedule pods that request 9 Gi of memory to that node.

If you want to turn on eviction so that eviction is triggered when the node observes that available memory falls below 10% of capacity for 30 seconds, or immediately when it falls below 5% of capacity, you need the scheduler to see allocatable as 8Gi. Therefore, ensure your system reservation covers the greater of your eviction thresholds.

```
capacity = 10 Gi
eviction-threshold = 10 Gi * .05 = .5 Gi
system-reserved = (10Gi * .01) + eviction-threshold = 1.5 Gi
allocatable = capacity - system-reserved = 8.5 Gi
```

You must set **`system-reserved`** equal to the amount of resource you want to reserve for system-daemons, plus the amount of resource you want to reserve before triggering evictions.

This configuration ensures that the scheduler does not place pods on a node that immediately induce memory pressure and trigger eviction assuming those pods use less than their configured request.

## 21.3. OUT OF RESOURCE AND OUT OF MEMORY

If the node experiences a system out of memory (OOM) event before it is able to reclaim memory, the node depends on the OOM killer to respond.

The node sets a **`oom_score_adj`** value for each container based on the quality of service for the pod.

**Table 21.4. Quality of Service OOM Scores**

| Quality of Service | `oom_score_adj` Value |
|---|---|
| **`Guaranteed`** | -998 |
| **`BestEffort`** | 1000 |
| **`Burstable`** | min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999) |

If the node is unable to reclaim memory prior to experiencing a system OOM event, the **`oom_killer`** calculates an **`oom_score`**:

```
% of node memory a container is using + `oom_score_adj` = `oom_score`
```

The node then kills the container with the highest score.

Containers with the lowest quality of service that are consuming the largest amount of memory relative to the scheduling request are failed first.

Unlike pod eviction, if a pod container is OOM failed, it can be restarted by the node based on its **`RestartPolicy`**.

## 21.4. RECOMMENDED PRACTICES

### 21.4.1. DaemonSets and Out of Resource Handling

If a node evicts a pod that was created by a DaemonSet, the pod will immediately be recreated and rescheduled back to the same node, because the node has no ability to distinguish a pod created from a DaemonSet versus any other object.

In general, DaemonSets should not create **`BestEffort`** pods to avoid being identified as a candidate pod for eviction. Instead DaemonSets should ideally launch **`Guaranteed`** pods.

# CHAPTER 22. MONITORING ROUTERS

## 22.1. 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.

## 22.2. VIEWING STATISTICS

The HAProxy router exposes a web listener for the HAProxy statistics. Enter the router's public IP address and the correctly configured port (**1936** by default) to view the statistics page, and enter the administrator password when prompted. This password and port are configured during the router installation, but they can be found by viewing the ***haproxy.config*** file on the container.

## 22.3. DISABLING STATISTICS VIEW

By default the HAProxy statistics are exposed on port **1936** (with a password protected account). To disable exposing the HAProxy statistics, specify **0** as the stats port number.

```
$ oadm router hap --service-account=router --stats-port=0
```

Note: HAProxy will still collect and store statistics, it would just *not* expose them via a web listener. You can still get access to the statistics by sending a request to the HAProxy AF_UNIX socket inside the HAProxy Router container.

```
$ cmd="echo 'show stat' | socat - UNIX-
CONNECT:/var/lib/haproxy/run/haproxy.sock"
$ routerPod=$(oc get pods --selector="router=router"  \
    --template="{{with index .items 0}}{{.metadata.name}}{{end}}")
$ oc exec $routerPod -- bash -c "$cmd"
```

**Important**

For security purposes, the **oc exec** command does not work when accessing privileged containers. Instead, you can SSH into a node host, then use the **docker exec** command on the desired container.

## 22.4. VIEWING LOGS

To view a router log, run the **oc logs** command on the pod. Since the router is running as a plug-in process that manages the underlying implementation, the log is for the plug-in, not the actual HAProxy log.

To view the logs generated by HAProxy, start a syslog server and pass the location to a router pod using the following environment variables.

**Table 22.1. Router Syslog Variables**

| Environment Variable | Description |
|---|---|
| **ROUTER_SYSLOG_ADDRESS** | The IP address of the syslog server. Port **514** is the default if no port is specified. |
| **ROUTER_LOG_LEVEL** | Optional. Set to change the HAProxy log level. If not set, the default log level is **warning**. This can be changed to any log level that HAProxy supports. |

To set a running router pod to send messages to a syslog server:

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=<dest_ip:dest_port>
ROUTER_LOG_LEVEL=<level>
```

For example, the following sets HAProxy to send logs to 127.0.0.1 with the default port **514** and changes the log level to **debug**.

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1
ROUTER_LOG_LEVEL=debug
```

## 22.5. VIEWING THE ROUTER INTERNALS

### routes.json

Routes are processed by the HAProxy router, and are stored both in memory, on disk, and in the HAProxy configuration file. The internal route representation, which is passed to the template to generate the HAProxy configuration file, is found in the ***/var/lib/haproxy/router/routes.json*** file. When troubleshooting a routing issue, view this file to see the data being used to drive configuration.

### HAProxy configuration

You can find the HAProxy configuration and the backends that have been created for specific routes in the ***/var/lib/haproxy/conf/haproxy.config*** file. The mapping files are found in the same directory. The helper frontend and backends use mapping files when mapping incoming requests to a backend.

### Certificates

Certificates are stored in two places:

- Certificates for edge terminated and re-encrypt terminated routes are stored in the ***/var/lib/haproxy/router/certs*** directory.
- Certificates that are used for connecting to backends for re-encrypt terminated routes are stored in the ***/var/lib/haproxy/router/cacerts*** directory.

The files are keyed by the namespace and name of the route. The key, certificate, and CA certificate are concatenated into a single file. You can use OpenSSL to view the contents of these files.

# CHAPTER 23. HIGH AVAILABILITY

## 23.1. OVERVIEW

This topic describes setting up high availability for pods and services on your OpenShift Container Platform cluster.

IP failover manages a pool of Virtual IP (VIP) addresses on a set of nodes. Every VIP in the set will be serviced by a node selected from the set. As long a single node is available, the VIPs will be served. There is no way to explicitly distribute the VIPs over the nodes. so there may be nodes with no VIPs and other nodes with many VIPs. If there is only one node, all VIPs will be on it.

**Note**

The VIPs must be routable from outside the cluster.

IP failover monitors a port on each VIP to determine whether the port is reachable on the node. If the port is not reachable, the VIP will not be assigned to the node. If the port is set to `0`, this check is suppressed. IP failover uses Keepalived to host a set of externally accessible VIP addresses on a set of hosts. Each VIP is only serviced by a single host at a time. **Keepalived** uses the VRRP protocol to determine which host (from the set of hosts) will service which VIP. If a host becomes unavailable or if the service that **Keepalived** is watching does not respond, the VIP is switched to another host from the set. Thus, a VIP is always serviced as long as a host is available.

OpenShift Container Platform supports creation of IP failover deployment configration, by running the `oadm ipfailover` command. The IP failover deployment configration specifies the set of VIP addresses, and the set of nodes on which to service them. A cluster can have multiple IP failover deployment configurations, with each managing its own set of unique VIP addresses. Each node in the IP failover configuration runs an ipfailover pod, and this pod runs **Keepalived**.

When using VIPs to access a pod with host networking (e.g. a router), the application pod should be running on all nodes that are running the ipfailover pods. This enables any of the ipfailover nodes to become the master and service the VIPs when needed. If application pods are not running on all nodes with ipfailover, either some ipfailover nodes will never service the VIPs or some application pods will never receive any traffic. Use the same selector and replication count, for both ipfailover and the application pods, to avoid this mismatch.

While using VIPs to access a service, any of the nodes can be in the ipfailover set of nodes, since the service is reachable on all nodes (no matter where the application pod is running). Any of the ipfailover nodes can become master at any time. The service can either use external IPs and a service port or it can use a nodePort.

When using external IPs in the service definition the VIPs are set to the external IPs and the ipfailover monitoring port is set to the service port. A nodePort is open on every node in the cluster and the service will load balance traffic from whatever node currently supports the VIP. In this case, the ipfailover monitoring port is set to the nodePort in the service definition.

**Important**

Setting up a nodePort is a privileged operation.

**Important**

Even though a service VIP is highly available, performance can still be affected. **keepalived** makes sure that each of the VIPs is serviced by some node in the configuration, and several VIPs may end up on the same node even when other nodes have none. Strategies that externally load balance across a set of VIPs may be thawed when ipfailover puts multiple VIPs on the same node.

When you use ingressIP, you can set up ipfailover to have the same VIP range as the ingressIP range. You can also disable the monitoring port. In this case, all the VIPs will appear on same node in the cluster. Any user can set up a service with an ingressIP and have it highly available.

**Important**

There are a maximum of 255 VIPs in the cluster.

## 23.2. CONFIGURING IP FAILOVER

Use the **`oadm ipfailover`** command with suitable options, to create ipfailover deployment configuration.

**Important**

Currently, ipfailover is not compatible with cloud infrastructures. For AWS, an Elastic Load Balancer (ELB) can be used to make OpenShift Container Platform highly available, using the AWS console.

As an administrator, you can configure ipfailover on an entire cluster, or on a subset of nodes, as defined by the label selector. You can also configure multiple IP failover deployment configrations in your cluster, where each one is independent of the others. The **`oadm ipfailover`** command creates a ipfailover deployment configration which ensures that a failover pod runs on each of the nodes matching the constraints or the label used. This pod runs **Keepalived** which uses VRRP (Virtual Router Redundancy Protocol) among all the **Keepalived** daemons to ensure that the service on the watched port is available, and if it is not, **Keepalived** will automatically float the VIPs.

For production use, make sure to use a **`--selector=<label>`** with at least two nodes to select the nodes. Also, set a **`--replicas=<n>`** value that matches the number of nodes for the given labeled selector.

The **`oadm ipfailover`** command includes command line options that set environment variables that control **Keepalived**. The environment variables start with **`OPENSHIFT_HA_*`** and they can be changed as needed.

For example, the command below will create an IP failover configuration on a selection of nodes labeled **`router=us-west-ha`** (on 4 nodes with 7 virtual IPs monitoring a service listening on port 80, such as the router process).

```
$ oadm ipfailover --selector="router=us-west-ha" --virtual-
ips="1.2.3.4,10.1.1.100-104,5.6.7.8" --watch-port=80 --replicas=4 --
create
```

### 23.2.1. 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.

### 23.2.2. Check and Notify Scripts

The **keepalived** port monitoring feature supports one or two scripts that are run on a configured interval to verify that the application is available. The default script uses a simple TCP connection to verify that the application is running. This test is suppressed when the monitoring port is set to `0`. The administrator can supply an additional script that does the needed verification. For example, the script can test a web server by issuing a request and verifying the response. The script must exit with `0` for **PASS** and `1` for **FAIL**.

The administrator provides the additional script, via the `--check-script=<script>` option. By default, the check is done every two seconds, but can be changed using the `--check-interval=<seconds>` option.

For each VIP, **keepalived** keeps the state of the node. The VIP on the node may be in **MASTER**, **BACKUP**, or **FAULT** state. All VIPs on the node that are not in the **FAULT** state participate in the negotiation to decide which will be **MASTER** for the VIP. All of the losers enter the **BACKUP** state. When the **check** script on the **MASTER** fails, the VIP enters the **FAULT** state and triggers a renegotiation. When the **BACKUP** fails, the VIP enters the **FAULT** state. When the **check** script passes again on a VIP in the **FAULT** state, it exits **FAULT** and negotiates for **MASTER**. The resulting state is either **MASTER** or **BACKUP**.

The administrator can provide an optional **notify** script, which is called whenever the state changes. **Keepalived** passes the following three parameters to the script:

- **$1** - "GROUP"|"INSTANCE"
- **$2** - Name of the group or instance
- **$3** - The new state ("MASTER"|"BACKUP"|"FAULT")

These scripts run in the IP failover pod and use the pod’s file system, not the host file system. The options require the full path to the script. The administrator must make the script available in the pod to extract the results from running the **notify** script. The recommended approach for providing the scripts is to use a ConfigMap.

The full path names of the **check** and **notify** scripts are added to the **keepalived** configuration file, _ **/etc/keepalived/keepalived.conf_**, which is loaded every time **keepalived** starts. The scripts can be added to the pod with a ConfigMap as follows.

1. Create the desired script and create a ConfigMap to hold it. The script has no input arguments and must return **0** for **OK** and **1** for **FAIL**.

   The check script, ***mycheckscript.sh***:

   ```
   #!/bin/bash
       # Whatever tests are needed
       # E.g., send request and verify response
   exit 0
   ```

2. Create the ConfigMap:

   ```
   $ oc create configmap mycustomcheck --from-file=mycheckscript.sh
   ```

3. There are two approaches to adding the script to the pod: use **oc** commands or edit the deployment configuration.

   a. Using **oc** commands:

   ```
   $ oc set env dc/ipf-ha-router \
       OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
   $ oc volume dc/ipf-ha-router --add --overwrite \
       --name=config-volume \
       --mount-path=/etc/keepalive \
       --source='{"configMap": { "name": "mycustomcheck"}}'
   ```

   b. Editing the **ipf-ha-router** deployment configuration:

      i. Use **`oc edit dc ipf-ha-router`** to edit the router deployment configuration with a text editor.

      ```
      ...
          spec:
            containers:
            - env:
              - name: OPENSHIFT_HA_CHECK_SCRIPT  1
                value: /etc/keepalive/mycheckscript.sh
      ...
              volumeMounts: 2
              - mountPath: /etc/keepalive
                name: config-volume
            dnsPolicy: ClusterFirst
      ...
            volumes: 3
            - configMap:
                name: customrouter
              name: config-volume
      ...
      ```

      1

In the **`spec.container.env`** field, add the **`OPENSHIFT_HA_CHECK_SCRIPT`** environment variable to point to the mounted script file.

**2**

Add the **`spec.container.volumeMounts`** field to create the mount point.

**3**

Add a new **`spec.volumes`** field to mention the ConfigMap.

ii. Save the changes and exit the editor. This restarts **ipf-ha-router**.

### 23.2.3. Keepalived Multicast

OpenShift Container Platform's ipfailover internally uses **keepalived**.

**Important**

Please ensure that multicast is enabled on the nodes labeled above and they can accept network traffic for 224.0.0.18 (the VRRP multicast IP address).

Before starting the **keepalived** daemon, the startup script verifies the **`iptables`** rule that allows multicast traffic to flow. If there is no such rule, the startup script creates a new rule and adds it to the IP tables configuration. Where this new rule gets added to the IP tables configuration depends on the **`--iptables-chain=`** option. If there is an **`--iptables-chain=`** option specified, the rule gets added to the specified chain in the option. Otherwise, the rule is added to the **`INPUT`** chain.

**Important**

The **`iptables`** rule must be present whenever there is one or more **keepalived** daemon running on the node.

The **`iptables`** rule can be removed after the last **keepalived** daemon terminates. The rule is not automatically removed.

You can manually manage the **`iptables`** rule on each of the nodes. It only gets created when none is present (as long as ipfailover is not created with the **`--iptable-chain=""`** option).

**Important**

You must ensure that the manually added rules persist after a system restart.

Be careful since every **keepalived** daemon uses the VRRP protocol over multicast 224.0.0.18 to negotiate with its peers. There must be a different VRRP-id (in the range 0..255) for each VIP.

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

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

echo "Check multicast groups ... "
ip maddr show $interface | grep 224.0.0 | grep $interface

EOF
done;
```

### 23.2.4. Command Line Options and Environment Variables

**Table 23.1. Command Line Options and Environment Variables**

| Option | Variable Name | Default | Notes |
|---|---|---|---|
| **`--watch-port`** | **`OPENSHIFT_HA_MONITOR_PORT`** | 80 | The ipfailover pod tries to open a TCP connection to this port on each VIP. If connection is established, the service is considered to be running. If this port is set to 0, the test always passes. |
| **`--interface`** | **`OPENSHIFT_HA_NETWORK_INTERFACE`** | | The interface name for the ip failover to use, to send VRRP traffic. By default, **`eth0`** is used. |
| **`--replicas`** | **`OPENSHIFT_HA_REPLICA_COUNT`** | 2 | Number of replicas to create. This must match **`spec.replicas`** value in ipfailover deployment configration. |
| **`--virtual-ips`** | **`OPENSHIFT_HA_VIRTUAL_IPS`** | | The list of IP address ranges to replicate. This must be provided. E.g., 1.2.3.4-6,1.2.3.9 Please see this discussion for more details. |
| **`--vrrp-id-offset`** | **`OPENSHIFT_HA_VRRP_ID_OFFSET`** | 0 | Please see VRRP Id Offset discussion for more details. |

| Option | Variable Name | Default | Notes |
|---|---|---|---|
| `--iptables-chain` | `OPENSHIFT_HA_IPTABLES_CHAIN` | INPUT | The name of the iptables chain, to automatically add an `iptables` rule to allow the VRRP traffic on. If the value is not set, an `iptables` rule will not be added. If the chain does not exist, it is not created. |

### 23.2.5. VRRP Id Offset

Each ipfailover pod managed by the ipfailover deployment configration (1 pod per node/replica) runs a **keepalived** daemon. As more ipfailover deployment configrations are configured, more pods are created and more daemons join into the common VRRP negotiation. This negotiation is done by all the **keepalived** daemons and it determines which nodes will service which VIPs.

Internally, **keepalived** assigns a unique vrrp-id to each VIP. The negotiation uses this set of vrrp-ids, when a decision is made, the VIP corresponding to the winning vrrp-id is serviced on the winning node.

Therefore, for every VIP defined in the ipfailover deployment configration, the ipfailover pod must assign a corresponding vrrp-id. This is done by starting at **`--vrrp-id-offset`** and sequentially assigning the vrrp-ids to the list of VIPs. The vrrp-ids may have values in the range 1..255.

When there are multiple ipfailover deployment configration care must be taken to specify **`--vrrp-id-offset`** so that there is room to increase the number of VIPS in the deployment configration and none of the vrrp-id ranges overlap.

### 23.2.6. Configuring a Highly-available Service

The following steps describe how to set up highly-available **router** and **geo-cache** network services with IP failover on a set of nodes.

1. Label the nodes that will be used for the services. This step can be optional if you run the services on all the nodes in your OpenShift Container Platform cluster and will use VIPs that can float within all nodes in the cluster.

   The following example defines a label for nodes that are servicing traffic in the US west geography **ha-svc-nodes=geo-us-west**

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

2. Create the service account. You can use ipfailover or when using a router (depending on your environment policies), you can either reuse the **router** service account created previously or a new ipfailover service account.

   The example below creates a new service account with the name ipfailover in the **default** namespace:

   ```
   $ oc create serviceaccount ipfailover -n default
   ```

3. Add the ipfailover service account in the **default** namespace to the **privileged** SCC:

```
$ oadm policy add-scc-to-user privileged
system:serviceaccount:default:ipfailover
```

4. Start the **router** and the **geo-cache** services.

   **Important**

   Since the ipfailover runs on all nodes from step 1, it is recommended to also run the router/service on all the step 1 nodes.

   a. Start the router with the nodes matching the labels used in the first step. The following example runs five instances using the ipfailover service account:

   ```
   $ oadm router ha-router-us-west --replicas=5 \
       --selector="ha-svc-nodes=geo-us-west" \
       --labels="ha-svc-nodes=geo-us-west" \
       --service-account=ipfailover
   ```

   b. Run the **geo-cache** service with a replica on each of the nodes. An example configuration for running a **geo-cache** service is provided here.

      **Important**

      Make sure that you replace the **myimages/geo-cache** Docker image referenced in the file with your intended image. Change the number of replicas to the number of nodes in the **geo-cache** label. Check that the label matches the one used in the first step.

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

5. Configure ipfailover for the **router** and **geo-cache** services. Each has its own VIPs and both use the same nodes labeled with **ha-svc-nodes=geo-us-west** in the first step. Please ensure that the number of replicas match the number of nodes listed in the label setup, in the first step.

   **Important**

   The **router**, **geo-cache**, and ipfailover all create deployment configration and all must have different names.

6. Specify the VIPs and the port number that ipfailover should monitor on the desired instances.

   Below is the ipfailover command for the **router**.

```
$ oadm ipfailover ipf-ha-router-us-west \
    --replicas=5 --watch-port=80 \
    --selector="ha-svc-nodes=geo-us-west" \
    --virtual-ips="10.245.2.101-105" \
    --iptables-chain="INPUT" \
    --service-account=ipfailover --create
```

Below is the **`oadm ipfailover`** command for the **geo-cache** service that is listening on port 9736. Since there are two **`ipfailover`** deployment configurations, the **`--vrrp-id-offset`** must be set so that each VIP gets its own offset. In this case, setting a value of **`10`** means that the **`ipf-ha-router-us-west`** can have a maximum of 10 VIPs (0-9) since **`ipf-ha-geo-cache`** is starting at 10.

```
$ oadm ipfailover ipf-ha-geo-cache \
    --replicas=5 --watch-port=9736 \
    --selector="ha-svc-nodes=geo-us-west" \
    --virtual-ips=10.245.3.101-105 \
    --vrrp-id-offset=10 \
    --service-account=ipfailover --create
```

In the commands above, there are **ipfailover**, **router**, and **geo-cache** pods on each node. The set of VIPs for each ipfailover configuration must not overlap and they must not be used elsewhere in the external or cloud environments. The five VIP addresses in each example, **`10.245.{2,3}.101-105`** are served by the two ipfailover deployment configurations. IP failover dynamically selects which address is served on which node.

The administrator sets up external DNS to point to the VIP addresses knowing that all the **router** VIPs point to the same **router**, and all the **geo-cache** VIPs point to the same **geo-cache** service. As long as one node remains running, all the VIP addresses are served.

7. Deploy the ipfailover router to monitor postgresql listening on node port 32439 and the external IP address, as defined in the **postgresql-ingress** service:

```
$ oadm ipfailover ipf-ha-postgresql \
    --replicas=1 <1> --selector="app-type=postgresql" <2> \
    --virtual-ips=10.9.54.100 <3> --watch-port=32439 <4>  \
    --credentials=/etc/origin/master/openshift-router.kubeconfig
\
    --service-account=ipfailover --create
```

Specifies the number of instances to deploy.

Restricts where the ipfailover is deployed.

Virtual IP address to monitor.

Port on which ipfailover will monitor on each node.

### 23.2.7. Dynamically Updating Virtual IPs for a Highly-available Service

The default deployment strategy for the IP failover service is to recreate the deployment. In order to dynamically update the VIPs for a highly available routing service with minimal or no downtime, you must:

- Update the IP failover service deployment configuration to use a rolling update strategy, and
- Update the **`OPENSHIFT_HA_VIRTUAL_IPS`** environment variable with the updated list or sets of virtual IP addresses.

The following example shows how to dynamically update the deployment strategy and the virtual IP addresses:

1. Consider an IP failover configuration that was created using the following:

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

2. Edit the deployment configuration:

```
$ oc edit dc/ipf-ha-router-us-west
```

3. Update the **`spec.strategy.type`** field from **`Recreate`** to **`Rolling`**:

```
spec:
  replicas: 5
  selector:
    ha-svc-nodes: geo-us-west
  strategy:
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    type: Rolling 1
```

1

Set to **`Rolling`**.

4. Update the **`OPENSHIFT_HA_VIRTUAL_IPS`** environment variable to contain the additional virtual IP addresses:

```
- name: OPENSHIFT_HA_VIRTUAL_IPS
  value: 10.245.2.101-105,10.245.2.110,10.245.2.201-205 1
```

1

**`10.245.2.110,10.245.2.201-205`** have been added to the list.

5. Update the external DNS to match the set of VIPs.

## 23.3. CONFIGURING SERVICE EXTERNALIP AND NODEPORT

The user can assign VIPs as ExternalIPs in a service. **Keepalived** makes sure that each VIP is served on some node in the ipfailover configuration. When a request arrives on the node, the service that is running on all nodes in the cluster, load balances the request among the service's endpoints.

The NodePorts can be set to the ipfailover watch port so that **keepalived** can check the application is running. The NodePort is exposed on all nodes in the cluster, therefore it is available to **keepalived** on all ipfailover nodes.

## 23.4. HIGH AVAILABILITY FOR INGRESSIP

In non-cloud clusters, ipfailover and ingressIP to a service can be combined. The result is high availability services for users that create services using ingressIP.

The approach is to specify an **`ingressIPNetworkCIDR`** range and then use the same range in creating the ipfailover configuration.

Since, ipfailover can support up to a maximum of 255 VIPs for the entire cluster, the **`ingressIPNetworkCIDR`** needs to be **`/24`** or less.

# CHAPTER 24. IPTABLES

## 24.1. OVERVIEW

There are many system components including OpenShift Container Platform, containers, and software that manage local firewall policies that rely on the kernel iptables configuration for proper network operation. In addition, the iptables configuration of all nodes in the cluster must be correct for networking to work.

All components independently work with iptables without knowledge of how other components are using them. This makes it very easy for one component to break another component's configuration. Further, OpenShift Container Platform and the Docker service assume that iptables remains set up exactly as they have set it up. They may not detect changes introduced by other components and if they do there may be some lag in implementing the fix. In particular, OpenShift Container Platform does monitor and fix problems. However, the Docker service does not.

**Important**

You must ensure that changes you make to the iptables configuration on a node do not impact the operation of OpenShift Container Platform and the Docker service. Also, changes will often need to be made on all nodes in the cluster. Use caution, as iptables is not designed to have multiple concurrent users and it is very easy to break OpenShift Container Platform and Docker networking.

The chains, order of the chains, and rules in the kernel iptables must be properly set up on each node in the cluster for OpenShift Container Platform and Docker networking to work properly. There are several tools and services that are commonly used in the system that interact with the kernel iptables and can accidentally impact OpenShift Container Platform and the Docker service.

## 24.2. IPTABLES

The iptables tool can be used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.

Independent of other use, such as a firewall, OpenShift Container Platform and the the Docker service manage chains in some of the tables. The chains are inserted in specific order and the rules are specific to their needs.

**Caution**

**`iptables --flush [chain]`** can remove key required configuration. Do not execute this command.

## 24.3. IPTABLES.SERVICE

The iptables service supports a local network firewall. It assumes total control of the iptables configuration. When it starts, it flushes and restores the complete iptables configuration. The restored rules are from its configuration file, ***/etc/sysconfig/iptables***. The configuration file is not kept up to date during operation, so the dynamically added rules are lost during every restart.

**Warning**

Stopping and starting **iptables.service** will destroy configuration that is required by OpenShift Container Platform and Docker. OpenShift Container Platform and Docker are not notified of the change.

```
# systemctl disable iptables.service
# systemctl mask iptables.service
```

If you need to run **iptables.service**, keep a limited configuration in the configuration file and rely on OpenShift Container Platform and Docker to install their needed rules.

The **iptables.service** configuration is loaded from:

```
/etc/sysconfig/iptables
```

To make permanent rules changes, edit the changes into this file. Do not include Docker or OpenShift Container Platform rules.

After **iptables.service** is started or restarted on a node, the Docker service and **atomic-openshift-node.service** must be restarted to reconstruct the needed iptables configuration.

**Important**

Restarting the Docker service will cause all containers running on the node to be stopped and restarted.

```
# systemctl restart iptables.service
# systemctl restart docker
# systemctl restart atomic-openshift-node.service
```

# CHAPTER 25. SECURING BUILDS BY STRATEGY

## 25.1. OVERVIEW

Builds in OpenShift Container Platform are run in privileged containers that have access to the Docker daemon socket. As a security measure, it is recommended to limit who can run builds and the strategy that is used for those builds. Custom builds are inherently less safe than Source builds, given that they can execute any code in the build with potentially full access to the node's Docker socket. Docker build permission should also be granted with caution as a vulnerability in the Docker build logic could result in a privileges being granted on the host node.

By default, all users that can create builds are granted permission to use 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. A user must have permission to create a build *and* permission to create on the build strategy subresource in order to create builds using that strategy. Default roles are provided which grant the **create** permission on the build strategy subresource.

**Table 25.1. Build Strategy Subresources and Roles**

| Strategy | Subresource | Role |
|---|---|---|
| Docker | builds/docker | system:build-strategy-docker |
| Source-to-Image | builds/source | system:build-strategy-source |
| Custom | builds/custom | system:build-strategy-custom |
| Custom | builds/jenkinspipeline | system:build-strategy-jenkinspipeline |

## 25.2. DISABLING A BUILD STRATEGY GLOBALLY

To prevent access to a particular build strategy globally, log in as a user with **cluster-admin** privileges and remove the corresponding role from the **system:authenticated** group:

```
$ oadm policy remove-cluster-role-from-group system:build-strategy-
custom system:authenticated
$ oadm policy remove-cluster-role-from-group system:build-strategy-
docker system:authenticated
$ oadm policy remove-cluster-role-from-group system:build-strategy-
source system:authenticated
$ oadm policy remove-cluster-role-from-group system:build-strategy-
jenkinspipeline system:authenticated
```

In versions prior to 3.2, the build strategy subresources were included in the **`admin`** and **`edit`** roles. Ensure the build strategy subresources are also removed from these roles:

```
$ oc edit clusterrole admin
$ oc edit clusterrole edit
```

For each role, remove the line that corresponds to the resource of the strategy to disable.

**Example 25.1. Disable the Docker Build Strategy for admin**

```
kind: ClusterRole
metadata:
  name: admin
...
rules:
- resources:
  - builds/custom
  - builds/docker 1
  - builds/source
  ...
...
```

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

## 25.3. RESTRICTING BUILD STRATEGIES TO A USER GLOBALLY

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

1. Disable global access to the build strategy.
2. Assign the role corresponding to the build strategy to a specific user. For example, to add the **system:build-strategy-docker** cluster role to the user **devuser**:

```
$ oadm policy add-cluster-role-to-user system:build-strategy-
docker devuser
```

**Warning**

Granting a user access at the cluster level to the **builds/docker** subresource means that the user will be able to create builds with the Docker strategy in any project in which they can create builds.

## 25.4. RESTRICTING BUILD STRATEGIES TO A USER WITHIN A

## PROJECT

Similar to granting the build strategy role to a user globally, to allow only a set of specific users within a project to create builds with a particular strategy:

1. Disable global access to the build strategy.

2. Assign the role corresponding to the build strategy to a specific user within a project. For example, to add the **system:build-strategy-docker** role within the project **devproject** to the user **devuser**:

   ```
   $ oadm policy add-role-to-user system:build-strategy-docker
   devuser -n devproject
   ```

# CHAPTER 26. RESTRICTING APPLICATION CAPABILITIES USING SECCOMP

## 26.1. OVERVIEW

Seccomp (secure computing mode) is used to restrict the set of system calls applications can make, allowing cluster administrators greater control over the security of workloads running in OpenShift Container Platform.

Seccomp support is achieved via two annotations in the pod configuration:

- **seccomp.security.alpha.kubernetes.io/pod**: profile applies to all containers in the pod that do not override
- **container.seccomp.security.alpha.kubernetes.io/<container_name>**: container-specific profile override

**Important**

Containers are run with **unconfined** seccomp settings by default.

For detailed design information, refer to the seccomp design document.

## 26.2. ENABLING SECCOMP

Seccomp is a feature of the Linux kernel. To ensure seccomp is enabled on your system, run:

```
$ cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

## 26.3. CONFIGURING OPENSHIFT CONTAINER PLATFORM FOR SECCOMP

A seccomp profile is a json file providing syscalls and the appropriate action to take when a syscall is invoked.

1. Create the seccomp profile.

   The default profile is sufficient in many cases, but the cluster administrator must define the security constraints of an individual system.

   To create your own custom profile, create a file on every node in the **`seccomp-profile-root`** directory.

   If you are using the default **runtime/default** profile, you do not need to create one.

2. Configure your nodes to use the **seccomp-profile-root** where your profiles will be stored. In the **node-config.yaml** via the **kubeletArguments**:

```
kubeletArguments:
  seccomp-profile-root:
    - "/your/path"
```

3. Restart the node service to apply the changes:

   ```
   # systemctl restart atomic-openshift-node
   ```

4. In order to control which profiles may be used, and to set the default profile, configure your SCC via the **seccompProfiles** field. The first profile will be used as a default.

   The allowable formats of the **seccompProfiles** field include:

   - **runtime/default**: the default profile for the container runtime (no profile required)
   - **unconfined**: unconfined profile, and disables seccomp
   - **localhost/<profile-name>**: the profile installed to the node's local seccomp profile root

     For example, if you are using the default **runtime/default** profile, configure the **restricted** SCC with:

     ```
     seccompProfiles:
     - runtime/default
     ```

## 26.4. CONFIGURING OPENSHIFT CONTAINER PLATFORM FOR A CUSTOM SECCOMP PROFILE

To ensure pods in your cluster run with a custom profile in the **restricted** SCC:

1. Create the seccomp profile in **seccomp-profile-root**.
2. Configure **seccomp-profile-root**:

   ```
   kubeletArguments:
     seccomp-profile-root:
       - "/your/path"
   ```

3. Restart the node service to apply the changes:

   ```
   # systemctl restart atomic-openshift-node
   ```

4. Configure the **restricted** SCC:

   ```
   seccompProfiles:
   - localhost/<profile-name>
   ```

# CHAPTER 27. SYSCTLS

## 27.1. OVERVIEW

Sysctl settings are exposed via Kubernetes, allowing users to modify certain kernel parameters at runtime for namespaces within a container. Only sysctls that are namespaced can be set independently on pods; if a sysctl is not namespaced (called *node-level*), it cannot be set within OpenShift Container Platform. Moreover, only those sysctls considered *safe* are whitelisted by default; other *unsafe* sysctls can be manually enabled on the node to be available to the user.

**Note**

As of OpenShift Container Platform 3.3.1, sysctl support is a feature in Technology Preview.

## 27.2. UNDERSTANDING SYSCTLS

In Linux, the sysctl interface allows an administrator to modify kernel parameters at runtime. Parameters are available via the ***/proc/sys/*** virtual process file system. The parameters cover various subsystems such as:

- kernel (common prefix: ***kernel.***)
- networking (common prefix: ***net.***)
- virtual memory (common prefix: ***vm.***)
- MDADM (common prefix: ***dev.***)

More subsystems are described in Kernel documentation. To get a list of all parameters, you can run:

```
$ sudo sysctl -a
```

## 27.3. NAMESPACED VS NODE-LEVEL SYSCTLS

A number of sysctls are *namespaced* in today’s Linux kernels. This means that they can be set independently for each pod on a node. Being namespaced is a requirement for sysctls to be accessible in a pod context within Kubernetes.

The following sysctls are known to be namespaced:

- ***kernel.shm****
- ***kernel.msg****
- ***kernel.sem***
- ***fs.mqueue.****
- ***net.****

Sysctls that are not namespaced are called *node-level* and must be set manually by the cluster administrator, either by means of the underlying Linux distribution of the nodes (e.g., via ***/etc/sysctls.conf***) or using a DaemonSet with privileged containers.

**Note**

Consider marking nodes with special sysctls as tainted. Only schedule pods onto them that need those sysctl settings. Use the Kubernetes *taints and toleration* feature to implement this.

## 27.4. SAFE VS UNSAFE SYSCTLS

Sysctls are grouped into *safe* and *unsafe* sysctls. In addition to proper namespacing, a safe sysctl must be properly isolated between pods on the same node. This means that setting a safe sysctl for one pod:

- must not have any influence on any other pod on the node,
- must not allow to harm the node's health, and
- must not allow to gain CPU or memory resources outside of the resource limits of a pod.

By far, most of the namespaced sysctls are not necessarily considered safe.

For OpenShift Container Platform 3.3.1, the following sysctls are supported (whitelisted) in the safe set:

- ***kernel.shm_rmid_forced***
- ***net.ipv4.ip_local_port_range***

This list will be extended in future versions when the kubelet supports better isolation mechanisms.

All safe sysctls are enabled by default. All unsafe sysctls are disabled by default and must be allowed manually by the cluster administrator on a per-node basis. Pods with disabled unsafe sysctls will be scheduled, but will fail to launch.

**Warning**

Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems like wrong behavior of containers, resource shortage, or complete breakage of a node.

## 27.5. ENABLING UNSAFE SYSCTLS

With the warning above in mind, the cluster administrator can allow certain unsafe sysctls for very special situations, e.g., high-performance or real-time application tuning.

If you want to use unsafe sysctls, cluster administrators must enable them individually on nodes. Only namespaced sysctls can be enabled this way.

1. Use the **`kubeletArguments`** field in the ***/etc/origin/node/node-config.yaml*** file, as described in Configuring Node Resources, to set the desired unsafe sysctls:

   ```
   kubeletArguments:
     experimental-allowed-unsafe-sysctls:
       - "kernel.msg*,net.ipv4.route.min_pmtu"
   ```

2. Restart the node service to apply the changes:

   ```
   # systemctl restart atomic-openshift-node
   ```

## 27.6. SETTING SYSCTLS FOR A POD

Sysctls are set on pods using annotations. They apply to all containers in the same pod.

Here is an example, with different annotations for safe and unsafe sysctls:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  annotations:
    security.alpha.kubernetes.io/sysctls: kernel.shm_rmid_forced=1
    security.alpha.kubernetes.io/unsafe-sysctls:
net.ipv4.route.min_pmtu=1000,kernel.msgmax=1 2 3
spec:
  ...
```

**Note**

A pod with the unsafe sysctls specified above will fail to launch on any node that has not enabled those two unsafe sysctls explicitly. As with node-level sysctls, use the taints and toleration feature or labels on nodes to schedule those pods onto the right nodes.

# CHAPTER 28. BUILDING DEPENDENCY TREES

## 28.1. OVERVIEW

OpenShift Container Platform 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.

## 28.2. USAGE

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

**Table 28.1. 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 29. BACKUP AND RESTORE

## 29.1. OVERVIEW

In OpenShift Container Platform, you can *back up* (saving state to separate storage) and *restore* (recreating state from separate storage) at the cluster level. There is also some preliminary support for per-project backup. The full state of a cluster installation includes:

- etcd data on each master
- API objects
- registry storage
- volume storage

This topic does not cover how to back up and restore persistent storage, as those topics are left to the underlying storage provider. However, an example of how to perform a **generic** backup of application data is provided.

**Important**

This topic only provides a generic way of backing up applications and the OpenShift Container Platform cluster. It can not take into account custom requirements. Therefore, you should create a full backup and restore procedure. To prevent data loss, necessary precautions should be taken.

## 29.2. PREREQUISITES

1. Because the restore procedure involves a complete reinstallation, save all the files used in the initial installation. This may include:
   - ***~/.config/openshift/installer.cfg.yml*** (from the Quick Installation method)
   - Ansible playbooks and inventory files (from the Advanced Installation method)
   - ***/etc/yum.repos.d/ose.repo*** (from the Disconnected Installation method)
2. Backup the procedures for post-installation steps. Some installations may involve steps that are not included in the installer. This may include changes to the services outside of the control of OpenShift Container Platform or the installation of extra services like monitoring agents. Additional configuration that is not supported yet by the advanced installer might also be affected, for example when using multiple authentication providers.
3. Install packages that provide various utility commands:

   ```
   # yum install etcd
   ```

4. If using a container-based installation, pull the etcd image instead:

   ```
   # docker pull rhel7/etcd
   ```

Note the location of the **etcd** data directory (or **`$ETCD_DATA_DIR`** in the following sections), which depends on how **etcd** is deployed.

| Deployment Type | Data Directory |
|---|---|
| all-in-one cluster | ***/var/lib/openshift/openshift.local.etcd*** |
| external etcd (not on master) | ***/var/lib/etcd*** |
| embedded etcd (on master) | ***/var/lib/origin/etcd*** |

## 29.3. CLUSTER BACKUP

1. Save all the certificates and keys, on each master:

```
# cd /etc/origin/master
# tar cf /tmp/certs-and-keys-$(hostname).tar \
    master.proxy-client.crt \
    master.proxy-client.key \
    proxyca.crt \
    proxyca.key \
    master.server.crt \
    master.server.key \
    ca.crt \
    ca.key \
    master.etcd-client.crt \
    master.etcd-client.key \
    master.etcd-ca.crt
```

2. If **etcd** is running on more than one host, stop it on each host:

```
# sudo systemctl stop etcd
```

   Although this step is not strictly necessary, doing so ensures that the **etcd** data is fully synchronized.

3. Create an **etcd** backup:

```
# etcdctl backup \
    --data-dir $ETCD_DATA_DIR \
    --backup-dir $ETCD_DATA_DIR.bak
```

**Note**

If **etcd** is running on more than one host, the various instances regularly synchronize their data, so creating a backup for one of them is sufficient.

**Note**

For a container-based installation, you must use **`docker exec`** to run **etcdctl** inside the container.

## 29.4. CLUSTER RESTORE

**Note**

This restore operation only works for single-member **etcd** clusters. For multiple-member **etcd** clusters, see Restoring **etcd**.

To restore the cluster:

1. Reinstall OpenShift Container Platform.

   This should be done in the same way that OpenShift Container Platform was previously installed.

2. Run all necessary post-installation steps.

3. Restore the certificates and keys, on each master:

```
# cd /etc/origin/master
# tar xvf /tmp/certs-and-keys-$(hostname).tar
```

4. Restore from the **etcd** backup:

```
# mv $ETCD_DATA_DIR $ETCD_DATA_DIR.orig
# cp -Rp $ETCD_DATA_DIR.bak $ETCD_DATA_DIR
# chcon -R --reference $ETCD_DATA_DIR.orig $ETCD_DATA_DIR
# chown -R etcd:etcd $ETCD_DATA_DIR
```

## 29.5. ADDING NEW ETCD HOSTS

In cases where etcd hosts have failed, but you have at least one host still running, you can use the one surviving host to recover etcd hosts without downtime.

### Suggested Cluster Size

Having a cluster with an odd number of etcd hosts can account for fault tolerance. Having an odd number of etcd hosts does not change the number needed for majority, but increases the tolerance for failure. For example, a cluster size of seven hosts has a majority of four, leaving a failure tolerance of three. This ensures that four hosts will be guaranteed to operate.

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

**Note**

The following presumes you have a backup of the ***/etc/etcd*** configuration for the etcd hosts.

1. Add the desired number of hosts to the cluster. The rest of this procedure presumes you have added just one host, but if adding multiple, perform all steps on each host.

2. Upgrade etcd on the surviving node:

```
# yum install etcd iptables-services
```

   Ensure version **`etcd-2.3.7-4.el7.x86_64`** or greater is installed, and that the same version is installed on each host.

3. On the new host, add the appropriate iptables rules:

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

4. Generate the required certificates for the new host. On a surviving etcd host:

   a. Create a copy of the ***/etc/etcd/ca/*** directory.

   b. Set the variables and working directory for the certificates, ensuring to create the ***PREFIX*** directory if one has not been created:

```
# cd /etc/etcd
# export NEW_ETCD="<NEW_HOST_NAME>"

# export CN=$NEW_ETCD
# export SAN="IP:<NEW_HOST_IP>"
# export PREFIX="./generated_certs/etcd-$CN/"
```

   c. Create the ***server.csr*** and ***server.crt*** certificates:

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

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

d. Create the ***peer.csr*** and ***peer.crt*** certificates:

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

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

e. Copy the ***ca.crt*** and ***etcd.conf*** files, and archive the contents of the directory:

```
# cp ca.crt ${PREFIX}
# cp etcd.conf ${PREFIX}
# tar -czvf ${PREFIX}${CN}.tgz -C ${PREFIX} .
```

f. Transfer the files to the new etcd hosts:

```
# scp ${PREFIX}${CN}.tgz  $CN:/etc/etcd/
```

5. While still on the surviving etcd host, add the new host to the cluster, take the copy of etcd, and transfer it to the new host:

a. Add the new host to the cluster:

```
# export ETCD_CA_HOST="<SURVIVING_ETCD_HOSTNAME>"
# export NEW_ETCD="<NEW_ETCD_HOSTNAME>"
# export NEW_ETCD_IP="<NEW_HOST_IP>"

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

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<SURVIVING_ETCD_HOST>=https:/<SURVIVING_HOST_IP>:2380
ETCD_INITIAL_CLUSTER_STATE="existing"
```

b. Create a backup of the surviving etcd host, and transfer the contents to the new host:

**Note**

Skip this step if version is lower than **`etcd-2.3.7-4`** or if etcd database size is smaller than 700 MB.

If the etcd backup is larger than 700 MB, prune the resource, or clear transistor to transistor logic (TTL) data, such as events. If the backup is still larger than 700 MB, stop the other hosts before performing this step.

```
# export NODE_ID="<NEW_NODE_ID>"
# etcdctl backup --keep-cluster-id --node-id ${NODE_ID} \
  --data-dir /var/lib/etcd --backup-dir
/var/lib/etcd/$NEW_ETCD-backup
# tar -cvf $NEW_ETCD-backup.tar.gz -C
/var/lib/etcd/$NEW_ETCD-backup/ .
# scp $NEW_ETCD-backup.tar.gz $NEW_ETCD:/var/lib/etcd/
```

6. On the new host, extract the backup data and set the permissions:

```
# tar -xf /etc/etcd/<NEW_ETCD_HOSTNAME> -C /etc/etcd/ --overwrite
# chown etcd:etcd /etc/etcd/*

# rm -rf /var/lib/etcd/member
# tar -xf /var/lib/etcd/<NEW_ETCD_HOSTNAME> -C /var/lib/etcd/
# chown -R etcd:etcd /var/lib/etcd/
```

7. On the new etcd host’s ***etcd.conf*** file:

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

      - ETCD_NAME
      - ETCD_INITIAL_CLUSTER
      - ETCD_INITIAL_CLUSTER_STATE

      Replace the IP address with the "NEW_ETCD" value for:

      - ETCD_LISTEN_PEER_URLS
      - ETCD_LISTEN_CLIENT_URLS
      - ETCD_INITIAL_ADVERTISE_PEER_URLS
      - ETCD_ADVERTISE_CLIENT_URLS

      For replacing failed hosts, you will need to remove the failed hosts from the etcd configuration.

8. Start etcd on the new host:

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

9. To verify that the new host has been added successfully:

```
etcdctl -C https://${ETCD_CA_HOST}:2379 --ca-
file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key cluster-health
```

## 29.6. PROJECT BACKUP

A future release of OpenShift Container Platform will feature specific support for per-project back up and restore.

For now, to back up API objects at the project level, use **`oc export`** for each object to be saved. For example, to save the deployment configuration **`frontend`** in YAML format:

```
$ oc export dc frontend -o yaml > dc-frontend.yaml
```

To back up all of the project (with the exception of cluster objects like namespaces and projects):

```
$ oc export all -o yaml > project.yaml
```

### 29.6.1. Role Bindings

Sometimes custom policy role bindings are used in a project. For example, a project administrator can give another user a certain role in the project and grant that user project access.

These role bindings can be exported:

```
$ oc get rolebindings -o yaml --export=true > rolebindings.yaml
```

### 29.6.2. Service Accounts

If custom service accounts are created in a project, these need to be exported:

```
$ oc get serviceaccount -o yaml --export=true > serviceaccount.yaml
```

### 29.6.3. Secrets

Custom secrets like source control management secrets (SSH Public Keys, Username/Password) should be exported if they are used:

```
$ oc get secret -o yaml --export=true > secret.yaml
```

### 29.6.4. Persistent Volume Claims

If the an application within a project uses a persistent volume through a persistent volume claim (PVC), these should be backed up:

```
$ oc get pvc -o yaml --export=true > pvc.yaml
```

## 29.7. PROJECT RESTORE

To restore a project, recreate the project and recreate all all of the objects that were exported during the backup:

```
$ oc new-project myproject
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml
```

**Note**

Some resources can fail to be created (for example, pods and default service accounts).

## 29.8. APPLICATION DATA BACKUP

In many cases, application data can be backed up using the **`oc rsync`** command, assuming **`rsync`** is installed within the container image. The Red Hat **rhel7** base image does contain **`rsync`**. Therefore, all images that are based on **rhel7** contain it as well.

**Warning**

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

Other means of backup may exist depending on the type of the persistent volume (for example, Cinder, NFS, Gluster, or others).

The paths to back up are also *application specific*. You can determine what path to back up by looking at the **`mountPath`** for volumes in the **`deploymentconfig`**.

### Example of Backing up a Jenkins Deployment’s Application Data

1. Get the application data **`mountPath`** from the **`deploymentconfig`**:

```
$ oc export dc/jenkins|grep mountPath
        - mountPath: /var/lib/jenkins
```

2. Get the name of the pod that is currently running:

```
$ oc get po --selector=deploymentconfig=jenkins
NAME              READY     STATUS    RESTARTS   AGE
jenkins-1-a3347   1/1       Running   0          18h
```

3. Use the **`oc rsync`** command to copy application data:

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

**Note**

This type of application data backup can only be performed while an application pod is currently running.

## 29.9. APPLICATION DATA RESTORE

The process for restoring application data is similar to the application backup procedure using the **`oc rsync`** tool. The same restrictions apply and the process of restoring application data requires a persistent volume.

### Example of Restoring a Jenkins Deployment's Application Data

1. Verify the backup:

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

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

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

**Note**

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

3. Restart the application with new data (*optional*):

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

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

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

# CHAPTER 30. TROUBLESHOOTING OPENSHIFT SDN

## 30.1. OVERVIEW

As described in the SDN documentation there are multiple layers of interfaces that are created to correctly pass the traffic from one container to another. In order to debug connectivity issues, you have to test the different layers of the stack to work out where the problem arises. This guide will help you dig down through the layers to identify the problem and how to fix it.

Part of the problem is that OpenShift Container Platform can be set up many ways, and the networking can be wrong in a few different places. So this document will work through some scenarios that, hopefully, will cover the majority of cases. If your problem is not covered, the tools and concepts that are introduced should help guide debugging efforts.

## 30.2. NOMENCLATURE

**Cluster**

The set of machines in the cluster. *i.e.* the Masters and the Nodes.

**Master**

A controller of the OpenShift Container Platform cluster. Note that the master may not be a node in the cluster, and thus, may not have IP connectivity to the pods.

**Node**

Host in the cluster running OpenShift Container Platform that can host pods.

**Pod**

Group of containers running on a node, managed by OpenShift Container Platform.

**Service**

Abstraction that presents a unified network interface that is backed by one or more pods.

**Router**

A web proxy that can map various URLs and paths into OpenShift Container Platform services to allow external traffic to travel into the cluster.

**Node Address**

The IP address of a node. This is assigned and managed by the owner of the network to which the node is attached. Must be reachable from any node in the cluster (master and client).

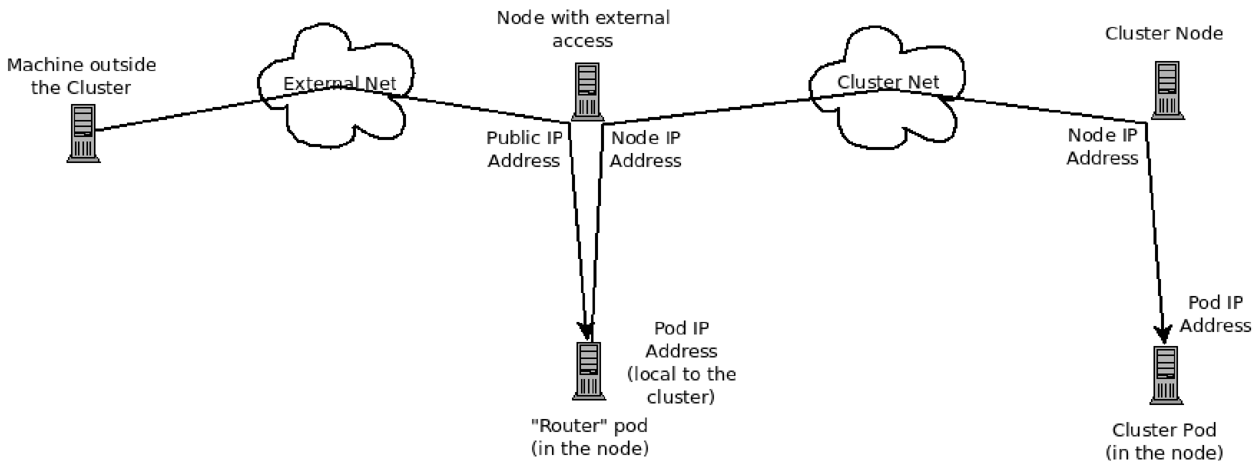
**Pod Address**

The IP address of a pod. These are assigned and managed by OpenShift Container Platform. By default they are assigned out of the 10.128.0.0/14 network (or, in older versions, 10.1.0.0/16). Only reachable from the client nodes.

**Service Address**

An IP address that represents the service, and is mapped to a pod address internally. These are assigned and managed by OpenShift Container Platform. By default they are assigned out of the 172.30.0.0/16 network. Only reachable from the client nodes.

The following diagram shows all of the pieces involved with external access.


Machine outside the Cluster
External Net
Node with external access
Public IP Address
Node IP Address
Cluster Net
Cluster Node
Node IP Address
Pod IP Address (local to the cluster)
"Router" pod (in the node)
Pod IP Address
Cluster Pod (in the node)

## 30.3. DEBUGGING EXTERNAL ACCESS TO AN HTTP SERVICE

If you are on an machine outside the cluster and are trying to access a resource provided by the cluster there needs to be a process running in a pod that listens on a public IP address and "routes" that traffic inside the cluster. The OpenShift Container Platform router serves that purpose for HTTP, HTTPS (with SNI), WebSockets, or TLS (with SNI).

Assuming you can't access an HTTP service from the outside of the cluster, let's start by reproducing the problem on the command line of the machine where things are failing. Try:

```
curl -kv http://foo.example.com:8000/bar    # But replace the argument
with your URL
```

If that works, are you reproducing the bug from the right place? It is also possible that the service has some pods that work, and some that don't. So jump ahead to the Section 30.4, "Debugging the Router" section.

If that failed, then let's resolve the DNS name to an IP address (assuming it isn't already one):

```
dig +short foo.example.com                  # But replace the hostname
with yours
```

If that doesn't give back an IP address, it's time to troubleshoot DNS, but that's outside the scope of this guide.

**Important**

Make sure that the IP address that you got back is one that you expect to be running the router. If it's not, fix your DNS.

Next, use **`ping -c`** ***`address`*** and **`tracepath`** ***`address`*** to check that you can reach the router host. It is possible that they will not respond to ICMP packets, in which case those tests will fail, but the router machine may be reachable. In which case, try using the telnet command to access the port for the router directly:

```
telnet 1.2.3.4 8000
```

You may get:

```
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^]'.
```

If so, there's something listening on the port on the IP address. That's good. Hit **`ctrl-]`** then hit the *enter* key and then type **`close`** to quit telnet. Move on to the Section 30.4, "Debugging the Router" section to check other things on the router.

Or you could get:

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection refused
```

Which tells us that the router is not listening on that port. Please see the Section 30.4, "Debugging the Router" section for more pointers on how to configure the router.

Or if you see:

```
Trying 1.2.3.4...
  telnet: connect to address 1.2.3.4: Connection timed out
```

Which tells us that you can't talk to anything on that IP address. Check your routing, firewalls, and that you have a router listening on that IP address. To debug the router, see the Section 30.4, "Debugging the Router" section. For IP routing and firewall issues, debugging that is beyond the purview of this guide.

## 30.4. DEBUGGING THE ROUTER

Now that you have an IP address, we need to **ssh** to that machine and check that the router software is running on that machine and configured correctly. So let's **ssh** there and get administrative OpenShift Container Platform credentials.

**Note**

If you have access to administrator credentials but are no longer logged in as the default system user **system:admin**, you can log back in as this user at any time as long as the credentials are still present in your CLI configuration file. The following command logs in and switches to the **default** project:

```
$ oc login -u system:admin -n default
```

Check that the router is running:

```
# oc get endpoints --namespace=default --selector=router
NAMESPACE    NAME                 ENDPOINTS
default      router               10.128.0.4:80
```

If that command fails, then your OpenShift Container Platform configuration is broken. Fixing that is outside the scope of this document.

You should see one or more router endpoints listed, but that won't tell you if they are running on the machine with the given external IP address, since the endpoint IP address will be one of the pod addresses that is internal to the cluster. To get the list of router host IP addresses, run:

```
# oc get pods --all-namespaces --selector=router --template='{{range
.items}}HostIP: {{.status.hostIP}}   PodIP: {{.status.podIP}}{{end}}
{{"\n"}}'
HostIP: 192.168.122.202   PodIP: 10.128.0.4
```

You should see the host IP that corresponds to your external address. If you do not, please refer to the router documentation to configure the router pod to run on the right node (by setting the affinity correctly) or update your DNS to match the IP addresses where the routers are running.

At this point in the guide, you should be on a node, running your router pod, but you still cannot get the HTTP request to work. First we need to make sure that the router is mapping the external URL to the correct service, and if that works, we need to dig into that service to make sure that all endpoints are reachable.

Let's list all of the routes that OpenShift Container Platform knows about:

```
# oc get route --all-namespaces
NAME              HOST/PORT         PATH      SERVICE        LABELS
TLS TERMINATION
route-unsecured   www.example.com   /test     service-name
```

If the host name and path from your URL don't match anything in the list of returned routes, then you need to add a route. See the router documentation.

If your route is present, then you need to debug access to the endpoints. That's the same as if you were debugging problems with a service, so please continue on with the next Section 30.5, "Debugging a Service" section.

## 30.5. DEBUGGING A SERVICE

If you can't communicate with a service from inside the cluster (either because your services can't communicate directly, or because you are using the router and everything works until you get into the cluster) then you need to work out what endpoints are associated with a service and debug them.

First, let's get the services:

```
# oc get services --all-namespaces
NAMESPACE   NAME              LABELS
SELECTOR                  IP(S)            PORT(S)
default     docker-registry   docker-registry=default
docker-registry=default   172.30.243.225   5000/TCP
default     kubernetes        component=apiserver,provider=kubernetes
<none>                    172.30.0.1       443/TCP
default     router            router=router
router=router             172.30.213.8     80/TCP
```

You should see your service in the list. If not, then you need to define your service.

The IP addresses listed in the service output are the Kubernetes service IP addresses that Kubernetes will map to one of the pods that backs that service. So you should be able to talk to that IP address. But, unfortunately, even if you can, it doesn’t mean all pods are reachable; and if you can’t, it doesn’t mean all pods aren’t reachable. It just tells you the status of the *one* that kubeproxy hooked you up to.

Let’s test the service anyway. From one of your nodes:

```
curl -kv http://172.30.243.225:5000/bar                  # Replace the
argument with your service IP address and port
```

Then, let’s work out what pods are backing our service (replace **`docker-registry`** with the name of the broken service):

```
# oc get endpoints --selector=docker-registry
NAME                ENDPOINTS
docker-registry     10.128.2.2:5000
```

From this, we can see that there’s only one endpoint. So, if your service test succeeded, and the router test succeeded, then something really odd is going on. But if there’s more than one endpoint, or the service test failed, try the following *for each* endpoint. Once you identify what endpoints aren’t working, then proceed to the next section.

First, test each endpoint (change the URL to have the right endpoint IP, port, and path):

```
curl -kv http://10.128.2.2:5000/bar
```

If that works, great, try the next one. If it failed, make a note of it and we’ll work out why, in the next section.

If all of them failed, then it is possible that the local node is not working, jump to the Section 30.7, “Debugging Local Networking” section.

If all of them worked, then jump to the Section 30.11, “Debugging Kubernetes” section to work out why the service IP address isn’t working.

## 30.6. DEBUGGING NODE TO NODE NETWORKING

Using our list of non-working endpoints, we need to test connectivity to the node.

1. Make sure that all nodes have the expected IP addresses:

   ```
   # oc get hostsubnet
   NAME                    HOST                    HOST IP
   SUBNET
   rh71-os1.example.com    rh71-os1.example.com    192.168.122.46
   10.1.1.0/24
   rh71-os2.example.com    rh71-os2.example.com    192.168.122.18
   10.1.2.0/24
   rh71-os3.example.com    rh71-os3.example.com    192.168.122.202
   10.1.0.0/24
   ```

   If you are using DHCP they could have changed. Ensure the host names, IP addresses, and subnets match what you expect. If any node details have changed, use **`oc edit`**

**hostsubnet** to correct the entries.

2. After ensuring the node addresses and host names are correct, list the endpoint IPs and node IPs:

   ```
   # oc get pods --selector=docker-registry \
       --template='{{range .items}}HostIP: {{.status.hostIP}}
   PodIP: {{.status.podIP}}{{end}}{{"\n"}}'

   HostIP: 192.168.122.202   PodIP: 10.128.0.4
   ```

3. Find the endpoint IP address you made note of before and look for it in the **PodIP** entry, and find the corresponding **HostIP** address. Then test connectivity at the node host level using the address from **HostIP**:

   - **ping -c 3 <IP_address>**: No response could mean that an intermediate router is eating the ICMP traffic.
   - **tracepath <IP_address>**: Shows the IP route taken to the target, if ICMP packets are returned by all hops.

     If both **tracepath** and **ping** fail, then look for connectivity issues with your local or virtual network.

4. For local networking, check the following:

   - Check the route the packet takes out of the box to the target address:

     ```
     # ip route get 192.168.122.202
       192.168.122.202 dev ens3  src 192.168.122.46
         cache
     ```

     In the above example, it will go out the interface named **ens3** with the source address of **192.168.122.46** and go directly to the target. If that is what you expected, use **ip a show dev ens3** to get the interface details and make sure that is the expected interface.

     An alternate result may be the following:

     ```
     # ip route get 192.168.122.202
       1.2.3.4 via 192.168.122.1 dev ens3  src 192.168.122.46
     ```

     It will pass through the **via** IP value to route appropriately. Ensure that the traffic is routing correctly. Debugging route traffic is beyond the scope of this guide.

Other debugging options for node to node networking can be solved with the following:

- Do you have ethernet link on both ends? Look for **Link detected: yes** in the output from **ethtool <network_interface>**.
- Are your duplex settings, and ethernet speeds right on both ends? Look through the rest of the **ethtool <network_interface>** information.
- Are the cables plugged in correctly? To the correct ports?

- Are the switches configured correctly?

Once you have ascertained that the node to node connectivity is fine, we need to look at the SDN configuration on both ends.

## 30.7. DEBUGGING LOCAL NETWORKING

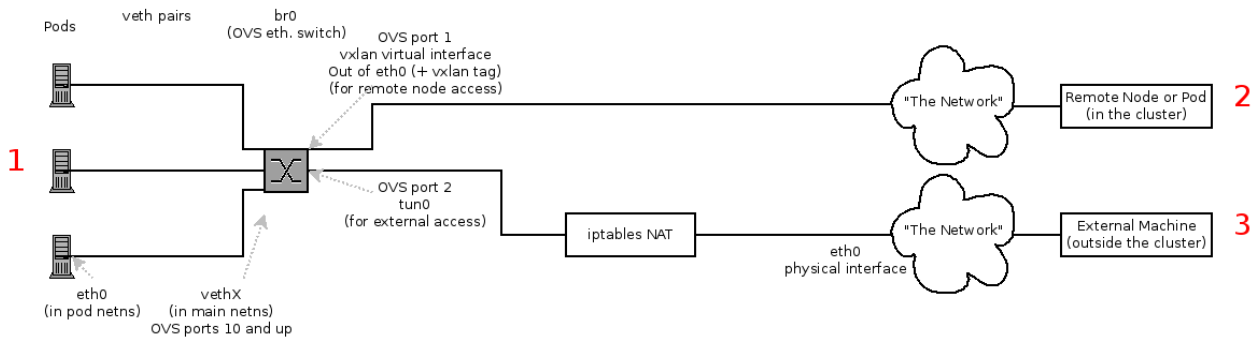
At this point we should have a list of one or more endpoints that you can't communicate with, but that have node to node connectivity. For each one, we need to work out what is wrong, but first you need to understand how the SDN sets up the networking on a node for the different pods.

### 30.7.1. The Interfaces on a Node

These are the interfaces that the OpenShift SDN creates:

- **`br0`**: The OVS bridge device that containers will be attached to. OpenShift SDN also configures a set of non-subnet-specific flow rules on this bridge.
- **`tun0`**: An OVS internal port (port 2 on **`br0`**). This gets assigned the cluster subnet gateway address, and is used for external network access. OpenShift SDN configures **`netfilter`** and routing rules to enable access from the cluster subnet to the external network via NAT.
- **`vxlan0`**: The OVS VXLAN device (port 1 on **`br0`**), which provides access to containers on remote nodes.
- **`vethX`** (in the main netns): A Linux virtual ethernet peer of **`eth0`** in the docker netns. It will be attached to the OVS bridge on one of the other ports.

### 30.7.2. SDN Flows Inside a Node



Depending on what you are trying to access (or be accessed from) the path will vary. There are four different places the SDN connects (inside a node). They are labeled in red on the diagram above.

- **Pod:** Traffic is going from one pod to another on the same machine (1 to a different 1)
- **Remote Node (or Pod):** Traffic is going from a local pod to a remote node or pod in the same cluster (1 to 2)
- **External Machine:** Traffic is going from a local pod outside the cluster (1 to 3)

Of course the opposite traffic flows are also possible.

### 30.7.3. Debugging Steps

#### 30.7.3.1. Is IP Forwarding Enabled?

Check that **`sysctl net.ipv4.ip_forward`** is set to 1 (and check the host if this is a VM)

#### 30.7.3.2. Are your routes correct?

Check the route tables with **`ip route`**:

```
# ip route
default via 192.168.122.1 dev ens3
10.128.0.0/14 dev tun0  proto kernel  scope link
# This sends all pod traffic into OVS
10.128.2.0/23 dev tun0  proto kernel  scope link  src 10.128.2.1
# This is traffic going to local pods, overriding the above
169.254.0.0/16 dev ens3  scope link  metric 1002
# This is for Zeroconf (may not be present)
172.17.0.0/16 dev docker0  proto kernel  scope link  src 172.17.42.1
# Docker's private IPs... used only by things directly configured by
docker; not {product-title}
192.168.122.0/24 dev ens3  proto kernel  scope link  src 192.168.122.46
# The physical interface on the local subnet
```

You should see the 10.128.x.x lines (assuming you have your pod network set to the default range in your configuration). If you do not, check the OpenShift Container Platform logs (see the Section 30.10, “Reading the Logs” section)

### 30.7.4. Is the Open vSwitch configured correctly?

Check the Open vSwitch bridges on both sides:

```
# ovs-vsctl list-br
br0
```

This should be **`br0`**.

You can list all of the ports that ovs knows about:

```
# ovs-ofctl -O OpenFlow13 dump-ports-desc br0
OFPST_PORT_DESC reply (OF1.3) (xid=0x2):
 1(vxlan0): addr:9e:f1:7d:4d:19:4f
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 2(tun0): addr:6a:ef:90:24:a3:11
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 8(vethe19c6ea): addr:1e:79:f3:a0:e8:8c
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(br0): addr:0a:7f:b4:33:c2:43
     config:     PORT_DOWN
     state:      LINK_DOWN
```

```
        speed: 0 Mbps now, 0 Mbps max
```

In particular, the **vethX** devices for all of the active pods should be listed as ports.

Next, list the flows that are configured on that bridge:

```
# ovs-ofctl -O OpenFlow13 dump-flows br0
```

The results will vary slightly depending on whether you are using the **ovs-subnet** or **ovs-multitenant** plug-in, but there are certain general things you can look for:

1. Every remote node should have a flow matching **tun_src=<node_IP_address>** (for incoming VXLAN traffic from that node) and another flow including the action **set_field:<node_IP_address>->tun_dst** (for outgoing VXLAN traffic to that node).
2. Every local pod should have flows matching **arp_spa=<pod_IP_address>** and **arp_tpa=<pod_IP_address>** (for incoming and outgoing ARP traffic for that pod), and flows matching **nw_src=<pod_IP_address>** and **nw_dst=<pod_IP_address>** (for incoming and outgoing IP traffic for that pod).

If there are flows missing, please look in the Section 30.10, "Reading the Logs" section.

#### 30.7.4.1. Is the `iptables` configuration correct?

Check the output from **iptables-save** to make sure you are not filtering traffic. However, OpenShift Container Platform sets up iptables rules during normal operation, so do not be surprised to see entries there.

#### 30.7.4.2. Is your external network correct?

Check external firewalls, if any, allow traffic to the target address (this is site-dependent, and beyond the purview of this guide).

## 30.8. DEBUGGING VIRTUAL NETWORKING

### 30.8.1. Builds on a Virtual Network are Failing

If you are installing OpenShift Container Platform using a virtual network (for example, OpenStack), and a build is failing, the maximum transmission unit (MTU) of the target node host might not be compatible with the MTU of the primary network interface (for example, **eth0**).

For a build to complete successfully, the MTU of an SDN must be less than the eth0 network MTU in order to pass data to between node hosts.

1. Check the MTU of your network by running the **ip addr** command:

```
# ip addr
---
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:56:4c:11 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.0/24 brd 172.16.0.0 scope global dynamic eth0
       valid_lft 168sec preferred_lft 168sec
    inet6 fe80::f816:3eff:fe56:4c11/64 scope link
       valid_lft forever preferred_lft forever
---
```

The MTU of the above network is 1500.

2. The MTU in your node configuration must be lower than the network value. Check the **`mtu`** in the node configuration of the targeted node host:

```
# cat /etc/origin/node/node-config.yaml
...
networkConfig:
   mtu: 1450
   networkPluginName: company/openshift-ovs-subnet
...
```

In the above node configuration file, the **`mtu`** value is lower than the network MTU, so no configuration is needed. If the **`mtu`** value was higher, edit the file and lower the value to at least 50 units fewer than the MTU of the primary network interface, then restart the node service. This would allow larger packets of data to pass between nodes.

## 30.9. DEBUGGING POD EGRESS

If you are trying to access an external service from a pod, e.g.:

```
curl -kv github.com
```

Make sure that the DNS is resolving correctly:

```
dig +search +noall +answer github.com
```

That should return the IP address for the github server, but check that you got back the correct address. If you get back no address, or the address of one of your machines, then you may be matching the wildcard entry in yoir local DNS server.

To fix that, you either need to make sure that DNS server that has the wildcard entry is not listed as a **`nameserver`** in your **`/etc/resolv.conf`** *or* you need to make sure that the wildcard domain is not listed in the **`search`** list.

If the correct IP address was returned, then try the debugging advice listed above in Section 30.7, “Debugging Local Networking”. Your traffic should leave the Open vSwitch on port 2 to pass through the **`iptables`** rules, then out the route table normally.

## 30.10. READING THE LOGS

Run: **`journalctl -u atomic-openshift-node.service --boot | less`**

Look for the **`Output of setup script:`** line. Everything starting with '+' below that are the script steps. Look through that for obvious errors.

Following the script you should see lines with **`Output of adding table=0`**. Those are the OVS rules, and there should be no errors.

## 30.11. DEBUGGING KUBERNETES

Check **`iptables -t nat -L`** to make sure that the service is being NAT'd to the right port on the local machine for the **`kubeproxy`**.

**Warning**

This is all changing soon… Kubeproxy is being eliminated and replaced with an **`iptables`**-only solution.

## 30.12. FURTHER HELP

1. Run the script at https://raw.githubusercontent.com/openshift/openshift-sdn/master/hack/debug.sh on the master (or from another machine with access to the master) to generate useful debugging information.
2. When debugging IP failover problems, run the script at https://raw.githubusercontent.com/openshift/openshift-sdn/master/hack/ipf-debug.sh on the master (or from another machine with access to the master) to generate useful debugging information.

## 30.13. MISCELLANEOUS NOTES

### 30.13.1. Other clarifications on ingress

- Kube - declare a service as NodePort and it will claim that port on all machines in the cluster (on what interface?) and then route into kube-proxy and then to a backing pod. See http://kubernetes.io/v1.0/docs/user-guide/services.html#type-nodeport (some node must be accessible from outside)
- Kube - declare as a LoadBalancer and something *you* have to write does the rest
- OS/AE - Both use the router

### 30.13.2. TLS Handshake Timeout

When a pod fails to deploy, check its docker log for a TLS handshake timeout:

```
$ docker log <container_id>
...
[...] couldn't get deployment [...] TLS handshake timeout
...
```

This condition, and generally, errors in establishing a secure connection, may be caused by a large difference in the MTU values between tun0 and the primary interface (e.g., eth0), such as when tun0 MTU is 1500 and eth0 MTU is 9000 (jumbo frames).

### 30.13.3. Other debugging notes

- Peer interfaces (of a Linux virtual ethernet pair) can be determined with **`ethtool -S`** ***`ifname`***
- Driver type: **`ethtool -i`** ***`ifname`***

# CHAPTER 31. DIAGNOSTICS TOOL

## 31.1. OVERVIEW

The **`oc adm diagnostics`** command runs a series of checks for error conditions in the host or cluster. Specifically, it:

- Verifies that the default registry and router are running and correctly configured.
- Checks **`ClusterRoleBindings`** and **`ClusterRoles`** for consistency with base policy.
- Checks that all of the client configuration contexts are valid and can be connected to.
- Checks that SkyDNS is working properly and the pods have SDN connectivity.
- Validates master and node configuration on the host.
- Checks that nodes are running and available.
- Analyzes host logs for known errors.
- Checks that systemd units are configured as expected for the host.

## 31.2. USING THE DIAGNOSTICS TOOL

OpenShift Container Platform can be deployed in many ways: built from source, included in a VM image, in a container image, or as enterprise RPMs. Each method implies a different configuration and environment. To minimize environment assumptions, the diagnostics were added to the **`openshift`** binary so that wherever there is an OpenShift Container Platform server or client, the diagnostics can run in the exact same environment.

To use the diagnostics tool, preferably on a master host and as cluster administrator, run:

```
$ oc adm diagnostics
```

This runs all available diagnostics, skipping any that do not apply. For example, the **NodeConfigCheck** does not run unless a node configuration is available. You can also run specific diagnostics by name as you work to address issues. For example:

```
$ oc adm diagnostics NodeConfigCheck UnitStatus
```

Diagnostics look for configuration files in standard locations:

- Client:
  - As indicated by the **`$KUBECONFIG`** environment variable variable
  - ***~/.kube/config file***
- Master:
  - ***/etc/origin/master/master-config.yaml***
- Node:
  - ***/etc/origin/node/node-config.yaml***

Non-standard locations can be specified with flags (respectively, **`--config`**, **`--master-config`**, and **`--node-config`**). If a configuration file is not found or specified, related diagnostics are skipped.

Consult the output with the **`--help`** flag for all available options.

## 31.3. RUNNING DIAGNOSTICS IN A SERVER ENVIRONMENT

Master and node diagnostics are most useful in a specific target environment, which is a deployment of RPMs with Ansible deployment logic. This provides some diagnostic benefits:

- Master and node configuration is based on a configuration file in a standard location.
- Systemd units are configured to manage the server(s).
- All components log to journald.

Having configuration files where Ansible places them means that you will generally not need to specify where to find them. Running **`oc adm diagnostics`** without flags will look for master and node configurations in the standard locations and use them if found; this should make the Ansible-installed use case as simple as possible. Also, it is easy to specify configuration files that are not in the expected locations:

```
$ oc adm diagnostics --master-config=<file_path> --node-config=
<file_path>
```

Systemd units and logs entries in journald are necessary for the current log diagnostic logic. For other deployment types, logs may be going into files, to stdout, or may combine node and master. At this time, for these situations, log diagnostics are not able to work properly and will be skipped.

## 31.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT

You may have access as an ordinary user, and/or as a **cluster-admin** user, and/or may be running on a host where OpenShift Container Platform master or node servers are operating. The diagnostics attempt to use as much access as the user has available.

A client with ordinary access should be able to diagnose its connection to the master and run a diagnostic pod. If multiple users or masters are configured, connections will be tested for all, but the diagnostic pod only runs against the current user, server, or project.

A client with **cluster-admin** access available (for any user, but only the current master) should be able to diagnose the status of infrastructure such as nodes, registry, and router. In each case, running **`oc adm diagnostics`** looks for the client configuration in its standard location and uses it if available.

# CHAPTER 32. IDLING APPLICATIONS

## 32.1. OVERVIEW

As an OpenShift Container Platform administrator, you can idle applications in order to reduce the consumption of resources.

Applications are made of services, as well as other scalable resources, such as deployment configurations. The action of idling an application involves idling all associated resources.

## 32.2. IDLING APPLICATIONS

Idling an application involves finding the scalable resources (deployment configurations, replication controllers, and others) associated with a service. Idling an application finds the service and marks it as idled, scaling down the resources to zero replicas.

You can use the **`oc idle`** command to idle a single service, or use the **`--resource-names-file`** option to idle multiple services.

### 32.2.1. Idling Single Services

Idle a single service with the following command:

```
$ oc idle <service>
```

### 32.2.2. Idling Multiple Services

Idle multiple services by creating a list of the desired services, then using the **`--resource-names-file`** option with the **`oc idle`** command.

This is helpful if an application spans across a set of services, or when idling multiples services in conjunction with a script in order to idle applications in bulk.

1. Create a file containing a list of the services, each on their own line.
2. Idle the services using the **`--resource-names-file`** option:

   ```
   $ oc idle --resource-names-file <filename>
   ```

## 32.3. UNIDLING APPLICATIONS

Application services become active again when they receive network traffic and will be scaled back up their previous state. This includes both traffic to the services and traffic passing through routes.

This works when using the default HAProxy router. You will need to configure other routers to detect idling to unidle these applications.

# CHAPTER 33. REVISION HISTORY: CLUSTER ADMINISTRATION

## 33.1. THU FEB 16 2017

| Affected Topic | Description of Change |
|---|---|
| Managing Pods | Added more information about disabling MAC filtering for the Egress router. |
| Aggregating Container Logs | Added a requirement that port 9300 be open for Elasticsearch to the Pre-deployment Configuration section. |
| High Availability | Added a step for deploying the ipfailover router to monitor postgresql listening to the Configuring a Highly-available Service section |

## 33.2. MON FEB 06 2017

| Affected Topic | Description of Change |
|---|---|
| Troubleshooting OpenShift SDN | Updated OpenShift SDN information. |
| Handling Out of Resource Errors | Added clarifying details around **`kubeletArguments`** in the Example Scenario section. |

## 33.3. WED JAN 25 2017

| Affected Topic | Description of Change |
|---|---|
| Monitoring Routers | Removed references to the deprecated **`--credentials`** option. |
| High Availability | Removed references to the deprecated **`--credentials`** option. |
| Pruning Objects | Added a NOTE box to the Pruning Images section explaining that pruning images with the **`--namespace`** flag does not remove images. |

| Affected Topic | Description of Change |
|---|---|
| Configuring the Cluster to Use Unique External IPs | Added a CAUTION box indicating that external IPs assigned to services of type **`LoadBalancer`** will always be in the range of **`ingressIPNetworkCIDR`** and updated Ingress CIDR references to the new default. |
| Managing Pods | Added clarifying details around **`EgressNetworkPolicy`**, including that it only affects external traffic (not pod-to-pod traffic) and that you can only have a single **`EgressNetworkPolicy`** in a namespace; added additional details about rule ordering and that breaking the rules can result in all egress traffic from the namespace being dropped. |

## 33.4. WED JAN 18 2017

OpenShift Container Platform 3.4 initial release.

| Affected Topic | Description of Change |
|---|---|
| Managing Pods | Updated the **`EgressNetworkPolicy`** example in the Configuring Pod Access Limits section. |
| | Added a new Setting Pod Disruption Budgets section, which allows the specification of safety constraints on pods during operations. |
| Setting Quotas | Added information for the **`requests.storage`** value and a ***storage-consumption.yaml*** example. |
| Setting Limit Ranges | Updated the PVC limit range within the Limit Range Object Definition example. |
| | Added the PersistentVolumeClaim Limits section. |
| Pruning Objects | Added a new Image Pruning Problems section, which describes possible issues with image pruning and how to avoid them with more generic tag naming. |
| Handling Out of Resource Errors | Added information about disk-based eviction. |