OpenShift Container Platform 4.2

Nodes

Configuring and managing nodes in OpenShift Container Platform 4.2

Last Updated: 2019-10-31

OpenShift Container Platform 4.2 Nodes

Configuring and managing nodes in OpenShift Container Platform 4.2

Legal Notice

Abstract

This document provides instructions for configuring and managing the nodes, Pods, and containers in your cluster. It also provides information on configuring Pod scheduling and placement, using jobs and DaemonSets to automate tasks, and other tasks to ensure an efficient cluster.

Table of Contents

CHAPTER 1. WORKING WITH PODS

1.1. USING PODS

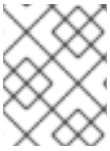
A *pod* is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

1.1.1. Understanding pods

Pods are the rough equivalent of a machine instance (physical or virtual) to a Container. Each pod is allocated its own internal IP address, therefore owning its entire port space, and Containers within pods can share their local storage and networking.

Pods have a lifecycle; they are defined, then they are assigned to run on a node, then they run until their Container(s) exit or they are removed for some other reason. Pods, depending on policy and exit code, might be removed after exiting, or can be retained in order to enable access to the logs of their Containers.

OpenShift Container Platform treats pods as largely immutable; changes cannot be made to a pod definition while it is running. OpenShift Container Platform implements changes by terminating an existing pod and recreating it with modified configuration, base image(s), or both. Pods are also treated as expendable, and do not maintain state when recreated. Therefore pods should usually be managed by higher-level controllers, rather than directly by users.



NOTE

For the maximum number of pods per OpenShift Container Platform node host, see the Cluster Limits.

WARNING

Bare pods that are not managed by a replication controller will be not rescheduled upon node disruption.

1.1.2. Example pod configurations

OpenShift Container Platform leverages the Kubernetes concept of a *pod*, which is one or more Containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

The following is an example definition of a pod that provides a long-running service, which is actually a part of the OpenShift Container Platform infrastructure: the integrated Container image registry. It demonstrates many features of pods, most of which are discussed in other topics and thus only briefly mentioned here:

Pod Object Definition (YAML)

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

```
metadata:
  name: example
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/example
  uid: 5cc30063-0265780783bc
  resourceVersion: '165032'
  creationTimestamp: '2019-02-13T20:31:37Z'
  labels:                                    (1)
    app: hello-openshift
  annotations:
    openshift.io/scc: anyuid
spec:
  restartPolicy: Always                      (2)
  serviceAccountName: default
  imagePullSecrets:
    - name: default-dockercfg-5zrhb
  priority: 0
  schedulerName: default-scheduler
  terminationGracePeriodSeconds: 30
  nodeName: ip-10-0-140-16.us-east-2.compute.internal
  securityContext:                           (3)
    seLinuxOptions:
      level: 's0:c11,c10'
  containers:                                (4)
    - resources: {}
      terminationMessagePath: /dev/termination-log
      name: hello-openshift
      securityContext:
        capabilities:
          drop:
            - MKNOD
        procMount: Default
      ports:
        - containerPort: 8080
          protocol: TCP
      imagePullPolicy: Always
      volumeMounts:                          (5)
        - name: default-token-wbqsl
          readOnly: true
          mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      terminationMessagePolicy: File
      image: registry.redhat.io/openshift4/ose-ogging-eventrouter:v4.2 (6)
  serviceAccount: default                    (7)
  volumes:                                   (8)
    - name: default-token-wbqsl
      secret:
        secretName: default-token-wbqsl
        defaultMode: 420
  dnsPolicy: ClusterFirst
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
```

```
    lastTransitionTime: '2019-02-13T20:31:37Z'
  - type: Ready
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: ContainersReady
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: PodScheduled
    status: 'True'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
hostIP: 10.0.140.16
startTime: '2019-02-13T20:31:37Z'
containerStatuses:
  - name: hello-openshift
    state:
      waiting:
        reason: ContainerCreating
    lastState: {}
    ready: false
    restartCount: 0
    image: openshift/hello-openshift
    imageID: ''
qosClass: BestEffort
```

1. Pods can be "tagged" with one or more labels, which can then be used to select and manage groups of pods in a single operation. The labels are stored in key/value format in the **metadata** hash. One label in this example is **registry=default**.

2. The pod restart policy with possible values **Always**, **OnFailure**, and **Never**. The default value is **Always**.

3. OpenShift Container Platform defines a security context for Containers which specifies whether they are allowed to run as privileged Containers, run as a user of their choice, and more. The default context is very restrictive but administrators can modify this as needed.

4. **containers** specifies an array of Container definitions; in this case (as with most), just one.

5. The Container specifies where external storage volumes should be mounted within the Container. In this case, there is a volume for storing the registry's data, and one for access to credentials the registry needs for making requests against the OpenShift Container Platform API.

6. Each Container in the pod is instantiated from its own Container image.

7. Pods making requests against the OpenShift Container Platform API is a common enough pattern that there is a **serviceAccount** field for specifying which service account user the pod should authenticate as when making the requests. This enables fine-grained access control for custom infrastructure components.

8. The pod defines storage volumes that are available to its Container(s) to use. In this case, it provides an ephemeral volume for the registry storage and a **secret** volume containing the service

account credentials.

NOTE

This pod definition does not include attributes that are filled by OpenShift Container Platform automatically after the pod is created and its lifecycle begins. The Kubernetes pod documentation has details about the functionality and purpose of pods.

1.2. VIEWING PODS

As an administrator, you can view the pods in your cluster and to determine the health of those pods and the cluster as a whole.

1.2.1. About pods

OpenShift Container Platform leverages the Kubernetes concept of a *pod*, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed. Pods are the rough equivalent of a machine instance (physical or virtual) to a container.

You can view a list of pods associated with a specific project or view usage statistics about pods.

1.2.2. Viewing pods in a project

You can view a list of pods associated with the current project, including the number of replica, the current status, number or restarts and the age of the pod.

Procedure

To view the pods in a project:

1. Change to the project:

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

2. Run the following command:

```
$ oc get pods
```

For example:

```
$ oc get pods -n openshift-console
NAME                  READY   STATUS    RESTARTS   AGE
console-698d866b78-bnshf   1/1     Running   2          165m
console-698d866b78-m87pm   1/1     Running   2          165m
```

Add the **-o wide** flags to view the pod IP address and the node where the pod is located.

```
$ oc get pods -o wide

NAME                  READY   STATUS    RESTARTS   AGE    IP           NODE
NOMINATED NODE
console-698d866b78-bnshf   1/1     Running   2          166m   10.128.0.24   ip-10-0-152-
```

```
71.ec2.internal    <none>
console-698d866b78-m87pm   1/1     Running   2          166m   10.129.0.23   ip-10-0-173-
237.ec2.internal   <none>
```

1.2.3. Viewing pod usage statistics

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

Prerequisites

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

Procedure

To view the usage statistics:

1. Run the following command:

```
$ oc adm top pods
```

For example:

```
$ oc adm top pods -n openshift-console
NAME                          CPU(cores)   MEMORY(bytes)
console-7f58c69899-q8c8k      0m           22Mi
console-7f58c69899-xhbgg      0m           25Mi
downloads-594fcccf94-bcxk8    3m           18Mi
downloads-594fcccf94-kv4p6    2m           15Mi
```

2. Run the following command to view the usage statistics for pods with labels:

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

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

1.3. CONFIGURING AN OPENSHIFT CONTAINER PLATFORM CLUSTER FOR PODS

As an administrator, you can create and maintain an efficient cluster for pods.

By keeping your cluster efficient, you can provide a better environment for your developers using such tools as what a pod does when it exits, ensuring that the required number of pods is always running, when to restart pods designed to run only once, limit the bandwidth available to pods, and how to keep pods running during disruptions.

1.3.1. Configuring how pods behave after restart

A pod restart policy determines how OpenShift Container Platform responds when Containers in that pod exit. The policy applies to all Containers in that pod.

The possible values are:

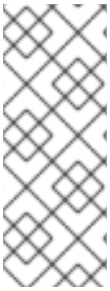
- **Always** - Tries restarting a successfully exited Container on the pod continuously, with an exponential back-off delay (10s, 20s, 40s) until the pod is restarted. The default is **Always**.
- **OnFailure** - Tries restarting a failed Container on the pod with an exponential back-off delay (10s, 20s, 40s) capped at 5 minutes.
- **Never** - Does not try to restart exited or failed Containers on the pod. Pods immediately fail and exit.

After the pod is bound to a node, the pod will never be bound to another node. This means that a controller is necessary in order for a pod to survive node failure:

Condition	Controller Type	Restart Policy
Pods that are expected to terminate (such as batch computations)	Job	**OnFailure** or **Never**
Pods that are expected to not terminate (such as web servers)	Replication Controller	**Always**.
Pods that must run one-per-machine	Daemonset	Any

If a Container on a pod fails and the restart policy is set to **OnFailure**, the pod stays on the node and the Container is restarted. If you do not want the Container to restart, use a restart policy of **Never**.

If an entire pod fails, OpenShift Container Platform starts a new pod. Developers must address the possibility that applications might be restarted in a new pod. In particular, applications must handle temporary files, locks, incomplete output, and so forth caused by previous runs.



NOTE

Kubernetes architecture expects reliable endpoints from cloud providers. When a cloud provider is down, the kubelet prevents OpenShift Container Platform from restarting.

If the underlying cloud provider endpoints are not reliable, do not install a cluster using cloud provider integration. Install the cluster as if it was in a no-cloud environment. It is not recommended to toggle cloud provider integration on or off in an installed cluster.

For details on how OpenShift Container Platform uses restart policy with failed Containers, see the Example States in the Kubernetes documentation.

1.3.2. 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.

Procedure

To limit the bandwidth on a pod:

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

   Limited Pod Object Definition

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

2. Create the pod using the object definition:

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

1.3.3. Understanding how to use pod disruption budgets to specify the number of pods that must be up

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.

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, either:
  - **minAvailable** is the number of pods must always be available, even during a disruption.
  - **maxUnavailable** is the number of Pods can be unavailable during a disruption.

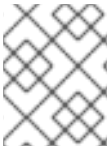
NOTE

A **maxUnavailable** of **0%or `0 or a minAvailable of `100%** or equal to the number of replicas, is permitted, but can block nodes from being drained.

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.



NOTE

Depending on your pod priority and preemption settings, lower-priority pods might be removed despite their pod disruption budget requirements.

1.3.3.1. Specifying the number of pods that must be up with pod disruption budgets

You can use a **PodDisruptionBudget** object to specify the minimum number or percentage of replicas that must be up at a time.

Procedure

To configure a pod disruption budget:

1. Create a YAML file with the an object definition similar to the following:

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

(1) **PodDisruptionBudget** is part of the **policy/v1beta1** API group.

(2) 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%**.

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

Or:

```
+
[source,yaml]
```

-

apiVersion: policy/v1beta1 <1> kind: PodDisruptionBudget metadata: name: my-pdb spec: maxUnavailable: 25% <2> selector: <3> matchLabels: foo: bar

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

1. Run the following command to add the object to project:

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

1.3.4. Preventing pod removal using critical pods

There are a number of core components that are critical to a fully functional cluster, but, run on a regular cluster node rather than the master. A cluster might stop working properly if a critical add-on is evicted.

Pods marked as critical are not allowed to be evicted.

,Procedure

To make a pod critical:

1. Create a pod specification or edit existing pods to include the **system-cluster-critical** priority class:

```
spec:
  template:
    metadata:
      name: critical-pod
    priorityClassName: system-cluster-critical 1
```

1 Default priority class for pods that should never be evicted from a node.

Alternatively, you can specify **system-node-critical** for pods that are important to the cluster but can be removed if necessary.

1. Create the pod:

```
$ oc create -f <file-name>.yaml
```

1.4. AUTOMATICALLY SCALING PODS

As a developer, you can use a horizontal pod autoscaler (HPA) to specify how OpenShift Container Platform should automatically increase or decrease the scale of a replication controller or deployment configuration, based on metrics collected from the pods that belong to that replication controller or deployment configuration.

1.4.1. Understanding horizontal pod autoscalers

You can create a horizontal pod autoscaler to specify the minimum and maximum number of pods you want to run, as well as the CPU utilization or memory utilization your pods should target.

IMPORTANT

Autoscaling for Memory Utilization is a Technology Preview feature only.

After you create a horizontal pod autoscaler, OpenShift Container Platform begins to query the CPU and/or memory resource metrics on the pods. When these metrics are available, the horizontal pod autoscaler computes the ratio of the current metric utilization with the desired metric utilization, and scales up or down accordingly. The query and scaling occurs at a regular interval, but can take one to two minutes before metrics become available.

For replication controllers, this scaling corresponds directly to the replicas of the replication controller. For deployment configurations, scaling corresponds directly to the replica count of the deployment configuration. Note that autoscaling applies only to the latest deployment in the **Complete** phase.

OpenShift Container Platform automatically accounts for resources and prevents unnecessary autoscaling during resource spikes, such as during start up. Pods in the **unready** state have **0 CPU** usage when scaling up and the autoscaler ignores the pods when scaling down. Pods without known metrics have **0% CPU** usage when scaling up and **100% CPU** when scaling down. This allows for more stability during the HPA decision. To use this feature, you must configure readiness checks to determine if a new pod is ready for use.

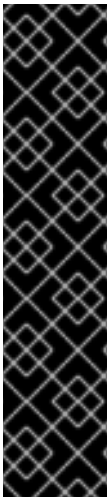
In order to use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics.

1.4.1.1. Supported metrics

The following metrics are supported by horizontal pod autoscalers:

Table 1.1. Metrics

Metric	Description	API version
CPU utilization	Number of CPU cores used. Can be used to calculate a percentage of the pod's requested CPU.	**autoscaling/v1**, **autoscaling/v2beta2**
Memory utilization	Amount of memory used. Can be used to calculate a percentage of the pod's requested memory.	**autoscaling/v2beta2**



IMPORTANT

For memory-based autoscaling, memory usage must increase and decrease proportionally to the replica count. On average:

- An increase in replica count must lead to an overall decrease in memory (working set) usage per-pod.
- A decrease in replica count must lead to an overall increase in per-pod memory usage.

Use the OpenShift Container Platform web console to check the memory behavior of your application and ensure that your application meets these requirements before using memory-based autoscaling.

1.4.2. Understanding how to create a horizontal pod autoscaler

How you create the horizontal pod autoscaler (HPA) depends on whether you want to scale for CPU or memory utilization.

CPU utilization

For CPU utilization:, you can create a horizontal pod autoscaler using the command line or by creating a **HorizontalPodAutoscaler** object.

When creating an HPA to control pod scaling based on CPU utilization, you specify the maximum number of pods you want to run at any given time. You can also specify a minimum number of pods.

The following command creates a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the **image-registry** DeploymentConfig to maintain an average CPU utilization of 50% across all Pods.

```
$ oc autoscale dc/image-registry --min 1 --max 10 --cpu-percent=50
```

The command creates the following object configuration:

Horizontal Pod Autoscaler Object Definition for CPU utilization

```
$ oc edit hpa image-registry
```

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  annotations:
    autoscaling.alpha.kubernetes.io/conditions:
'[{"type":"AbleToScale","status":"True","lastTransitionTime":"2019-05-
22T20:49:57Z","reason":"SucceededGetScale","message":"the
      HPA controller was able to get the target's current scale"},
{"type":"ScalingActive","status":"False","lastTransitionTime":"2019-05-
22T20:49:57Z","reason":"FailedGetResourceMetric","message":"the
      HPA was unable to compute the replica count: missing request for cpu"}]'
  creationTimestamp: 2019-05-22T20:49:42Z
  name: image-registry (1)
  namespace: default
  resourceVersion: "325215"
```

```
  selfLink: /apis/autoscaling/v1/namespaces/default/horizontalpodautoscalers/image-registry
  uid: 1fd7585a-7cd3-11e9-9d00-0e2a93384702
spec:
  maxReplicas: 10 2
  minReplicas: 1 3
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1
    kind: DeploymentConfig 4
    name: image-registry 5
  targetCPUUtilizationPercentage: 50 6
status:
  currentReplicas: 3
  desiredReplicas: 0
```

1. The name of this horizontal pod autoscaler object.
2. The lower limit for the number of pods that can be set by the autoscaler. If not specified or negative, the server will apply a default value.
3. The upper limit for the number of pods that can be set by the autoscaler. This value is required.
4. The kind of object to scale, DeploymentConfig or ReplicationController.
5. The name of the object to scale.
6. The percentage of the requested CPU that each pod should ideally be using.

Memory utilization

For memory utilization, you can specify the minimum number of pods and the average memory utilization your pods should target as well, otherwise those are given default values from the OpenShift Container Platform server.

You can specify resource metrics in terms of direct values, instead of as percentages of the requested value, by using a target type of **AverageValue** instead of **AverageUtilization**, and setting the corresponding **target.averageValue** field instead of the **target.averageUtilization**.

Horizontal Pod Autoscaler Object Definition for memory utilization

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale 1
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 2
    name: example 3
    kind: DeploymentConfig 4
  minReplicas: 1 5
  maxReplicas: 10 6
  metrics:
  - type: Resource
    resource:
      name: cpu
```

```
    target:
      type: Utilization 7
      averageUtilization: 50
```

1. The name of this horizontal pod autoscaler object.
2. The API version of the object to scale.
3. The name of the object to scale.
4. The kind of object to scale.
5. The lower limit for the number of pods that can be set by the autoscaler. If not specified or negative, the server will apply a default value.
6. The upper limit for the number of pods that can be set by the autoscaler. This value is required.
7. The type of must be either Utilization, Value, or AverageValue.

1.4.2.1. Creating a horizontal pod autoscaler for CPU utilization

You can create a horizontal pod autoscaler (HPA) to automatically scale pods when CPU usage exceeds a specified percentage. You create the HPA for a replication controller or deployment controller, based on how your pods were created.

Prerequisites

In order to use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

```
Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Timestamp:             2019-05-23T18:47:56Z
Window:                1m0s
Events:                <none>
```

Procedure

- Use one of the following commands to create a horizontal pod autoscaler for CPU utilization for a deployment controller or a replication controller:

```
oc autoscale dc/<deployment-name> \ 1
  --min <number> \ 2
  --max <number> \ 3
  --cpu-percent=<percent> 4

oc autoscale rc/<file-name> --min <number> --max <number> --cpu-percent=<percent>
```

1. Specify the deployment object or replica file.
2. Specify the minimum number of replicas when scaling down.
3. Specify the maximum number of replicas when scaling up.
4. Specify the target average CPU utilization, represented as a percent of requested CPU, over all the pods. If not specified or negative, a default autoscaling policy will be used.

For example:

```
oc autoscale dc/example --min=5 --max=7 --cpu-percent=75
```

The following example shows autoscaling for the **example** deployment configuration. The initial deployment requires 3 pods. The HPA object increased that minumum to 5 and will increase the pods up to 7 if CPU usage on the pods reaches 75%:

```
$ oc get dc example
NAME      REVISION   DESIRED   CURRENT   TRIGGERED BY
example   1          3         3         config

$ oc autoscale dc/example --min=5 --max=7 --cpu-percent=75
horizontalpodautoscaler.autoscaling/example autoscaled

$ oc get dc
NAME      REVISION   DESIRED   CURRENT   TRIGGERED BY
example   1          5         5         config
```

1.4.2.2. Creating a horizontal pod autoscaler object for memory utilization

You can create a horizontal pod autoscaler to automatically scale pods in a Deployment when memory usage exceeds a specified limit.

IMPORTANT

Autoscaling for memory utilization is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information on Red Hat Technology Preview features support scope, see https://access.redhat.com/support/offerings/techpreview/.

Prerequisites

In order to use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

```
Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-
kube-scheduler-ip-10-0-135-131.ec2.internal
Timestamp:             2019-05-23T18:47:56Z
Window:                1m0s
Events:                <none>
```

Procedure

To create a horizontal pod autoscaler for memory utilization:

1. Create a YAML file that contains one of the following:

Sample HPA object for an absolute value

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale (1)
```

```
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (2)
    name: example (3)
    kind: DeploymentConfig (4)
  minReplicas: 1 (5)
  maxReplicas: 10 (6)
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        name: memory-absolute
        targetAverageValue: 500Mi (7)
```

1. Specify the name of this horizontal pod autoscaler object.
2. Specify **apps/v1** as the API version of the object to scale.
3. Specify the name of the object to scale.
4. Specify the kind of object to scale.
5. Specify the minimum number of replicas when scaling down.
6. Specify the maximum number of replicas when scaling up.
7. Specify the average amount of memory used per pod.

Sample HPA object for a percentage

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale (1)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (2)
    name: example (3)
    kind: DeploymentConfig (4)
  minReplicas: 1 (5)
  maxReplicas: 10 (6)
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        name: memory-percent
        type: Utilization
        averageUtilization: 50 (7)
```

1. Specify the name of this horizontal pod autoscaler object.
2. Specify **apps/v1** as the API version of the object to scale.
3. Specify the name of the object to scale.
4. Specify the kind of object to scale.
5. Specify the minimum number of replicas when scaling down.
6. Specify the maximum number of replicas when scaling up.
7. The average percentage of the requested memory that each pod should be using.

1. Create the autoscaler from the above file:

```
$ oc create -f <file-name>.yaml
```

For example:

```
$ oc create -f hpa.yaml

horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

2. Verify that the HPA was created:

```
$ oc get hpa memory-autoscale

NAME              REFERENCE                TARGETS         MINPODS   MAXPODS
REPLICAS   AGE
memory-autoscale   DeploymentConfig/example   <unknown>/500Mi   1         10        0
56s
```

```
$ oc describe hpa memory-autoscale

Name:                                  memory-autoscale
Namespace:                               default
Labels:                                <none>
Annotations:                             <none>
CreationTimestamp:                             Wed, 22 May 2019 20:56:35 -0400
Reference:                             DeploymentConfig/example
Metrics:                               ( current / target )
  resource cpu on pods  (as a percentage of request):  <unknown>/500Mi
Min replicas:                            1
Max replicas:                            10
DeploymentConfig pods:                         0 current / 0 desired
Events:                                <none>
```

1.4.3. Understanding horizontal pod autoscaler status conditions

You can use the status conditions set to determine whether or not the horizontal pod autoscaler (HPA) is able to scale and whether or not it is currently restricted in any way.

The HPA status conditions are available with the **v2beta1** version of the autoscaling API.

The HPA responds with the following status conditions:

- The **AbleToScale** condition indicates whether HPA is able to fetch and update metrics, as well as whether any backoff-related conditions could prevent scaling.
  - A **True** condition indicates scaling is allowed.
  - A **False** condition indicates scaling is not allowed for the reason specified.
- The **ScalingActive** condition indicates whether the HPA is enabled (for example, the replica count of the target is not zero) and is able to calculate desired metrics.
  - A **True** condition indicates metrics is working properly.
  - A **False** condition generally indicates a problem with fetching metrics.
- The **ScalingLimited** condition indicates that the desired scale was capped by the maximum or minimum of the horizontal pod autoscaler.
  - A **True** condition indicates that you need to raise or lower the minimum or maximum replica count in order to scale.
  - A **False** condition indicates that the requested scaling is allowed.

```
$ oc describe hpa cm-test
Name:                         cm-test
Namespace:                    prom
Labels:                       <none>
Annotations:                  <none>
CreationTimestamp:            Fri, 16 Jun 2017 18:09:22 +0000
Reference:                    ReplicationController/cm-test
Metrics:                      ( current / target )
  "http_requests" on pods:    66m / 500m
Min replicas:                 1
Max replicas:                 4
ReplicationController pods:   1 current / 1 desired
Conditions: 1
  Type            Status    Reason              Message
  ----            ------    ------              -------
  AbleToScale     True      ReadyForNewScale    the last scale time was sufficiently old
as to warrant a new scale
  ScalingActive   True      ValidMetricFound    the HPA was able to successfully
calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the
acceptable range
Events:
```

1 The horizontal pod autoscaler status messages.

The following is an example of a pod that is unable to scale:

```
Conditions:
  Type         Status  Reason          Message
  ----         ------  ------          -------
  AbleToScale  False   FailedGetScale  the HPA controller was unable to get the target's current
scale: no matches for kind "ReplicationController" in group "apps"
```

```
Events:
  Type     Reason          Age              From                       Message
  ----     ------          ----             ----                       -------
  Warning  FailedGetScale  6s (x3 over 36s)  horizontal-pod-autoscaler  no matches for kind
"ReplicationController" in group "apps"
```

The following is an example of a pod that could not obtain the needed metrics for scaling:

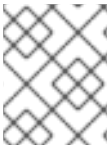
```
Conditions:
  Type                Status    Reason                   Message
  ----                ------    ------                   -------
  AbleToScale         True      SucceededGetScale        the HPA controller was able to get the target's
current scale
  ScalingActive       False     FailedGetResourceMetric  the HPA was unable to compute the replica
count: unable to get metrics for resource cpu: no metrics returned from heapster
```

The following is an example of a pod where the requested autoscaling was less than the required minimums:

```
Conditions:
  Type            Status    Reason            Message
  ----            ------    ------            -------
  AbleToScale     True      ReadyForNewScale  the last scale time was sufficiently old as to warrant
a new scale
  ScalingActive   True      ValidMetricFound  the HPA was able to successfully calculate a replica
count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable
range
```

1.4.3.1. Viewing horizontal pod autoscaler status conditions

You can view the status conditions set on a pod by the horizontal pod autoscaler (HPA).



NOTE

The horizontal pod autoscaler status conditions are available with the **v2beta1** version of the autoscaling API.

Prerequisites

In order to use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal

Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
```

```
  Usage:
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Timestamp:             2019-05-23T18:47:56Z
Window:                1m0s
Events:                <none>
```

Procedure

To view the status conditions on a pod, use the following command with the name of the pod:

```
$ oc describe hpa <pod-name>
```

For example:

```
$ oc describe hpa cm-test
```

The conditions appear in the **Conditions** field in the output.

```
Name:                         cm-test
Namespace:                    prom
Labels:                       <none>
Annotations:                  <none>
CreationTimestamp:            Fri, 16 Jun 2017 18:09:22 +0000
Reference:                    ReplicationController/cm-test
Metrics:                      ( current / target )
  "http_requests" on pods:    66m / 500m
Min replicas:                 1
Max replicas:                 4
ReplicationController pods:   1 current / 1 desired
Conditions: 1
  Type            Status    Reason              Message
  ----            ------    ------              -------
  AbleToScale     True      ReadyForNewScale    the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True      ValidMetricFound    the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable range
```

1.4.4. Additional resources

For more information on replication controllers and deployment controllers, see Understanding Deployments and DeploymentConfigs.

1.5. PROVIDING SENSITIVE DATA TO PODS

Some applications need sensitive information, such as passwords and user names, that you do not want developers to have.

As an administrator, you can use *Secret* objects to provide this information without exposing that information in clear text.

1.5.1. Understanding secrets

The **Secret** object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods. You can mount secrets into Containers using a volume plug-in or the system can use secrets to perform actions on behalf of a pod.

Key properties include:

- Secret data can be referenced independently from its definition.
- Secret data volumes are backed by temporary file-storage facilities (tmpfs) and never come to rest on a node.
- Secret data can be shared within a namespace.

YAML Secret Object Definition

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque (1)
data: (2)
  username: dmFsdWUtMQ0K (3)
  password: dmFsdWUtMg0KDQo=
stringData: (4)
  hostname: myapp.mydomain.com (5)
```

(1) (1) Indicates the structure of the secret's key names and values.

(2) The allowable format for the keys in the **data** field must meet the guidelines in the **DNS_SUBDOMAIN** value in the Kubernetes identifiers glossary.

(3) The value associated with keys in the **data** map must be base64 encoded.

(4) Entries in the **stringData** map are converted to base64 and the entry will then be moved to the **data** map automatically. This field is write-only; the value will only be returned via the **data** field.

(5) The value associated with keys in the **stringData** map is made up of plain text strings.

You must create a secret before creating the pods that depend on that secret.

When creating secrets:

- Create a secret object with secret data.
- Update the pod's service account to allow the reference to the secret.
- Create a pod, which consumes the secret as an environment variable or as a file (using a **secret** volume).

1.5.1.1. Types of secrets

The value in the **type** field indicates the structure of the secret's key names and values. The type can be used to enforce the presence of user names and keys in the secret object. If you do not want validation, use the **opaque** type, which is the default.

Specify one of the following types to trigger minimal server-side validation to ensure the presence of specific key names in the secret data:

- **kubernetes.io/service-account-token**. Uses a service account token.
- **kubernetes.io/basic-auth**. Use with Basic Authentication.
- **kubernetes.io/ssh-auth**. Use with SSH Key Authentication.
- **kubernetes.io/tls**. Use with TLS certificate authorities.

Specify **type: Opaque** if you do not want validation, which means the secret does not claim to conform to any convention for key names or values. An *opaque* secret, allows for unstructured **key:value** pairs that can contain arbitrary values.

NOTE

You can specify other arbitrary types, such as **example.com/my-secret-type**. These types are not enforced server-side, but indicate that the creator of the secret intended to conform to the key/value requirements of that type.

For examples of different secret types, see the code samples in *Using Secrets*.

1.5.1.2. Example secret configurations

The following are sample secret configuration files.

YAML Secret That Will Create Four Files

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: dmFsdWUtMQ0K      (1)
  password: dmFsdWUtMQ0KDQo=  (2)
stringData:
  hostname: myapp.mydomain.com  (3)
  secret.properties: |-         (4)
    property1=valueA
    property2=valueB
```

1. File contains decoded values.
2. File contains decoded values.
3. File contains the provided string.
4. File contains the provided data.

YAML of a Pod Populating Files in a Volume with Secret Data

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
          # name must match the volume name below
          - name: secret-volume
            mountPath: /etc/secret-volume
            readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never
```

YAML of a Pod Populating Environment Variables with Secret Data

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username
  restartPolicy: Never
```

YAML of a Build Config Populating Environment Variables with Secret Data

```
apiVersion: v1
kind: BuildConfig
metadata:
```

```
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username
```

1.5.1.3. Secret data keys

Secret keys must be in a DNS subdomain.

1.5.2. Understanding how to create secrets

As an administrator you must create a secret before developers can create the pods that depend on that secret.

When creating secrets:

- Create a secret object with secret data.
- Update the pod's service account to allow the reference to the secret.
- Create a pod, which consumes the secret as an environment variable or as a file (using a **secret** volume).

1.5.2.1. Secret creation restrictions

To use a secret, a pod needs to reference the secret. A secret can be used with a pod in three ways:

- To populate environment variables for Containers.
- As files in a volume mounted on one or more of its Containers.
- By kubelet when pulling images for the pod.

Volume type secrets write data into the Container as a file using the volume mechanism. Image pull secrets use service accounts for the automatic injection of the secret into all pods in a namespaces.

When a template contains a secret definition, the only way for the template to use the provided secret is to ensure that the secret volume sources are validated and that the specified object reference actually points to an object of type **Secret**. Therefore, a secret needs to be created before any pods that depend on it. The most effective way to ensure this is to have it get injected automatically through the use of a service account.

Secret API objects reside in a namespace. They can only be referenced by pods in that same namespace.

Individual secrets are limited to 1MB in size. This is to discourage the creation of large secrets that could exhaust apiserver and kubelet memory. However, creation of a number of smaller secrets could also exhaust memory.

1.5.2.2. Creating an opaque secret

As an administrator, you can create a opaque secret, which allows for unstructured **key:value** pairs that can contain arbitrary values.

Procedure

1. Create a secret object in a YAML file on master.
   For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque (1)
data:
  username: dXNlci1uYW1l
  password: cGFzc3dvcmQ=
```

(1) Specifies an opaque secret.

2. Use the following command to create a secret object:

```
$ oc create -f <filename>
```

Then:

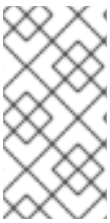
1. Update the service account for the pod where you want to use the secret to allow the reference to the secret.
2. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume).

1.5.3. Understanding how to update secrets

When you modify the value of a secret, the value (used by an already running pod) will not dynamically change. To change a secret, you must delete the original pod and create a new pod (perhaps with an identical PodSpec).

Updating a secret follows the same workflow as deploying a new Container image. You can use the **kubectl rolling-update** command.

The **resourceVersion** value in a secret is not specified when it is referenced. Therefore, if a secret is updated at the same time as pods are starting, then the version of the secret will be used for the pod will not be defined.



NOTE

Currently, it is not possible to check the resource version of a secret object that was used when a pod was created. It is planned that pods will report this information, so that a controller could restart ones using a old **resourceVersion**. In the interim, do not update the data of existing secrets, but create new ones with distinct names.

1.5.4. About using signed certificates with secrets

To secure communication to your service, you can configure OpenShift Container Platform to generate a signed serving certificate/key pair that you can add into a secret in a project.

A *service serving certificate secret* is intended to support complex middleware applications that need out-of-the-box certificates. It has the same settings as the server certificates generated by the administrator tooling for nodes and masters.

Service pod specification configured for a service serving certificates secret.

```
apiVersion: v1
  kind: Service
  metadata:
    name: registry
    annotations:
      service.alpha.openshift.io/serving-cert-secret-name: registry-cert 1
....
```

1 Specify the name for the certificate

Other pods can trust cluster-created certificates (which are only signed for internal DNS names), by using the CA bundle in the */var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt* file that is automatically mounted in their pod.

The signature algorithm for this feature is **x509.SHA256WithRSA**. To manually rotate, delete the generated secret. A new certificate is created.

1.5.4.1. Generating signed certificates for use with secrets

To use a signed serving certificate/key pair with a pod, create or edit the service to add the **service.alpha.openshift.io/serving-cert-secret-name** annotation, then add the secret to the pod.

Procedure

To create a *service serving certificate secret*:

1. Edit the pod specification for your service.
2. Add the **service.alpha.openshift.io/serving-cert-secret-name** annotation with the name you want to use for your secret.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  annotations:
     service.alpha.openshift.io/serving-cert-secret-name: my-cert 1
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

The certificate and key are in PEM format, stored in **tls.crt** and **tls.key** respectively.

3. Create the service:

```
$ oc create -f <file-name>.yaml
```

4. View the secret to make sure it was created:

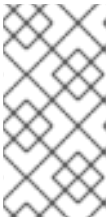
```
$ oc get secrets

NAME                  TYPE                                  DATA      AGE
my-cert               kubernetes.io/tls                     2         9m

$ oc describe secret my-service-pod
Name:         my-service-pod
Namespace:    openshift-console
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: builder
              kubernetes.io/service-account.uid: ab-11e9-988a-0eb4e1b4a396

Type:  kubernetes.io/service-account-token

Data

ca.crt:     5802 bytes
namespace:  17 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Ii
wia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJvcGVuc2hpZnQtY29uc
29sZSIsImt1YmVyb
cnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJhYmE4Y2UyZC00MzVlLTExZTktOTg4YS0wZWI0ZTFiNGEz
OTYiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6b3BlbnNoaWZ
```

5. Edit your pod specification with that secret.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
  volumes:
  - name: foo
    secret:
      secretName: my-cert
      items:
      - key: username
        path: my-group/my-username
        mode: 511
```

When it is available, your pod will run. The certificate will be good for the internal service DNS name, **<service.name>.<service.namespace>.svc**.

The certificate/key pair is automatically replaced when it gets close to expiration. View the expiration date in the **service.alpha.openshift.io/expiry** annotation on the secret, which is in RFC3339 format.



NOTE

In most cases, the service DNS name **<service.name>.<service.namespace>.svc** is not externally routable. The primary use of **<service.name>.<service.namespace>.svc** is for intracluster or intraservice communication, and with re-encrypt routes.

1.5.5. Troubleshooting secrets

If a service certificate generation fails with (service's **service.alpha.openshift.io/serving-cert-generation-error** annotation contains):

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

The service that generated the certificate no longer exists, or has a different **serviceUID**. You must force certificates regeneration by removing the old secret, and clearing the following annotations on the service **service.alpha.openshift.io/serving-cert-generation-error**, **service.alpha.openshift.io/serving-cert-generation-error-num**:

```
$ oc delete secret <secret_name>
$ oc annotate service <service_name> service.alpha.openshift.io/serving-cert-generation-error-1
$ oc annotate service <service_name> service.alpha.openshift.io/serving-cert-generation-error-num-1
```

NOTE

The command removing annotation has a **-** after the annotation name to be removed.

1.6. USING DEVICE PLUG-INS TO ACCESS EXTERNAL RESOUCES WITH PODS

Device plug-ins allow you to use a particular device type (GPU, InfiniBand, or other similar computing resources that require vendor-specific initialization and setup) in your OpenShift Container Platform pod without needing to write custom code.

1.6.1. Understanding device plug-ins

The device plug-in provides a consistent and portable solution to consume hardware devices across clusters. The device plug-in provides support for these devices through an extension mechanism, which makes these devices available to Containers, provides health checks of these devices, and securely shares them.

IMPORTANT

OpenShift Container Platform supports the device plug-in API, but the device plug-in Containers are supported by individual vendors.

A device plug-in is a gRPC service running on the nodes (external to the **kubelet**) that is responsible for managing specific hardware resources. Any device plug-in must support following remote procedure calls (RPCs):

```
service DevicePlugin {
    // GetDevicePluginOptions returns options to be communicated with Device
    // Manager
    rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

    // ListAndWatch returns a stream of List of Devices
    // Whenever a Device state change or a Device disappears, ListAndWatch
    // returns the new list
    rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

    // Allocate is called during container creation so that the Device
    // Plug-in can run device specific operations and instruct Kubelet
    // of the steps to make the Device available in the container
    rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

    // PreStartcontainer is called, if indicated by Device Plug-in during
    // registration phase, before each container start. Device plug-in
    // can run device specific operations such as reseting the device
    // before making devices available to the container
    rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

Example device plug-ins

- Nvidia GPU device plug-in for COS-based operating system
- Nvidia official GPU device plug-in
- Solarflare device plug-in
- KubeVirt device plug-ins: vfio and kvm

NOTE

For easy device plug-in reference implementation, there is a stub device plug-in in the Device Manager code: ***vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go***.

1.6.1.1. Methods for deploying a device plug-in

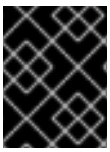
- Daemonsets are the recommended approach for device plug-in deployments.
- Upon start, the device plug-in will try to create a UNIX domain socket at ***/var/lib/kubelet/device-plugin/*** on the node to serve RPCs from Device Manager.

- Since device plug-ins must manage hardware resources, access to the host file system, as well as socket creation, they must be run in a privileged security context.
- More specific details regarding deployment steps can be found with each device plug-in implementation.

1.6.2. Understanding the Device Manager

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plug-ins known as device plug-ins.

You can advertise specialized hardware without requiring any upstream code changes.



IMPORTANT

OpenShift Container Platform supports the device plug-in API, but the device plug-in Containers are supported by individual vendors.

Device Manager advertises devices as **Extended Resources**. User pods can consume devices, advertised by Device Manager, using the same **Limit/Request** mechanism, which is used for requesting any other **Extended Resource**.

Upon start, the device plug-in registers itself with Device Manager invoking **Register** on the ***/var/lib/kubelet/device-plugins/kubelet.sock*** and starts a gRPC service at ***/var/lib/kubelet/device-plugins/<plugin>.sock*** for serving Device Manager requests.

Device Manager, while processing a new registration request, invokes **ListAndWatch** remote procedure call (RPC) at the device plug-in service. In response, Device Manager gets a list of **Device** objects from the plug-in over a gRPC stream. Device Manager will keep watching on the stream for new updates from the plug-in. On the plug-in side, the plug-in will also keep the stream open and whenever there is a change in the state of any of the devices, a new device list is sent to the Device Manager over the same streaming connection.

While handling a new pod admission request, Kubelet passes requested **Extended Resources** to the Device Manager for device allocation. Device Manager checks in its database to verify if a corresponding plug-in exists or not. If the plug-in exists and there are free allocatable devices as well as per local cache, **Allocate** RPC is invoked at that particular device plug-in.

Additionally, device plug-ins can also perform several other device-specific operations, such as driver installation, device initialization, and device resets. These functionalities vary from implementation to implementation.

1.6.3. Enabling Device Manager

Enable Device Manager to implement a device plug-in to advertise specialized hardware without any upstream code changes.

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plug-ins known as device plug-ins.

1. Obtain the label associated with the static Machine Config Pool CRD for the type of node you want to configure. Perform one of the following steps:
    a. View the Machine Config:

```
# oc describe machineconfig <name>
```

For example:

```
# oc describe machineconfig 00-worker

oc describe machineconfig 00-worker
Name:         00-worker
Namespace:
Labels:       machineconfiguration.openshift.io/role=worker 1
```

1 1 Label required for the device manager.

Procedure

1. Create a Custom Resource (CR) for your configuration change.

   Sample configuration for a Device Manager CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr 2
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true 3
```

   1 Assign a name to CR.

   2 Enter the label from the Machine Config Pool.

   3 Set **DevicePlugins** to 'true`.

2. Create the device manager:

```
$ oc create -f devicemgr.yaml

kube
letconfig.machineconfiguration.openshift.io/devicemgr created
```

3. Ensure that Device Manager was actually enabled by confirming that */var/lib/kubelet/device-plugins/kubelet.sock* is created on the node. This is the UNIX domain socket on which the Device Manager gRPC server listens for new plug-in registrations. This sock file is created when the Kubelet is started only if Device Manager is enabled.

1.7. INCLUDING POD PRIORITY IN POD SCHEDULING DECISIONS

You can enable pod priority and preemption in your cluster. Pod priority indicates the importance of a pod relative to other pods and queues the pods based on that priority. Pod preemption allows the cluster to evict, or preempt, lower-priority pods so that higher-priority pods can be scheduled if there is no available space on a suitable node Pod priority also affects the scheduling order of pods and out-of-resource eviction ordering on the node.

To use priority and preemption, you create priority classes that define the relative weight of your pods. Then, reference a priority class in the pod specification to apply that weight for scheduling.

Preemption is controlled by the **disablePreemption** parameter in the scheduler configuration file, which is set to **false** by default.

1.7.1. Understanding pod priority

When you use the Pod Priority and Preemption feature, the scheduler orders pending pods by their priority, and a pending pod is placed ahead of other pending pods with lower priority in the scheduling queue. As a result, the higher priority pod might be scheduled sooner than pods with lower priority if its scheduling requirements are met. If a pod cannot be scheduled, scheduler continues to schedule other lower priority pods.

1.7.1.1. Pod priority classes

You can assign pods a priority class, which is a non-namespaced object that defines a mapping from a name to the integer value of the priority. The higher the value, the higher the priority.

A priority class object can take any 32-bit integer value smaller than or equal to 1000000000 (one billion). Reserve numbers larger than one billion for critical pods that should not be preempted or evicted. By default, OpenShift Container Platform has two reserved priority classes for critical system pods to have guaranteed scheduling.

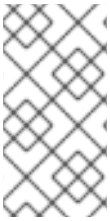
```
$ oc get priorityclasses
NAME                     CREATED AT
cluster-logging          2019-03-13T14:45:12Z
system-cluster-critical  2019-03-13T14:01:10Z
system-node-critical     2019-03-13T14:01:10Z
```

- **system-node-critical** - This priority class has a value of 2000001000 and is used for all pods that should never be evicted from a node. Examples of pods that have this priority class are **sdn-ovs**, **sdn**, and so forth. A number of critical components include the **system-node-critical** priority class by default, for example:
  - master-api
  - master-controller
  - master-etcd
  - sdn
  - sdn-ovs
  - sync
- **system-cluster-critical** - This priority class has a value of 2000000000 (two billion) and is used with pods that are important for the cluster. Pods with this priority class can be evicted from a node in certain circumstances. For example, pods configured with the **system-node-**

critical priority class can take priority. However, this priority class does ensure guaranteed scheduling. Examples of pods that can have this priority class are fluentd, add-on components like descheduler, and so forth. A number of critical components include the **system-cluster-critical** priority class by default, for example:

  - fluentd
  - metrics-server
  - descheduler

- **cluster-logging** - This priority is used by Fluentd to make sure Fluentd pods are scheduled to nodes over other apps.



NOTE

If you upgrade your existing cluster, the priority of your existing pods is effectively zero. However, existing pods with the **scheduler.alpha.kubernetes.io/critical-pod** annotation are automatically converted to **system-cluster-critical** class. Fluentd cluster logging pods with the annotation are converted to the **cluster-logging** priority class.

1.7.1.2. Pod priority names

After you have one or more priority classes, you can create pods that specify a priority class name in a pod specification. The priority admission controller uses the priority class name field to populate the integer value of the priority. If the named priority class is not found, the pod is rejected.

1.7.2. Understanding pod preemption

When a developer creates a pod, the pod goes into a queue. If the developer configured the pod for pod priority or preemption, the scheduler picks a pod from the queue and tries to schedule the pod on a node. If the scheduler cannot find space on an appropriate node that satisfies all the specified requirements of the pod, preemption logic is triggered for the pending pod.

When the scheduler preempts one or more pods on a node, the **nominatedNodeName** field of higher-priority pod specification is set to the name of the node, along with the **nodename** field. The scheduler uses the **nominatedNodeName** field to keep track of the resources reserved for pods and also provides information to the user about preemptions in the clusters.

After the scheduler preempts a lower-priority pod, the scheduler honors the graceful termination period of the pod. If another node becomes available while scheduler is waiting for the lower-priority pod to terminate, the scheduler can schedule the higher-priority pod on that node. As a result, the **nominatedNodeName** field and **nodeName** field of the pod specification might be different.

Also, if the scheduler preempts pods on a node and is waiting for termination, and a pod with a higher-priority pod than the pending pod needs to be scheduled, the scheduler can schedule the higher-priority pod instead. In such a case, the scheduler clears the **nominatedNodeName** of the pending pod, making the pod eligible for another node.

Preemption does not necessarily remove all lower-priority pods from a node. The scheduler can schedule a pending pod by removing a portion of the lower-priority pods.

The scheduler considers a node for pod preemption only if the pending pod can be scheduled on the node.

1.7.2.1. Pod preemption and other scheduler settings

If you enable pod priority and preemption, consider your other scheduler settings:

Pod priority and pod disruption budget

A pod disruption budget specifies the minimum number or percentage of replicas that must be up at a time. If you specify pod disruption budgets, OpenShift Container Platform respects them when preempting pods at a best effort level. The scheduler attempts to preempt pods without violating the pod disruption budget. If no such pods are found, lower-priority pods might be preempted despite their pod disruption budget requirements.

Pod priority and pod affinity

Pod affinity requires a new pod to be scheduled on the same node as other pods with the same label.

If a pending pod has inter-pod affinity with one or more of the lower-priority pods on a node, the scheduler cannot preempt the lower-priority pods without violating the affinity requirements. In this case, the scheduler looks for another node to schedule the pending pod. However, there is no guarantee that the scheduler can find an appropriate node and pending pod might not be scheduled.

To prevent this situation, carefully configure pod affinity with equal-priority pods.

1.7.2.2. Graceful termination of preempted pods

When preempting a pod, the scheduler waits for the pod graceful termination period to expire, allowing the pod to finish working and exit. If the pod does not exit after the period, the scheduler kills the pod. This graceful termination period creates a time gap between the point that the scheduler preempts the pod and the time when the pending pod can be scheduled on the node.

To minimize this gap, configure a small graceful termination period for lower-priority pods.

1.7.3. Configuring priority and preemption

You apply pod priority and preemption by creating a priority class object and associating pods to the priority using the **priorityClassName** in your pod specifications.

Sample priority class object

```
apiVersion: scheduling.k8s.io/v1beta1
kind: PriorityClass
metadata:
  name: high-priority (1)
value: 1000000 (2)
globalDefault: false (3)
description: "This priority class should be used for XYZ service pods only." (4)
```

1. The name of the priority class object.
2. The priority value of the object.
3. Optional field that indicates whether this priority class should be used for pods without a priority class name specified. This field is **false** by default. Only one priority class with **globalDefault** set to **true** can exist in the cluster. If there is no priority class with **globalDefault:true**, the priority of pods with no priority class name is zero. Adding a priority class with **globalDefault:true** affects only pods created after the priority class is added and does not change the priorities of existing pods.

4 Optional arbitrary text string that describes which pods developers should use with this priority class.

Procedure

To configure your cluster to use priority and preemption:

1. Create one or more priority classes:
   a. Specify a name and value for the priority.
   b. Optionally specify the **globalDefault** field in the priority class and a description.
2. Create a pod specification or edit existing pods to include the name of a priority class, similar to the following:

   Sample pod specification with priority class name

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority 1
```

   1 Specify the priority class to use with this pod.

3. Create the pod:

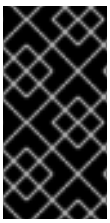
```
$ oc create -f <file-name>.yaml
```

   You can add the priority name directly to the pod configuration or to a pod template.

1.7.4. Disabling priority and preemption

You can disable the pod priority and preemption feature.

After the feature is disabled, the existing pods keep their priority fields, but preemption is disabled, and priority fields are ignored. If the feature is disabled, you cannot set a priority class name in new pods.



IMPORTANT

Critical pods rely on scheduler preemption to be scheduled when a cluster is under resource pressure. For this reason, Red Hat recommends not disabling preemption. DaemonSet pods are scheduled by the DaemonSet controller and not affected by disabling preemption.

Procedure

To disable the preemption for the cluster:

1. Edit the Scheduler Operator Custom Resource to add the **disablePreemption: true** parameter:

```
oc edit scheduler cluster
```

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: '2019-03-12T01:45:02Z'
  generation: 1
  name: example
  resourceVersion: '1882034'
  selfLink: /apis/config.openshift.io/v1/schedulers/example
  uid: 743701e9-4468-11e9-bd34-02a7fe1bf828
spec:
  disablePreemption: true
```

1.8. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS

A *node selector* specifies a map of key-value pairs. The rules are defined using custom labels on nodes and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the indicated key-value pairs as the label on the node.

If you are using node affinity and node selectors in the same pod configuration, see the important considerations below.

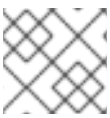
1.8.1. Using node selectors to control pod placement

You can use node selector labels on pods to control where the pod is scheduled.

With node selectors, OpenShift Container Platform schedules the pods on nodes that contain matching labels.

You can add labels to a node or MachineConfig, but the labels will not persist if the node or machine goes down. Adding the label to the MachineSet ensures that new nodes or machines will have the label.

To add node selectors to an existing pod, add a node selector to the controlling object for that node, such as a ReplicaSet, Daemonset, or StatefulSet. Any existing pods under that controlling object are recreated on a node with a matching label. If you are creating a new pod, you can add the node selector directly to the pod spec.



NOTE

You cannot add a node selector to an existing scheduled pod.

Prerequisites

If you want to add a node selector to existing pods, determine the controlling object for that pod. For exeample, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** ReplicaSet:

```
$ oc describe pod router-default-66d5cf9464-7pwkc

Name:             router-default-66d5cf9464-7pwkc
Namespace:        openshift-ingress

....

Controlled By:    ReplicaSet/router-default-66d5cf9464
```

The web console lists the controlling object under **ownerReferences** in the pod YAML:

```
ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: router-default-66d5cf9464
    uid: d81dd094-da26-11e9-a48a-128e7edf0312
    controller: true
    blockOwnerDeletion: true
```

Procedure

1. Add the desired label to your nodes:

```
$ oc label <resource> <name> <key>=<value>
```

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

The label is applied to the node:

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    beta.kubernetes.io/os: linux
    failure-domain.beta.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.2'
    node-role.kubernetes.io/worker: ''
    failure-domain.beta.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    beta.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    region: east (1)
    beta.kubernetes.io/arch: amd64
    type: user-node (2)
  ....
```

1 2 Specify the label(s) you will add to the node.

Alternatively, you can add the label to a MachineSet:

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c
```

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet

....

spec:
  replicas: 2
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: ci-ln-89dz2y2-d5d6b-4995x
      machine.openshift.io/cluster-api-machine-role: worker
      machine.openshift.io/cluster-api-machine-type: worker
      machine.openshift.io/cluster-api-machineset: ci-ln-89dz2y2-d5d6b-4995x-worker-us-east-
1a
  template:
    metadata:
      creationTimestamp: null
      labels:
        machine.openshift.io/cluster-api-cluster: ci-ln-89dz2y2-d5d6b-4995x
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: ci-ln-89dz2y2-d5d6b-4995x-worker-us-
east-1a
    spec:
      metadata:
        creationTimestamp: null
        labels:
          region: east 1
          type: user-node 2
....
```

1 2 Specify the label(s) you will add to the node.

2. Add the desired node selector a pod:

   - To add a node selector to existing and furture pods, add a node selector to the controlling object for the pods:
     For example:

     ```
     kind: ReplicaSet

     ....

     spec:

     ....
     ```

```
template:
  metadata:
    creationTimestamp: null
    labels:
      ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
      pod-template-hash: 66d5cf9464
  spec:
    nodeSelector:
      beta.kubernetes.io/os: linux
      node-role.kubernetes.io/worker: ''
      type: user-node 1
```

1 Add the desired node selector.

- For a new pod, you can add the selector to the pod specification directly:

```
apiVersion: v1
kind: Pod

...

spec:
  nodeSelector:
    <key>: <value>

...
```

For example:

```
apiVersion: v1
kind: Pod

....

spec:
  nodeSelector:
    region: east
    type: user-node
```

NOTE

If you are using node selectors and node affinity in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

CHAPTER 2. CONTROLLING POD PLACEMENT ONTO NODES (SCHEDULING)

2.1. CONTROLLING POD PLACEMENT USING THE SCHEDULER

Pod scheduling is an internal process that determines placement of new pods onto nodes within the cluster.

The scheduler code has a clean separation that watches new pods as they get created and identifies the most suitable node to host them. It then creates bindings (pod to node bindings) for the pods using the master API.

Default pod scheduling

OpenShift Container Platform comes with a default scheduler that serves the needs of most users. The default scheduler uses both inherent and customization tools to determine the best fit for a pod.

Advanced pod scheduling

In situations where you might want more control over where new pods are placed, the OpenShift Container Platform advanced scheduling features allow you to configure a pod so that the pod is required or has a preference to run on a particular node or alongside a specific pod by.

- Using pod affinity and anti-affinity rules.
- Controlling pod placement with pod affinity.
- Controlling pod placement with node affinity.
- Placing pods on overcomitted nodes.
- Controlling pod placement with node selectors.
- Controlling pod placement with taints and tolerations.

2.1.1. Scheduler Use Cases

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

2.1.1.1. Infrastructure Topological Levels

Administrators can define multiple topological levels for their infrastructure (nodes) by specifying labels on nodes. For example: **region=r1**, **zone=z1**, **rack=s1**.

These label names have no particular meaning and administrators are free to name their infrastructure levels anything, such as city/building/room. Also, administrators can define any number of levels for their infrastructure topology, with three levels usually being adequate (such as: **regions → zones → racks**). Administrators can specify affinity and anti-affinity rules at each of these levels in any combination.

2.1.1.2. Affinity

Administrators should be able to configure the scheduler to specify affinity at any topological level, or even at multiple levels. Affinity at a particular level indicates that all pods that belong to the same service are scheduled onto nodes that belong to the same level. This handles any latency requirements

of applications by allowing administrators to ensure that peer pods do not end up being too geographically separated. If no node is available within the same affinity group to host the pod, then the pod is not scheduled.

If you need greater control over where the pods are scheduled, see Using Node Affinity and Using Pod Affinity and Anti-affinity.

These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

2.1.1.3. Anti-Affinity

Administrators should be able to configure the scheduler to specify anti-affinity at any topological level, or even at multiple levels. Anti-affinity (or 'spread') at a particular level indicates that all pods that belong to the same service are spread across nodes that belong to that level. This ensures that the application is well spread for high availability purposes. The scheduler tries to balance the service pods across all applicable nodes as evenly as possible.

If you need greater control over where the pods are scheduled, see Using Node Affinity and Using Pod Affinity and Anti-affinity.

These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

2.2. CONFIGURING THE DEFAULT SCHEDULER TO CONTROL POD PLACEMENT

The default OpenShift Container Platform pod scheduler is responsible for determining placement of new pods onto nodes within the cluster. It reads data from the pod and tries to find a node that is a good fit based on configured policies. It is completely independent and exists as a standalone/pluggable solution. It does not modify the pod and just creates a binding for the pod that ties the pod to the particular node.

A selection of predicates and priorities defines the policy for the scheduler. See Modifying scheduler policy for a list of predicates and priorities.

Sample default scheduler object

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: 2019-05-20T15:39:01Z
  generation: 1
  name: cluster
  resourceVersion: "1491"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: 6435dd99-7b15-11e9-bd48-0aec821b8e34
spec:
  policy: 1
    name: scheduler-policy
  defaultNodeSelector: type=user-node,region=east 2
```

1. You can specify the name of a custom scheduler policy file.
2. Optionally, specify a default node selector to restrict pod placement to specific nodes.

2.2.1. Understanding default scheduling

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

Filters the Nodes

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

Prioritize the Filtered List of Nodes

This is achieved by passing each node through a series of priority_ functions that assign it a score between 0 - 10, with 0 indicating a bad fit and 10 indicating a good fit to host the pod. The scheduler configuration can also take in a simple *weight* (positive numeric value) for each priority function. The node score provided by each priority function is multiplied by the weight (default weight for most priorities is 1) and then combined by adding the scores for each node provided by all the priorities. This weight attribute can be used by administrators to give higher importance to some priorities.

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.

2.2.1.1. Understanding Scheduler Policy

The selection of the predicate and priorities defines the policy for the scheduler.

The scheduler configuration file is a JSON file, which must be named **policy.cfg**, that specifies the predicates and priorities the scheduler will consider.

In the absence of the scheduler policy file, the default scheduler behavior is used.

IMPORTANT

The predicates and priorities defined in the scheduler configuration file completely override the default scheduler policy. If any of the default predicates and priorities are required, you must explicitly specify the functions in the policy configuration.

Sample scheduler ConfigMap

```
apiVersion: v1
data:
  policy.cfg: |
    {
        "kind" : "Policy",
        "apiVersion" : "v1",
        "predicates" : [
                {"name" : "MaxGCEPDVolumeCount"},
                {"name" : "GeneralPredicates"},
                {"name" : "MaxAzureDiskVolumeCount"},
                {"name" : "MaxCSIVolumeCountPred"},
                {"name" : "CheckVolumeBinding"},
```

```
            {"name" : "MaxEBSVolumeCount"},
            {"name" : "PodFitsResources"},
            {"name" : "MatchInterPodAffinity"},
            {"name" : "CheckNodeUnschedulable"},
            {"name" : "NoDiskConflict"},
            {"name" : "NoVolumeZoneConflict"},
            {"name" : "MatchNodeSelector"},
            {"name" : "HostName"},
            {"name" : "PodToleratesNodeTaints"}
            ],
      "priorities" : [
            {"name" : "LeastRequestedPriority", "weight" : 1},
            {"name" : "BalancedResourceAllocation", "weight" : 1},
            {"name" : "ServiceSpreadingPriority", "weight" : 1},
            {"name" : "NodePreferAvoidPodsPriority", "weight" : 1},
            {"name" : "NodeAffinityPriority", "weight" : 1},
            {"name" : "TaintTolerationPriority", "weight" : 1},
            {"name" : "ImageLocalityPriority", "weight" : 1},
            {"name" : "SelectorSpreadPriority", "weight" : 1},
            {"name" : "InterPodAffinityPriority", "weight" : 1},
            {"name" : "EqualPriority", "weight" : 1}
            ]
   }
kind: ConfigMap
metadata:
  creationTimestamp: "2019-09-17T08:42:33Z"
  name: scheduler-policy
  namespace: openshift-config
  resourceVersion: "59500"
  selfLink: /api/v1/namespaces/openshift-config/configmaps/scheduler-policy
  uid: 17ee8865-d927-11e9-b213-02d1e1709840`
```

2.2.2. Creating a scheduler policy file

You can control change the default scheduling behavior by creating a JSON file with using the with the desired predicates and priorities. You then generate a ConfigMap from the JSON file and point the **cluster** Scheduler object to use the ConfigMap.

Procedure

To configure the scheduler policy:

1. Create the a JSON file named **policy.cfg** with the desired predicates and priorities.

   Sample scheduler JSON file

```
{
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [ 1
      {"name" : "PodFitsHostPorts"},
      {"name" : "PodFitsResources"},
      {"name" : "NoDiskConflict"},
      {"name" : "NoVolumeZoneConflict"},
      {"name" : "MatchNodeSelector"},
      {"name" : "MaxEBSVolumeCount"},
```

```
        {"name" : "MaxAzureDiskVolumeCount"},
        {"name" : "checkServiceAffinity"},
        {"name" : "PodToleratesNodeNoExecuteTaints"},
        {"name" : "MaxGCEPDVolumeCount"},
        {"name" : "MatchInterPodAffinity"},
        {"name" : "PodToleratesNodeTaints"},
        {"name" : "HostName"}
        ],
"priorities" : [ 2
        {"name" : "LeastRequestedPriority", "weight" : 1},
        {"name" : "BalancedResourceAllocation", "weight" : 1},
        {"name" : "ServiceSpreadingPriority", "weight" : 1},
        {"name" : "EqualPriority", "weight" : 1}
        ]
}
```

1 Add the predicates as needed.

2 Add the priorities as needed.

2. Create a ConfigMap based on the scheduler JSON file:

```
$ oc create configmap -n openshift-config --from-file=policy.cfg <configmap-name> 1
```

1 Enter a name for the ConfigMap.

For example:

```
$ oc create configmap -n openshift-config --from-file=policy.cfg scheduler-policy

configmap/scheduler-policy created
```

```
apiVersion: v1
data:
  policy.cfg: |
    {
        "kind" : "Policy",
        "apiVersion" : "v1",
        "predicates" : [
                {"name" : "MaxGCEPDVolumeCount"},
                {"name" : "GeneralPredicates"},
                {"name" : "MaxAzureDiskVolumeCount"},
                {"name" : "MaxCSIVolumeCountPred"},
                {"name" : "CheckVolumeBinding"},
                {"name" : "MaxEBSVolumeCount"},
                {"name" : "PodFitsResources"},
                {"name" : "MatchInterPodAffinity"},
                {"name" : "CheckNodeUnschedulable"},
                {"name" : "NoDiskConflict"},
                {"name" : "NoVolumeZoneConflict"},
                {"name" : "MatchNodeSelector"},
                {"name" : "HostName"},
                {"name" : "PodToleratesNodeTaints"}
```

```
            ],
        "priorities" : [
            {"name" : "LeastRequestedPriority", "weight" : 1},
            {"name" : "BalancedResourceAllocation", "weight" : 1},
            {"name" : "ServiceSpreadingPriority", "weight" : 1},
            {"name" : "NodePreferAvoidPodsPriority", "weight" : 1},
            {"name" : "NodeAffinityPriority", "weight" : 1},
            {"name" : "TaintTolerationPriority", "weight" : 1},
            {"name" : "ImageLocalityPriority", "weight" : 1},
            {"name" : "SelectorSpreadPriority", "weight" : 1},
            {"name" : "InterPodAffinityPriority", "weight" : 1},
            {"name" : "EqualPriority", "weight" : 1}
            ]
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2019-09-17T08:42:33Z"
  name: scheduler-policy
  namespace: openshift-config
  resourceVersion: "59500"
  selfLink: /api/v1/namespaces/openshift-config/configmaps/scheduler-policy
  uid: 17ee8865-d927-11e9-b213-02d1e1709840`
```

3. Edit the Scheduler Operator Custom Resource to add the ConfigMap:

```
$ oc patch Scheduler cluster --type='merge' -p '{"spec":{"policy":{"name":"<configmap-name>"}}}' --type=merge (1)
```

(1) Specify the name of the ConfigMap.

For example:

```
$ oc patch Scheduler cluster --type='merge' -p '{"spec":{"policy":{"name":"scheduler-policy"}}}' --type=merge
```

After making the change to the Scheduler config resource, wait for the **opensift-kube-apiserver** pods to redeploy. This can take several minutes. Until the pods redeploy, new scheduler does not take effect.

4. Verify the scheduler policy is configured by viewing the log of a scheduler pod in the **openshift-kube-scheduler** namespace. The following command checks for the predoicates and priorites that are being registered by the scheduler:

```
$ oc logs <scheduler-pod> | grep predicates
```

For example:

```
$ oc logs openshift-kube-scheduler-ip-10-0-141-29.ec2.internal | grep predicates

Creating scheduler with fit predicates 'map[MaxGCEPDVolumeCount:{}
MaxAzureDiskVolumeCount:{} CheckNodeUnschedulable:{} NoDiskConflict:{}
NoVolumeZoneConflict:{} MatchNodeSelector:{} GeneralPredicates:{}
MaxCSIVolumeCountPred:{} CheckVolumeBinding:{} MaxEBSVolumeCount:{}
PodFitsResources:{} MatchInterPodAffinity:{} HostName:{} PodToleratesNodeTaints:{}]' and
```

```
priority functions 'map[InterPodAffinityPriority:{} LeastRequestedPriority:{}
ServiceSpreadingPriority:{} ImageLocalityPriority:{} SelectorSpreadPriority:{} EqualPriority:{}
BalancedResourceAllocation:{} NodePreferAvoidPodsPriority:{} NodeAffinityPriority:{}
TaintTolerationPriority:{}]'
```

2.2.3. Modifying scheduler policies

You change scheduling behavior by creating or editing your scheduler policy ConfigMap in the **openshift-config** project. Add and remove predicates and priorities to the ConfigMap to create a *scheduler policy*.

Procedure

To modify the current custom schedluling:

- Edit the scheduler policy ConfigMap:

```
$ oc edit configmap <configmap-name>  -n openshift-config
```

For example:

```
$ oc edit configmap scheduler-policy -n openshift-config

apiVersion: v1
data:
  policy.cfg: |
    {
        "kind" : "Policy",
        "apiVersion" : "v1",
        "predicates" : [ 1
                {"name" : "MaxGCEPDVolumeCount"},
                {"name" : "GeneralPredicates"},
                {"name" : "MaxAzureDiskVolumeCount"},
                {"name" : "MaxCSIVolumeCountPred"},
                {"name" : "CheckVolumeBinding"},
                {"name" : "MaxEBSVolumeCount"},
                {"name" : "PodFitsResources"},
                {"name" : "MatchInterPodAffinity"},
                {"name" : "CheckNodeUnschedulable"},
                {"name" : "NoDiskConflict"},
                {"name" : "NoVolumeZoneConflict"},
                {"name" : "MatchNodeSelector"},
                {"name" : "HostName"},
                {"name" : "PodToleratesNodeTaints"}
                ],
        "priorities" : [ 2
                {"name" : "LeastRequestedPriority", "weight" : 1},
                {"name" : "BalancedResourceAllocation", "weight" : 1},
                {"name" : "ServiceSpreadingPriority", "weight" : 1},
                {"name" : "NodePreferAvoidPodsPriority", "weight" : 1},
                {"name" : "NodeAffinityPriority", "weight" : 1},
                {"name" : "TaintTolerationPriority", "weight" : 1},
                {"name" : "ImageLocalityPriority", "weight" : 1},
                {"name" : "SelectorSpreadPriority", "weight" : 1},
                {"name" : "InterPodAffinityPriority", "weight" : 1},
```

```
            {"name" : "EqualPriority", "weight" : 1}
            ]
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2019-09-17T17:44:19Z"
  name: policy-configmap
  namespace: openshift-kube-scheduler
  resourceVersion: "15370"
  selfLink: /api/v1/namespaces/openshift-kube-scheduler/configmaps/policy-configmap
```

1. Add or remove predicates as needed.
2. Add, remove, or change the weight of predicates as needed.

It can take a few minutes for the scheduler to restart the pods with the updated policy.

- Change the policies and predicates being used:

  1. Remove the scheduler policy CongifMap:

     ```
     $ oc delete configmap -n openshift-config <name>
     ```

     For example:

     ```
     $ oc delete configmap -n openshift-config  scheduler-policy
     ```

  2. Edit the **policy.cfg** file to add and remove policies and predicates as needed.
     For example:

     ```
     $ vi policy.cfg
     ```

     ```
     apiVersion: v1
     data:
       policy.cfg: |
         {
         "kind" : "Policy",
         "apiVersion" : "v1",
         "predicates" : [
                 {"name" : "PodFitsHostPorts"},
                 {"name" : "PodFitsResources"},
                 {"name" : "NoDiskConflict"},
                 {"name" : "NoVolumeZoneConflict"},
                 {"name" : "MatchNodeSelector"},
                 {"name" : "MaxEBSVolumeCount"},
                 {"name" : "MaxAzureDiskVolumeCount"},
                 {"name" : "CheckVolumeBinding"},
                 {"name" : "CheckServiceAffinity"},
                 {"name" : "PodToleratesNodeNoExecuteTaints"},
                 {"name" : "MaxGCEPDVolumeCount"},
                 {"name" : "MatchInterPodAffinity"},
                 {"name" : "PodToleratesNodeTaints"},
                 {"name" : "HostName"}
                 ],
     ```

```
    "priorities" : [
        {"name" : "LeastRequestedPriority", "weight" : 2},
        {"name" : "BalancedResourceAllocation", "weight" : 2},
        {"name" : "ServiceSpreadingPriority", "weight" : 2},
        {"name" : "EqualPriority", "weight" : 2}
        ]
    }
```

3. Re-create the scheduler policy ConfigMap based on the scheduler JSON file:

```
$ oc create configmap -n openshift-config --from-file=policy.cfg <configmap-name> 1
```

1 Enter a name for the ConfigMap.

For example:

```
$ oc create configmap -n openshift-config --from-file=policy.cfg scheduler-policy

configmap/scheduler-policy created
```

2.2.3.1. Understanding the scheduler predicates

Predicates are rules that filter out unqualified nodes.

There are several predicates provided by default in OpenShift Container Platform. Some of these predicates can be customized by providing certain parameters. Multiple predicates can be combined to provide additional filtering of nodes.

2.2.3.1.1. 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.

2.2.3.1.1.1. Default Predicates

The default scheduler policy includes the following predicates:

NoVolumeZoneConflict checks that the volumes a pod requests are available in the zone.

```
{"name" : "NoVolumeZoneConflict"}
```

MaxEBSVolumeCount checks the maximum number of volumes that can be attached to an AWS instance.

```
{"name" : "MaxEBSVolumeCount"}
```

MaxAzureDiskVolumeCount checks the maximum number of Azure Disk Volumes.

```
{"name" : "MaxAzureDiskVolumeCount"}
```

PodToleratesNodeTaints checks if a pod can tolerate the node taints.

```
{"name" : "PodToleratesNodeTaints"}
```

CheckNodeUnschedulable checks if a pod can be scheduled on a node with **Unschedulable** spec.

```
{"name" : "CheckNodeUnschedulable"}
```

CheckVolumeBinding evaluates if a pod can fit based on the volumes, it requests, for both bound and unbound PVCs. * For PVCs that are bound, the predicate checks that the corresponding PV's node affinity is satisfied by the given node. * For PVCs that are unbound, the predicate searched for available PVs that can satisfy the PVC requirements and that the PV node affinity is satisfied by the given node.

The predicate returns true if all bound PVCs have compatible PVs with the node, and if all unbound PVCs can be matched with an available and node-compatible PV.

```
{"name" : "CheckVolumeBinding"}
```

NoDiskConflict checks if the volume requested by a pod is available.

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

MaxGCEPDVolumeCount checks the maximum number of Google Compute Engine (GCE) Persistent Disks (PD).

```
{"name" : "MaxGCEPDVolumeCount"}
```

MaxCSIVolumeCountPred

MatchInterPodAffinity checks if the pod affinity/anti-affinity rules permit the pod.

```
{"name" : "MatchInterPodAffinity"}
```

2.2.3.1.1.2. Other Static Predicates

OpenShift Container Platform also supports the following predicates:

NOTE

The **CheckNode-*** predicates cannot be used if the **Taint Nodes By Condition** feature is enabled. The **Taint Nodes By Condition** feature is enabled by default.

CheckNodeMemoryPressure checks if a pod can be scheduled on a node with a memory pressure condition.

```
{"name" : "CheckNodeMemoryPressure"}
```

CheckNodeDiskPressure checks if a pod can be scheduled on a node with a disk pressure condition.

```
{"name" : "CheckNodeDiskPressure"}
```

CheckNodeCondition checks if a pod can be scheduled on a node reporting **out of disk**, **network unavailable**, or **not ready** conditions.

-

```
{"name" : "CheckNodeCondition"}
```

CheckNodeLabelPresence checks if all of the specified labels exist on a node, regardless of their value.

```
{"name" : "CheckNodeLabelPresence"}
```

checkServiceAffinity checks that ServiceAffinity labels are homogeneous for pods that are scheduled on a node.

```
{"name" : "checkServiceAffinity"}
```

PodToleratesNodeNoExecuteTaints checks if a pod tolerations can tolerate a node **NoExecute** taints.

```
{"name" : "PodToleratesNodeNoExecuteTaints"}
```

2.2.3.1.2. General Predicates

The following general predicates check whether non-critical predicates and essential predicates pass. Non-critical predicates are the predicates that only non-critical pods must pass and essential predicates are the predicates that all pods must pass.

The default scheduler policy includes the general predicates.

Non-critical general predicates
PodFitsResources determines a fit based on resource availability (CPU, memory, GPU, and so forth). The nodes can declare their resource capacities and then pods can specify what resources they require. Fit is based on requested, rather than used resources.

```
{"name" : "PodFitsResources"}
```

Essential general predicates
PodFitsHostPorts determines if a node has free ports for the requested pod ports (absence of port conflicts).

```
{"name" : "PodFitsHostPorts"}
```

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

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

MatchNodeSelector determines fit based on node selector (nodeSelector) queries defined in the pod.

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

2.2.3.2. Understanding the scheduler priorities

Priorities are rules that rank nodes according to preferences.

A custom set of priorities can be specified to configure the scheduler. There are several priorities provided by default in OpenShift Container Platform. Other priorities can be customized by providing certain parameters. Multiple priorities can be combined and different weights can be given to each in

order to impact the prioritization.

2.2.3.2.1. Static Priorities

Static priorities do not take any configuration parameters from the user, except weight. A weight is required to be specified and cannot be 0 or negative.

These are specified in the scheduler policy Configmap, **policy-configmap** in the **openshift-config** project.

2.2.3.2.1.1. Default Priorities

The default scheduler policy includes the following priorities. Each of the priority function has a weight of **1** except **NodePreferAvoidPodsPriority**, which has a weight of **10000**.

NodeAffinityPriority prioritizes nodes according to node affinity scheduling preferences

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

TaintTolerationPriority prioritizes nodes that have a fewer number of *intolerable* taints on them for a pod. An intolerable taint is one which has key **PreferNoSchedule**.

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

ImageLocalityPriority prioritizes nodes that already have requested pod container's images.

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

SelectorSpreadPriority looks for services, replication controllers (RC), replication sets (RS), and stateful sets that match the pod, then finds existing pods that match those selectors. The scheduler favors nodes that have fewer existing matching pods. Then, it schedules the pod on a node with the smallest number of pods that match those selectors as the pod being scheduled.

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

InterPodAffinityPriority computes a sum by iterating through the elements of **weightedPodAffinityTerm** and adding *weight* to the sum if the corresponding PodAffinityTerm is satisfied for that node. The node(s) with the highest sum are the most preferred.

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

LeastRequestedPriority favors nodes with fewer requested resources. It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes nodes that have the highest available/remaining capacity.

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

BalancedResourceAllocation favors nodes with balanced resource usage rate. It calculates the difference between the consumed CPU and memory as a fraction of capacity, and prioritizes the nodes based on how close the two metrics are to each other. This should always be used together with **LeastRequestedPriority**.

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

NodePreferAvoidPodsPriority ignores pods that are owned by a controller other than a replication controller.

```
{"name" : "NodePreferAvoidPodsPriority", "weight" : 10000}
```

2.2.3.2.1.2. Other Static Priorities

OpenShift Container Platform also supports the following priorities:

EqualPriority gives an equal weight of **1** to all nodes, if no priority configurations are provided. We recommend using this priority only for testing environments.

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

MostRequestedPriority prioritizes nodes with most requested resources. It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes based on the maximum of the average of the fraction of requested to capacity.

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

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

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

2.2.3.2.2. Configurable Priorities

You can configure these priorities in the scheduler policy Configmap, **policy-configmap** in the **openshift-config** project, to add labels to affect how the priorities.

The type of the priority function is identified by the argument that they take. Since these are configurable, multiple priorities of the same type (but different configuration parameters) can be combined as long as their user-defined names are different.

For information on using these priorities, see Modifying Scheduler Policy.

ServiceAntiAffinity takes a label and ensures a good spread of the pods belonging to the same service across the group of nodes based on the label values. It gives the same score to all nodes that have the same value for the specified label. It gives a higher score to nodes within a group with the least concentration of pods.

```
{
"kind": "Policy",
"apiVersion": "v1",

"priorities":[
   {
      "name":"<name>", 1
      "weight" : 1 2
      "argument":{
         "serviceAntiAffinity":{
            "label": "<label>" 3
            }
```

```
        }
      }
  ]
}
```

1. Specify a name for the priority.
2. Specify a weight. Enter a non-zero positive value.
3. Specify a label to match.

For example:

```
{
"kind": "Policy",
"apiVersion": "v1",
"priorities": [
   {
      "name":"RackSpread",
      "weight" : 1,
      "argument": {
         "serviceAntiAffinity": {
            "label": "rack"
            }
      }
   }
  ]
}
```

NOTE

In some situations using **ServiceAntiAffinity** based on custom labels does not spread pod as expected. See this Red Hat Solution .

*The **labelPreference** parameter gives priority based on the specified label. If the label is present on a node, that node is given priority. If no label is specified, priority is given to nodes that do not have a label.

```
{
"kind": "Policy",
"apiVersion": "v1",
"priorities":[
   {
      "name":"<name>", 1
      "weight" : 1 2
      "argument":{
         "labelPreference":{
            "label": "<label>", 3
            "presence": true 4
            }
         }
      }
   ]
}
```

1. Specify a name for the priority.
2. Specify a weight. Enter a non-zero positive value.
3. Specify a label to match.
4. Specify whether the label is required, either **true** or **false**.

2.2.4. Sample Policy Configurations

The configuration below specifies the default scheduler configuration, if it were to be specified using the scheduler policy file.

```
{
"kind": "Policy",
"apiVersion": "v1",
"predicates": [
    {
        "name": "RegionZoneAffinity", 1
        "argument": {
            "serviceAffinity": { 2
              "labels": "region, zone" 3
           }
        }
    }
  ],
"priorities": [
    {
        "name":"RackSpread", 4
        "weight" : 1,
        "argument": {
            "serviceAntiAffinity": { 5
                "label": "rack" 6
                }
           }
       }
    ]
}
```

1. The name for the predicate.
2. The type of predicate.
3. The labels for the predicate.
4. The name for the priority.
5. The type of priority.
6. The labels for the priority.

In all of the sample configurations below, the list of predicates and priority functions is truncated to include only the ones that pertain to the use case specified. In practice, a complete/meaningful scheduler policy should include most, if not all, of the default predicates and priorities listed above.

The following example defines three topological levels, region (affinity) → zone (affinity) → rack (anti-affinity):

```
{
"kind": "Policy",
"apiVersion": "v1",
"predicates": [
    {
        "name": "RegionZoneAffinity",
        "argument": {
            "serviceAffinity": {
              "label": "region, zone"
           }
        }
    }
  ],
"priorities": [
    {
        "name":"RackSpread",
        "weight" : 1,
        "argument": {
            "serviceAntiAffinity": {
                "label": "rack"
                }
           }
      }
   ]
}
```

The following example defines three topological levels, city (affinity) → building (anti-affinity) → room (anti-affinity):

```
{
"kind": "Policy",
"apiVersion": "v1",
"predicates": [
    {
        "name": "CityAffinity",
        "argument": {
            "serviceAffinity": {
              "label": "city"
           }
        }
    }
  ],
"priorities": [
    {
        "name":"BuildingSpread",
        "weight" : 1,
        "argument": {
            "serviceAntiAffinity": {
                "label": "building"
```

```
            }
        }
      },
    {
        "name":"RoomSpread",
        "weight" : 1,
        "argument": {
            "serviceAntiAffinity": {
                "label": "room"
                }
        }
      }
  ]
}
```

The following example defines a policy to only use nodes with the 'region' label defined and prefer nodes with the 'zone' label defined:

```
{
"kind": "Policy",
"apiVersion": "v1",
"predicates": [
    {
        "name": "RequireRegion",
        "argument": {
            "labelPreference": {
                "label": "region",
                "presence": true
          }
        }
     }
  ],
"priorities": [
    {
        "name":"ZonePreferred",
        "weight" : 1,
        "argument": {
            "labelPreference": {
                "label": "zone",
                "presence": true
                }
        }
      }
  ]
}
```

The following example combines both static and configurable predicates and also priorities:

```
{
"kind": "Policy",
"apiVersion": "v1",
"predicates": [
    {
        "name": "RegionAffinity",
        "argument": {
```

```
            "serviceAffinity": {
                "label": "region"
            }
          }
      },
      {
          "name": "RequireRegion",
          "argument": {
              "labelsPresence": {
                  "label": "region",
                  "presence": true
              }
          }
      },
      {
          "name": "BuildingNodesAvoid",
          "argument": {
              "labelsPresence": {
                  "label": "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}
      ]
  }
```

2.3. PLACING PODS RELATIVE TO OTHER PODS USING AFFINITY AND ANTI-AFFINITY RULES

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled. Anti-affinity is a property of pods that prevents a pod from being scheduled on a node.

In OpenShift Container Platform *pod affinity* and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key/value labels on other pods.

2.3.1. Understanding pod affinity

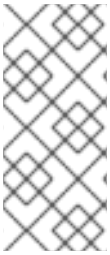
Pod affinity and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key/value labels on other pods.

- Pod affinity can tell the scheduler to locate a new pod on the same node as other pods if the label selector on the new pod matches the label on the current pod.
- Pod anti-affinity can prevent the scheduler from locating a new pod on the same node as pods with the same labels if the label selector on the new pod matches the label on the current pod.

For example, using affinity rules, you could spread or pack pods within a service or relative to pods in other services. Anti-affinity rules allow you to prevent pods of a particular service from scheduling on the same nodes as pods of another service that are known to interfere with the performance of the pods of the first service. Or, you could spread the pods of a service across nodes or availability zones to reduce correlated failures.

There are two types of pod affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.



NOTE

Depending on your pod priority and preemption settings, the scheduler might not be able to find an appropriate node for a pod without violating affinity requirements. If so, a pod might not be scheduled.

To prevent this situation, carefully configure pod affinity with equal-priority pods.

You configure pod affinity/anti-affinity through the pod specification files. You can specify a required rule, a preferred rule, or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example shows a pod specification configured for pod affinity and anti-affinity.

In this example, the pod affinity rule indicates that the pod can schedule onto a node only if that node has at least one already-running pod with a label that has the key **security** and value **S1**. The pod anti-affinity rule says that the pod prefers to not schedule onto a node if that node is already running a pod with label having key **security** and value **S2**.

Sample pod config file with pod affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution: 2
      - labelSelector:
```

```
        matchExpressions:
        - key: security 3
          operator: In 4
          values:
          - S1 5
      topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
```

1. Stanza to configure pod affinity.
2. Defines a required rule.
3. 5 The key and value (label) that must be matched to apply the rule.
4. The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.

Sample pod config file with pod anti-affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security 4
              operator: In 5
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
```

1. Stanza to configure pod anti-affinity.
2. Defines a preferred rule.
3. Specifies a weight for a preferred rule. The node with the highest weight is preferred.
4. Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.
5. The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**,

Exists, or **DoesNotExist**.

NOTE

If labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod continues to run on the node.

2.3.2. Configuring a pod affinity rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses affinity to allow scheduling with that pod.

Procedure

1. Create a pod with a specific label in the pod specification:

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

2. When creating other pods, edit the pod specification as follows:

   a. Use the **podAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter or **preferredDuringSchedulingIgnoredDuringExecution** parameter:

   b. Specify the key and value that must be met. If you want the new pod to be scheduled with the other pod, use the same **key** and **value** parameters as the label on the first pod.

```
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
      - key: security
        operator: In
        values:
        - S1
    topologyKey: failure-domain.beta.kubernetes.io/zone
```

   c. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.

   d. Specify a **topologyKey**, which is a prepopulated Kubernetes label that the system uses to denote such a topology domain.

3. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

2.3.3. Configuring a pod anti-affinity rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses an anti-affinity preferred rule to attempt to prevent scheduling with that pod.

Procedure

1. Create a pod with a specific label in the pod specification:

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s2
  labels:
    security: S2
spec:
  containers:
  - name: security-s2
    image: docker.io/ocpqe/hello-pod
```

2. When creating other pods, edit the pod specification to set the following parameters:

3. Use the **podAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter or **preferredDuringSchedulingIgnoredDuringExecution** parameter:

   a. Specify a weight for the node, 1-100. The node that with highest weight is preferred.

   b. Specify the key and values that must be met. If you want the new pod to not be scheduled with the other pod, use the same **key** and **value** parameters as the label on the first pod.

```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - S2
      topologyKey: kubernetes.io/hostname
```

   c. For a preferred rule, specify a weight, 1-100.

   d. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.

4. Specify a **topologyKey**, which is a prepopulated Kubernetes label that the system uses to denote such a topology domain.

5. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

2.3.4. Sample pod affinity and anti-affinity rules

The following examples demonstrate pod affinity and pod anti-affinity.

2.3.4.1. Pod Affinity

The following example demonstrates pod affinity for pods with matching labels and label selectors.

- The pod **team4** has the label **team:4**.

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- The pod **team4a** has the label selector **team:4** under **podAffinity**.

```
$ cat pod-team4a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4a
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: team
            operator: In
            values:
            - "4"
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- The **team4a** pod is scheduled on the same node as the **team4** pod.

2.3.4.2. Pod Anti-affinity

The following example demonstrates pod anti-affinity for pods with matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```
cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** has the label selector **security:s1** under **podAntiAffinity**.

```
cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s1
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-antiaffinity
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** cannot be scheduled on the same node as **pod-s1**.

2.3.4.3. Pod Affinity with no Matching Labels

The following example demonstrates pod affinity for pods without matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** has the label selector **security:s2**.

```
$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- The pod **pod-s2** is not scheduled unless there is a node with a pod that has the **security:s2** label. If there is no other pod with that label, the new pod remains in a pending state:

```
NAME     READY     STATUS    RESTARTS   AGE       IP        NODE
pod-s2   0/1       Pending   0          32s       <none>
```

2.4. CONTROLLING POD PLACEMENT ON NODES USING NODE AFFINITY RULES

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled.

In OpenShift Container Platformnode affinity is a set of rules used by the scheduler to determine where a pod can be placed. The rules are defined using custom labels on the nodes and label selectors specified in pods.

2.4.1. Understanding node affinity

Node affinity allows a pod to specify an affinity towards a group of nodes it can be placed on. The node does not have control over the placement.

For example, you could configure a pod to only run on a node with a specific CPU or in a specific availability zone.

There are two types of node affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

NOTE

If labels on a node change at runtime that results in an node affinity rule on a pod no longer being met, the pod continues to run on the node.

You configure node affinity through the pod specification file. You can specify a required rule, a preferred rule, or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example is a pod specification with a rule that requires the pod be placed on a node with a label whose key is **e2e-az-NorthSouth** and whose value is either **e2e-az-North** or **e2e-az-South**:

Sample pod configuration file with a node affinity required rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution: 2
        nodeSelectorTerms:
        - matchExpressions:
          - key: e2e-az-NorthSouth 3
            operator: In 4
            values:
            - e2e-az-North 5
            - e2e-az-South 6
  containers:
  - name: with-node-affinity
    image: docker.io/ocpqe/hello-pod
```

1 The stanza to configure node affinity.

2 Defines a required rule.

3 5 6 The key/value pair (label) that must be matched to apply the rule.

4 The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the pod specification. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

The following example is a node specification with a preferred rule that a node with a label whose key is **e2e-az-EastWest** and whose value is either **e2e-az-East** or **e2e-az-West** is preferred for the pod:

Sample pod configuration file with a node affinity preferred rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 1 3
        preference:
          matchExpressions:
```

```
        - key: e2e-az-EastWest 4
          operator: In 5
          values:
          - e2e-az-East 6
          - e2e-az-West 7
  containers:
  - name: with-node-affinity
    image: docker.io/ocpqe/hello-pod
```

1. The stanza to configure node affinity.
2. Defines a preferred rule.
3. Specifies a weight for a preferred rule. The node with highest weight is preferred.
4. 6 7 The key/value pair (label) that must be matched to apply the rule.
5. The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the pod specification. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

There is no explicit *node anti-affinity* concept, but using the **NotIn** or **DoesNotExist** operator replicates that behavior.

NOTE

If you are using node affinity and node selectors in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

2.4.2. Configuring a required node affinity rule

Required rules **must** be met before a pod can be scheduled on a node.

Procedure

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler is required to place on the node.

1. Add a label to a node using the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az1
```

2. In the pod specification, use the **nodeAffinity** stanza to configure the **requiredDuringSchedulingIgnoredDuringExecution** parameter:

   a. Specify the key and values that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **value** parameters as the label in the node.

   b. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt**, or **Gt**. For example, use the operator **In** to require the label to be in the node:

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
```

3. Create the pod:

```
$ oc create -f e2e-az2.yaml
```

2.4.3. Configuring a Preferred Node Affinity Rule

Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

Procedure

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler tries to place on the node.

1. Add a label to a node using the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2. In the pod specification, use the **nodeAffinity** stanza to configure the **preferredDuringSchedulingIgnoredDuringExecution** parameter:

   a. Specify a weight for the node, as a number 1-100. The node with highest weight is preferred.

   b. Specify the key and values that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **value** parameters as the label in the node:

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: e2e-az-name
```

```
      operator: In
      values:
      - e2e-az3
```

3. Specify an **operator**. The operator can be **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt**, or **Gt**. For example, use the operator **In** to require the label to be in the node.

4. Create the pod.

```
$ oc create -f e2e-az3.yaml
```

2.4.4. Sample node affinity rules

The following examples demonstrate node affinity.

2.4.4.1. Node Affinity with Matching Labels

The following example demonstrates node affinity for a node and pod with matching labels:

- The **Node1** node has the label **zone:us**:

```
$ oc label node node1 zone=us
```

- The pod **pod-s1** has the **zone** and **us** key/value pair under a required node affinity rule:

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- The pod **pod-s1** can be scheduled on **Node1**:

```
$ oc get pod -o wide
NAME     READY     STATUS      RESTARTS   AGE      IP      NODE
pod-s1   1/1       Running     0          4m       IP1     node1
```

2.4.4.2. Node Affinity with No Matching Labels

The following example demonstrates node affinity for a node and pod without matching labels:

- The **Node1** node has the label **zone:emea**:

```
$ oc label node node1 zone=emea
```

- The pod **pod-s1** has the **zone** and **us** key/value pair under a required node affinity rule:

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- The pod **pod-s1** cannot be scheduled on **Node1**:

```
$ oc describe pod pod-s1
<---snip--->
Events:
 FirstSeen LastSeen Count From              SubObjectPath  Type              Reason
 --------- -------- ----- ----              -------------  --------          ------
 1m        33s      8     default-scheduler Warning        FailedScheduling  No nodes are
available that match all of the following predicates:: MatchNodeSelector (1).
```

2.4.5. Additional resources

For information about changing node labels, see Understanding how to update labels on nodes .

2.5. PLACING PODS ONTO OVERCOMMITED NODES

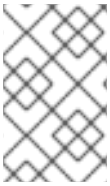
In an *overcommited* state, the sum of the container compute resource requests and limits exceeds the resources available on the system. Overcommitment might be desirable in development environments where a trade-off of guaranteed performance for capacity is acceptable.

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node. The scheduler uses requests for scheduling your container and providing a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

2.5.1. Understanding overcommitment

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node. The scheduler uses requests for scheduling your container and providing a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

OpenShift Container Platform administrators can control the level of overcommit and manage container density on nodes by configuring masters 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.



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.

After these 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.

2.5.2. Understanding nodes overcommitment

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

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.

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

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, OpenShift Container Platform configures the kernel to always overcommit memory by setting the **vm.overcommit_memory** parameter to **1**, overriding the default operating system setting.

OpenShift Container Platform also configures the kernel not to panic when it runs out of memory by setting the **vm.panic_on_oom** parameter to **0**. A setting of 0 instructs the kernel to call oom_killer in an Out of Memory (OOM) condition, which kills processes based on priority

You can view the current setting by running the following commands on your nodes:

```
$ sysctl -a |grep commit

vm.overcommit_memory = 1
```

```
$ sysctl -a |grep panic
vm.panic_on_oom = 0
```

NOTE

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

You can also perform the following configurations for each node:

- Disable or enforce CPU limits using CPU CFS quotas
- Reserve resources for system processes
- Reserve memory across quality of service tiers

2.6. CONTROLLING POD PLACEMENT USING NODE TAINTS

Taints and tolerations allow the Node to control which Pods should (or should not) be scheduled on them.

2.6.1. Understanding taints and tolerations

A *taint* allows a node to refuse pod to be scheduled unless that pod has a matching *toleration*.

You apply taints to a node through the node specification (**NodeSpec**) and apply tolerations to a pod through the pod specification (**PodSpec**). A taint on a node instructs the node to repel all pods that do not tolerate the taint.

Taints and tolerations consist of a key, value, and effect. An operator allows you to leave one of these parameters empty.

Table 2.1. Taint and toleration components

Parameter	Description
key	The **key** is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.
value	The **value** is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.

Parameter	Description	
effect	The effect is one of the following:	
	NoSchedule	• New pods that do not match the taint are not scheduled onto that node. • Existing pods on the node remain.
	PreferNoSchedule	• New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. • Existing pods on the node remain.
	NoExecute	• New pods that do not match the taint cannot be scheduled onto that node. • Existing pods on the node that do not have a matching toleration are removed.
operator		
	Equal	The **key**/**value**/**effect** parameters must match. This is the default.
	Exists	The **key**/**effect** parameters must match. You must leave a blank **value** parameter, which matches any.

A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:
  - the **key** parameters are the same;
  - the **value** parameters are the same;
  - the **effect** parameters are the same.
- If the **operator** parameter is set to **Exists**:
  - the **key** parameters are the same;
  - the **effect** parameters are the same.

The following taints are built into kubernetes:

- **node.kubernetes.io/not-ready**: The node is not ready. This corresponds to the node condition **Ready=False**.

- **node.kubernetes.io/unreachable**: The node is unreachable from the node controller. This corresponds to the node condition **Ready=Unknown**.
- **node.kubernetes.io/out-of-disk**: The node has insufficient free space on the node for adding new pods. This corresponds to the node condition **OutOfDisk=True**.
- **node.kubernetes.io/memory-pressure**: The node has memory pressure issues. This corresponds to the node condition **MemoryPressure=True**.
- **node.kubernetes.io/disk-pressure**: The node has disk pressure issues. This corresponds to the node condition **DiskPressure=True**.
- **node.kubernetes.io/network-unavailable**: The node network is unavailable.
- **node.kubernetes.io/unschedulable**: The node is unschedulable.
- **node.cloudprovider.kubernetes.io/uninitialized**: When the node controller is started with an external cloud provider, this taint is set on a node to mark it as unusable. After a controller from the cloud-controller-manager initializes this node, the kubelet removes this taint.

2.6.1.1. Understanding how to use toleration seconds to delay pod evictions

You can specify how long a pod can remain bound to a node before being evicted by specifying the **tolerationSeconds** parameter in the pod specification. If a taint with the **NoExecute** effect is added to a node, any pods that do not tolerate the taint are evicted immediately (pods that do tolerate the taint are not evicted). However, if a pod that to be evicted has the **tolerationSeconds** parameter, the pod is not evicted until that time period expires.

For example:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

Here, if this pod is running but does not have a matching taint, the pod stays bound to the node for 3,600 seconds and then be evicted. If the taint is removed before that time, the pod is not evicted.

2.6.1.2. Understanding how to use multiple taints

You can put multiple taints on the same node and multiple tolerations on the same pod. OpenShift Container Platform processes multiple taints and tolerations as follows:

1. Process the taints for which the pod has a matching toleration.
2. The remaining unmatched taints have the indicated effects on the pod:
   - If there is at least one unmatched taint with effect **NoSchedule**, OpenShift Container Platform cannot schedule a pod onto that node.
   - If there is no unmatched taint with effect **NoSchedule** but there is at least one unmatched taint with effect **PreferNoSchedule**, OpenShift Container Platform tries to not schedule the pod onto the node.

- If there is at least one unmatched taint with effect **NoExecute**, OpenShift Container Platform evicts the pod from the node (if it is already running on the node), or the pod is not scheduled onto the node (if it is not yet running on the node).
  - Pods that do not tolerate the taint are evicted immediately.
  - Pods that tolerate the taint without specifying **tolerationSeconds** in their toleration specification remain bound forever.
  - Pods that tolerate the taint with a specified **tolerationSeconds** remain bound for the specified amount of time.

For example:

- The node has the following taints:

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
$ oc adm taint nodes node1 key1=value1:NoExecute
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- The pod has the following tolerations:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

In this case, the pod cannot be scheduled onto the node, because there is no toleration matching the third taint. The pod continues running if it is already running on the node when the taint is added, because the third taint is the only one of the three that is not tolerated by the pod.

2.6.1.3. Preventing pod eviction for node problems

OpenShift Container Platform can be configured to represent **node unreachable** and **node not ready** conditions as taints. This allows per-pod specification of how long to remain bound to a node that becomes unreachable or not ready, rather than using the default of five minutes.

The Taint-Based Evictions feature is enabled by default. The taints are automatically added by the node controller and the normal logic for evicting pods from **Ready** nodes is disabled.

- If a node enters a not ready state, the **node.kubernetes.io/not-ready:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.
- If a node enters a not reachable state, the **node.kubernetes.io/unreachable:NoExecute** taint is added and pods cannot be scheduled on the node. Existing pods remain for the toleration seconds period.

This feature, in combination with **tolerationSeconds**, allows a pod to specify how long it should stay bound to a node that has one or both of these problems.

2.6.1.4. Understanding pod scheduling and node conditions (Taint Node by Condition)

OpenShift Container Platform automatically taints nodes that report conditions such as memory pressure and disk pressure. If a node reports a condition, a taint is added until the condition clears. The taints have the **NoSchedule** effect, which means no pod can be scheduled on the node, unless the pod has a matching toleration. This feature, **Taint Nodes By Condition**, is enabled by default.

The scheduler checks for these taints on nodes before scheduling pods. If the taint is present, the pod is scheduled on a different node. Because the scheduler checks for taints and not the actual Node conditions, you configure the scheduler to ignore some of these node conditions by adding appropriate Pod tolerations.

The DaemonSet controller automatically adds the following tolerations to all daemons, to ensure backward compatibility:

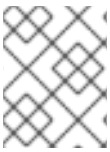
- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/out-of-disk (only for critical pods)
- node.kubernetes.io/unschedulable (1.10 or later)
- node.kubernetes.io/network-unavailable (host network only)

You can also add arbitrary tolerations to DaemonSets.

2.6.1.5. Understanding evicting pods by condition (Taint-Based Evictions)

The Taint-Based Evictions feature, enabled by default, evicts pods from a node that experiences specific conditions, such as **not-ready** and **unreachable**. When a node experiences one of these conditions, OpenShift Container Platform automatically adds taints to the node, and starts evicting and rescheduling the pods on different nodes.

Taint Based Evictions has a **NoExecute** effect, where any pod that does not tolerate the taint will be evicted immediately and any pod that does tolerate the taint will never be evicted.



NOTE

OpenShift Container Platform evicts pods in a rate-limited way to prevent massive pod evictions in scenarios such as the master becoming partitioned from the nodes.

This feature, in combination with **tolerationSeconds**, allows you to specify how long a pod should stay bound to a node that has a node condition. If the condition still exists after the **tolerationSections** period, the taint remains on the node and the pods are evicted in a rate-limited manner. If the condition clears before the **tolerationSeconds** period, pods are not removed.

OpenShift Container Platform automatically adds a toleration for **node.kubernetes.io/not-ready** and **node.kubernetes.io/unreachable** with **tolerationSeconds=300**, unless the pod configuration specifies either toleration.

```
spec
  tolerations:
    - key: node.kubernetes.io/not-ready
      operator: Exists
      effect: NoExecute
```

```
    tolerationSeconds: 300
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```

These tolerations ensure that the default pod behavior is to remain bound for 5 minutes after one of these node conditions problems is detected.

You can configure these tolerations as needed. For example, if you have an application with a lot of local state you might want to keep the pods bound to node for a longer time in the event of network partition, allowing for the partition to recover and avoiding pod eviction.

DaemonSet pods are created with NoExecute tolerations for the following taints with no tolerationSeconds:

- **node.kubernetes.io/unreachable**
- **node.kubernetes.io/not-ready**

This ensures that DaemonSet pods are never evicted due to these node conditions, even if the **DefaultTolerationSeconds** admission controller is disabled.

2.6.2. Adding taints and tolerations

You add taints to nodes and tolerations to pods allow the node to control which pods should (or should not) be scheduled on them.

Procedure

1. Use the following command using the parameters described in the taint and toleration components table:

   ```
   $ oc adm taint nodes <node-name> <key>=<value>:<effect>
   ```

   For example:

   ```
   $ oc adm taint nodes node1 key1=value1:NoExecute
   ```

   This example places a taint on **node1** that has key **key1**, value **value1**, and taint effect **NoExecute**.

2. Add a toleration to a pod by editing the pod specification to include a **tolerations** section:

   Sample pod configuration file with Equal operator

   ```
   tolerations:
   - key: "key1" 1
     operator: "Equal" 2
     value: "value1" 3
     effect: "NoExecute" 4
     tolerationSeconds: 3600 5
   ```

1 2 3 4 The toleration parameters, as described in the taint and toleration components table.

5 The **tolerationSeconds** parameter specifies how long a pod can remain bound to a node before being evicted.

For example:

Sample pod configuration file with Exists operator

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

Both of these tolerations match the taint created by the **oc adm taint** command above. A pod with either toleration would be able to schedule onto **node1**.

2.6.2.1. Dedicating a Node for a User using taints and tolerations

You can specify a set of nodes for exclusive use by a particular set of users.

Procedure

To specify dedicated nodes:

1. Add a taint to those nodes:
   For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom admission controller.
   Only the pods with the tolerations are allowed to use the dedicated nodes.

2.6.2.2. Binding a user to a Node using taints and tolerations

You can configure a node so that particular users can use only the dedicated nodes.

Procedure

To configure a node so that users can use only that node:

1. Add a taint to those nodes:
   For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. Add a corresponding toleration to the pods by writing a custom admission controller.
   The admission controller should add a node affinity to require that the pods can only schedule onto nodes labeled with the **key:value** label (**dedicated=groupName**).

3. Add a label similar to the taint (such as the **key:value** label) to the dedicated nodes.

2.6.2.3. Controlling Nodes with special hardware using taints and tolerations

In a cluster where a small subset of nodes have specialized hardware (for example GPUs), you can use taints and tolerations to keep pods that do not need the specialized hardware off of those nodes, leaving the nodes for pods that do need the specialized hardware. You can also require pods that need specialized hardware to use specific nodes.

Procedure

To ensure pods are blocked from the specialized hardware:

1. Taint the nodes that have the specialized hardware using one of the following commands:

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

2. Adding a corresponding toleration to pods that use the special hardware using an admission controller.

For example, the admission controller could use some characteristic(s) of the pod to determine that the pod should be allowed to use the special nodes by adding a toleration.

To ensure pods can only use the specialized hardware, you need some additional mechanism. For example, you could label the nodes that have the special hardware and use node affinity on the pods that need the hardware.

2.6.3. Removing taints and tolerations

You can remove taints from nodes and tolerations from pods as needed.

Procedure

To remove taints and tolerations:

1. To remove a taint from a node:

```
$ oc adm taint nodes <node-name> <key>-
```

For example:

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-

node/ip-10-0-132-248.ec2.internal untainted
```

2. To remove a toleration from a pod, edit the pod specification to remove the toleration:

```
tolerations:
- key: "key2"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

2.7. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS

A *node selector* specifies a map of key-value pairs. The rules are defined using custom labels on nodes

and selectors specified in pods. You can use node selectors to place specific pods on specific nodes, all pods in a project on specific nodes, or create a default node selector to schedule pods that do not have a defined node selector or project selector.

For the pod to be eligible to run on a node, the pod must have the indicated key-value pairs as the label on the node.

If you are using node affinity and node selectors in the same pod configuration, see the important considerations below.

2.7.1. Using node selectors to control pod placement

You can use node selector labels on pods to control where the pod is scheduled.

With node selectors, OpenShift Container Platform schedules the pods on nodes that contain matching labels.

You can add labels to a node or MachineConfig, but the labels will not persist if the node or machine goes down. Adding the label to the MachineSet ensures that new nodes or machines will have the label.

To add node selectors to an existing pod, add a node selector to the controlling object for that node, such as a ReplicaSet, Daemonset, or StatefulSet. Any existing pods under that controlling object are recreated on a node with a matching label. If you are creating a new pod, you can add the node selector directly to the pod spec.

NOTE

You cannot add a node selector to an existing scheduled pod.

Prerequisites

If you want to add a node selector to existing pods, determine the controlling object for that pod. For exeample, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** ReplicaSet:

```
$ oc describe pod router-default-66d5cf9464-7pwkc

Name:             router-default-66d5cf9464-7pwkc
Namespace:        openshift-ingress

....

Controlled By:    ReplicaSet/router-default-66d5cf9464
```

The web console lists the controlling object under **ownerReferences** in the pod YAML:

```
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: router-default-66d5cf9464
      uid: d81dd094-da26-11e9-a48a-128e7edf0312
      controller: true
      blockOwnerDeletion: true
```

Procedure

1. Add the desired label to your nodes:

```
$ oc label <resource> <name> <key>=<value>
```

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

The label is applied to the node:

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    beta.kubernetes.io/os: linux
    failure-domain.beta.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.2'
    node-role.kubernetes.io/worker: ''
    failure-domain.beta.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    beta.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    region: east (1)
    beta.kubernetes.io/arch: amd64
    type: user-node (2)
....
```

(1) (2) Specify the label(s) you will add to the node.

Alternatively, you can add the label to a MachineSet:

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c
```

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet

....

spec:
  replicas: 2
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: ci-ln-89dz2y2-d5d6b-4995x
      machine.openshift.io/cluster-api-machine-role: worker
      machine.openshift.io/cluster-api-machine-type: worker
      machine.openshift.io/cluster-api-machineset: ci-ln-89dz2y2-d5d6b-4995x-worker-us-east-
```

```
1a
 template:
   metadata:
     creationTimestamp: null
     labels:
       machine.openshift.io/cluster-api-cluster: ci-ln-89dz2y2-d5d6b-4995x
       machine.openshift.io/cluster-api-machine-role: worker
       machine.openshift.io/cluster-api-machine-type: worker
       machine.openshift.io/cluster-api-machineset: ci-ln-89dz2y2-d5d6b-4995x-worker-us-
east-1a
    spec:
      metadata:
        creationTimestamp: null
        labels:
          region: east (1)
          type: user-node (2)
....
```

(1) (2) Specify the label(s) you will add to the node.

2. Add the desired node selector a pod:

   - To add a node selector to existing and furture pods, add a node selector to the controlling object for the pods:
     For example:

     ```
     kind: ReplicaSet

     ....

     spec:

     ....

       template:
         metadata:
           creationTimestamp: null
           labels:
             ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
             pod-template-hash: 66d5cf9464
         spec:
           nodeSelector:
             beta.kubernetes.io/os: linux
             node-role.kubernetes.io/worker: ''
             type: user-node (1)
     ```

     (1) Add the desired node selector.

   - For a new pod, you can add the selector to the pod specification directly:

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

```
...

spec:
  nodeSelector:
    <key>: <value>

...
```

For example:

```
apiVersion: v1
kind: Pod

....

spec:
  nodeSelector:
    region: east
    type: user-node
```

NOTE

If you are using node selectors and node affinity in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

2.7.2. Creating default cluster-wide node selectors

You can use default node selectors on pods together with labels on nodes to constrain all pods created in a cluster to specific nodes.

With cluster node selectors, when you create a pod in that cluster, OpenShift Container Platform adds the appropriate **<key>:<value>** and schedules the pod on nodes with matching labels.

NOTE

You can add additional **<key>:<value>** pairs for the pod. But you cannot add a different **<value>** for a default **<key>**.

For example, if the cluster node selector is **region: east** the following pod spec adds a new pair and is allowed:

```
spec:
  nodeSelector:
```

```
    region: east
    type: user-node
```

The following pod spec uses a different value for **region** and is not allowed:

```
spec:
  nodeSelector:
    region: west
```

If the project where you are creating the pod has a project node selector, that selector takes preference over a cluster node selector.

Procedure

To add a default cluster node selector:

1. Edit the Scheduler Operator Custom Resource to add the cluster node selectors:

```
$ oc edit scheduler cluster
```

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec: {}
  policy:
spec:
  defaultNodeSelector: type=user-node,region=east (1)
```

(1) Add a node selector with the appropriate **<key>:<value>** pairs.

After making this change, wait for the pods in the **openshift-kube-apiserver** project to redeploy. This can take several minutes. The default cluster node selector does not take effect until the pods redeploy.

2. Edit a node or MachineSet to add labels:

```
$ oc label <resource> <name> <key>=<value>
```

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

To label a MachineSet:

```
$ oc label MachineSet abc612-msrtw-worker-us-east-1c type=user-node region=east
```

When you create a pod, OpenShift Container Platform adds the appropriate **<key>:<value>** and schedules the pod on the labeled node.

For example:

```
spec:
```

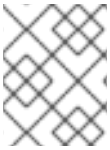
```
  nodeSelector:
    region: east
    type: user-node
```

2.7.3. Creating project-wide node selectors

You can use node selectors on a project together with labels on nodes to constrain all pods created in a namespace to the labeled nodes.

With project node selectors, when you create a pod in the namespace, OpenShift Container Platform adds the appropriate **<key>:<value>** and schedules the pod on nodes with matching labels.

You can add labels to a node or MachineConfig, but the labels will not persist if the node or machine goes down. Adding the label to the MachineSet ensures that new nodes or machines will have the label.



NOTE

You can add additional **<key>:<value>** pairs for the pod. But you cannot add a different **<value>** for a default **<key>**.

For example, if the project node selector is **region: east** the following pod spec adds a new pair and is allowed:

```
spec:
  nodeSelector:
    region: east
    type: user-node
```

The following pod spec uses a different value for **region** and is not allowed:

```
spec:
  nodeSelector:
    region: west
```

If there is a cluster-wide default node selector, a project node selector takes preference.

Procedure

To add a default project node selector:

1. Create a namespace or edit an existing namespace associated with the project to add the **openshift.io/node-selector** parameter:

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

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "type=user-node,region=east" 1
    openshift.io/sa.scc.mcs: s0:c17,c14
    openshift.io/sa.scc.supplemental-groups: 1000300000/10000
    openshift.io/sa.scc.uid-range: 1000300000/10000
  creationTimestamp: 2019-06-10T14:39:45Z
```

```
    labels:
      openshift.io/run-level: "0"
    name: demo
    resourceVersion: "401885"
    selfLink: /api/v1/namespaces/openshift-kube-apiserver
    uid: 96ecc54b-8b8d-11e9-9f54-0a9ae641edd0
  spec:
    finalizers:
    - kubernetes
  status:
    phase: Active
```

1 Add openshift.io/node-selector` with the appropriate **<key>:<value>** pairs.

2. Edit a node or MachineSet to add labels:

```
$ oc label <resource> <name> <key>=<value>
```

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

To label a MachineSet:

```
$ oc label MachineSet abc612-msrtw-worker-us-east-1c type=user-node region=east
```

When you create a pod in the namespace, OpenShift Container Platform adds the appropriate **<key>:<value>** and schedules the pod on the labeled node.

For example:

```
spec:
  nodeSelector:
    region: east
    type: user-node
```

CHAPTER 3. USING JOBS AND DAEMONSETS

3.1. RUNNING BACKGROUND TASKS ON NODES AUTOMATICALLY WITH DAEMONSETS

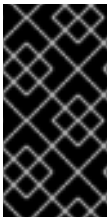
As an administrator, you can create and use DaemonSets to run replicas of a pod on specific or all nodes in an OpenShift Container Platform cluster.

A DaemonSet ensures that all (or some) nodes run a copy of a pod. As nodes are added to the cluster, pods are added to the cluster. As nodes are removed from the cluster, those pods are removed through garbage collection. Deleting a DaemonSet will clean up the Pods it created.

You can use daemonsets to create shared storage, run a logging pod on every node in your cluster, or deploy a monitoring agent on every node.

For security reasons, only cluster administrators can create daemonsets.

For more information on daemonsets, see the Kubernetes documentation.



IMPORTANT

Daemonset scheduling is incompatible with project's default node selector. If you fail to disable it, the daemonset gets restricted by merging with the default node selector. This results in frequent pod recreates on the nodes that got unselected by the merged node selector, which in turn puts unwanted load on the cluster.

3.1.1. Scheduled by default scheduler

A DaemonSet ensures that all eligible nodes run a copy of a Pod. Normally, the node that a Pod runs on is selected by the Kubernetes scheduler. However, previously daemonSet pods are created and scheduled by the DaemonSet controller. That introduces the following issues:

- Inconsistent Pod behavior: Normal Pods waiting to be scheduled are created and in Pending state, but DaemonSet pods are not created in Pending state. This is confusing to the user.
- Pod preemption is handled by default scheduler. When preemption is enabled, the DaemonSet controller will make scheduling decisions without considering pod priority and preemption.

ScheduleDaemonSetPods is enabled by default in OpenShift Container Platform which lets you to schedule DaemonSets using the default scheduler instead of the DaemonSet controller, by adding the NodeAffinity term to the DaemonSet pods, instead of the .spec.nodeName term. The default scheduler is then used to bind the pod to the target host. If node affinity of the DaemonSet pod already exists, it is replaced. The DaemonSet controller only performs these operations when creating or modifying DaemonSet pods, and no changes are made to the spec.template of the DaemonSet.

nodeAffinity: requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms: - matchFields: - key: metadata.name operator: In values: - target-host-name

In addition, node.kubernetes.io/unschedulable:NoSchedule toleration is added automatically to DaemonSet Pods. The default scheduler ignores unschedulable Nodes when scheduling DaemonSet Pods.

3.1.2. Creating daemonsets

When creating daemonsets, the **nodeSelector** field is used to indicate the nodes on which the daemonset should deploy replicas.

Prerequisites

- Before you start using daemonsets, disable the default project-wide node selector in your namespace, by setting the namespace annotation **openshift.io/node-selector** to an empty string:

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

- If you are creating a new project, overwrite the default node selector using **oc adm new-project <name> --node-selector=""**.

Procedure

To create a daemonset:

1. Define the daemonset yaml file:

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: hello-daemonset
spec:
  selector:
     matchLabels:
       name: hello-daemonset 1
  template:
    metadata:
      labels:
        name: hello-daemonset 2
    spec:
      nodeSelector: 3
        role: worker
      containers:
      - image: openshift/hello-openshift
        imagePullPolicy: Always
        name: registry
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
      serviceAccount: default
      terminationGracePeriodSeconds: 10
```

1. The label selector that determines which pods belong to the daemonset.
2. The pod template's label selector. Must match the label selector above.
3. The node selector that determines on which nodes pod replicas should be deployed. A matching label must be present on the node.

2. Create the daemonset object:

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

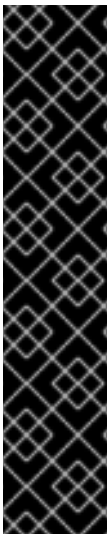
3. To verify that the pods were created, and that each node has a pod replica:

   a. Find the daemonset pods:

```
$ oc get pods
hello-daemonset-cx6md   1/1       Running   0          2m
hello-daemonset-e3md9   1/1       Running   0          2m
```

   b. View the pods to verify the pod has been placed onto the node:

```
$ oc describe pod/hello-daemonset-cx6md|grep Node
Node:        openshift-node01.hostname.com/10.14.20.134
$ oc describe pod/hello-daemonset-e3md9|grep Node
Node:        openshift-node02.hostname.com/10.14.20.137
```



IMPORTANT

- If you update a daemonset's pod template, the existing pod replicas are not affected.
- If you delete a daemonSet and then create a new daemonset with a different template but the same label selector, it recognizes any existing pod replicas as having matching labels and thus does not update them or create new replicas despite a mismatch in the pod template.
- If you change node labels, the daemonset adds pods to nodes that match the new labels and deletes pods from nodes that do not match the new labels.

To update a daemonset, force new pod replicas to be created by deleting the old replicas or nodes.

3.2. RUNNING TASKS IN PODS USING JOBS

A *job* executes a task in your OpenShift Container Platform cluster.

A job tracks the overall progress of a task and updates its status with information about active, succeeded, and failed pods. Deleting a job will clean up any pod replicas it created. Jobs are part of the Kubernetes API, which can be managed with **oc** commands like other object types.

See the Kubernetes documentation for more information about jobs.

3.2.1. Understanding jobs and CronJobs

A job tracks the overall progress of a task and updates its status with information about active, succeeded, and failed pods. Deleting a job will clean up any pods it created. Jobs are part of the Kubernetes API, which can be managed with **oc** commands like other object types.

There are two possible resource types that allow creating run-once objects in OpenShift Container Platform:

Job

A regular job is a run-once object that creates a task and ensures the job finishes.

CronJob

A CronJob can be scheduled to run multiple times, use a CronJob.

A *CronJob* builds on a regular job by allowing you to specify how the job should be run. CronJobs are part of the Kubernetes API, which can be managed with **oc** commands like other object types.

CronJobs are useful for creating periodic and recurring tasks, like running backups or sending emails. CronJobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period.

WARNING

A CronJob creates a job object approximately once per execution time of its schedule, but there are circumstances in which it fails to create a job or two jobs might be created. Therefore, jobs must be idempotent and you must configure history limits.

3.2.2. Understanding how to create jobs

Both resource types require a job configuration that consists of the following key parts:

- A pod template, which describes the pod that OpenShift Container Platform creates.
- An optional **parallelism** parameter, which specifies how many pods running in parallel at any point in time should execute a job. If not specified, this defaults to the value in the **completions** parameter.
- An optional **completions** parameter, specifying how many successful pod completions are needed to finish a job. If not specified, this value defaults to one.

3.2.2.1. Understanding how to set a maximum duration for jobs

When defining a job, you can define its maximum duration by setting the **activeDeadlineSeconds** field. It is specified in seconds and is not set by default. When not set, there is no maximum duration enforced.

The maximum duration is counted from the time when a first pod gets scheduled in the system, and defines how long a job can be active. It tracks overall time of an execution. After reaching the specified timeout, the job is terminated by OpenShift Container Platform.

3.2.2.2. Understanding how to set a job back off policy for pod failure

A Job can be considered failed, after a set amount of retries due to a logical error in configuration or other similar reasons. Failed Pods associated with the Job are recreated by the controller with an exponential back off delay (**10s**, **20s**, **40s** ...) capped at six minutes. The limit is reset if no new failed pods appear between controller checks.

Use the **spec.backoffLimit** parameter to set the number of retries for a job.

3.2.2.3. Understanding how to configure a CronJob to remove artifacts

CronJobs can leave behind artifact resources such as jobs or pods. As a user it is important to configure history limits so that old jobs and their pods are properly cleaned. There are two fields within CronJob's spec responsible for that:

- **.spec.successfulJobsHistoryLimit**. The number of successful finished jobs to retain (defaults to 3).
- **.spec.failedJobsHistoryLimit**. The number of failed finished jobs to retain (defaults to 1).

TIP

- Delete CronJobs that you no longer need:

  ```
  $ oc delete cronjob/<cron_job_name>
  ```

  Doing this prevents them from generating unnecessary artifacts.

- You can suspend further executions by setting the **spec.suspend** to true. All subsequent executions are suspended until you reset to **false**.

3.2.3. Known limitations

The job specification restart policy only applies to the *pods*, and not the *job controller*. However, the job controller is hard-coded to keep retrying jobs to completion.

As such, **restartPolicy: Never** or **--restart=Never** results in the same behavior as **restartPolicy: OnFailure** or **--restart=OnFailure**. That is, when a job fails it is restarted automatically until it succeeds (or is manually discarded). The policy only sets which subsystem performs the restart.

With the **Never** policy, the *job controller* performs the restart. With each attempt, the job controller increments the number of failures in the job status and create new pods. This means that with each failed attempt, the number of pods increases.

With the **OnFailure** policy, *kubelet* performs the restart. Each attempt does not increment the number of failures in the job status. In addition, kubelet will retry failed jobs starting pods on the same nodes.

3.2.4. Creating jobs

You create a job in OpenShift Container Platform by creating a job object.

Procedure

To create a job:

1. Create a YAML file similar to the following:

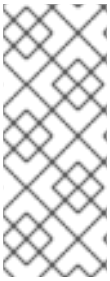
```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1    (1)
  completions: 1    (2)
```

```
  activeDeadlineSeconds: 1800 (3)
  backoffLimit: 6 (4)
  template: (5)
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure (6)
```

1. Optional value for how many pod replicas a job should run in parallel; defaults to **completions**.
2. Optional value for how many successful pod completions are needed to mark a job completed; defaults to one.
3. Optional value for the maximum duration the job can run.
4. Option value to set the number of retries for a job. This field defaults to six.
5. Template for the pod the controller creates.
6. The restart policy of the pod. This does not apply to the job controller.

2. Create the job:

```
$ oc create -f <file-name>.yaml
```



NOTE

You can also create and launch a job from a single command using **oc run**. The following command creates and launches the same job as specified in the previous example:

```
$ oc run pi --image=perl --replicas=1  --restart=OnFailure \
    --command -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

3.2.5. Creating CronJobs

You create a CronJob in OpenShift Container Platform by creating a job object.

Procedure

To create a CronJob:

1. Create a YAML file similar to the following:

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *" (1)
```

```
concurrencyPolicy: "Replace" 2
startingDeadlineSeconds: 200 3
suspend: true 4
successfulJobsHistoryLimit: 3 5
failedJobsHistoryLimit: 1 6
jobTemplate: 7
  spec:
    template:
      metadata:
        labels: 8
          parent: "cronjobpi"
      spec:
        containers:
        - name: pi
          image: perl
          command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: OnFailure 9
```

1 1 Schedule for the job specified in cron format. In this example, the job will run every minute.

2 2 An optional concurrency policy, specifying how to treat concurrent jobs within a CronJob. Only one of the following concurrent policies may be specified. If not specified, this defaults to allowing concurrent executions.

- **Allow** allows CronJobs to run concurrently.
- **Forbid** forbids concurrent runs, skipping the next run if the previous has not finished yet.
- **Replace** cancels the currently running job and replaces it with a new one.

3 3 An optional deadline (in seconds) for starting the job if it misses its scheduled time for any reason. Missed jobs executions will be counted as failed ones. If not specified, there is no deadline.

4 4 An optional flag allowing the suspension of a CronJob. If set to **true**, all subsequent executions will be suspended.

5 5 The number of successful finished jobs to retain (defaults to 3).

6 6 The number of failed finished jobs to retain (defaults to 1).

7 Job template. This is similar to the job example.

8 Sets a label for jobs spawned by this CronJob.

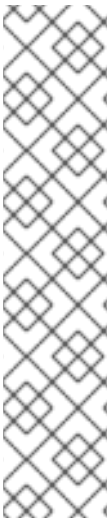
9 The restart policy of the pod. This does not apply to the job controller.

NOTE

The **.spec.successfulJobsHistoryLimit** and **.spec.failedJobsHistoryLimit** fields are optional. These fields specify how many completed and failed jobs should be kept. By default, they are set to **3** and **1** respectively. Setting a limit to **0** corresponds to keeping none of the corresponding kind of jobs after they finish.

2. Create the CronJob:

```
$ oc create -f <file-name>.yaml
```



NOTE

You can also create and launch a CronJob from a single command using **oc run**. The following command creates and launches the same CronJob as specified in the previous example:

```
$ oc run pi --image=perl --schedule='*/1 * * * *' \
    --restart=OnFailure --labels parent="cronjobpi" \
    --command -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

With **oc run**, the **--schedule** option accepts schedules in cron format.

When creating a CronJob, **oc run** only supports the **Never** or **OnFailure** restart policies (**--restart**).

CHAPTER 4. WORKING WITH NODES

4.1. VIEWING AND LISTING THE NODES IN YOUR OPENSHIFT CONTAINER PLATFORM CLUSTER

You can list all the nodes in your cluster to obtain information such as status, age, memory usage, and details about the nodes.

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.

4.1.1. About listing all the nodes in a cluster

You can get detailed information on the nodes in the cluster.

- The following command lists all nodes:

```
$ oc get nodes

NAME                 STATUS    ROLES     AGE       VERSION
master.example.com   Ready     master    7h        v1.14.6+c4799753c
node1.example.com    Ready     worker    7h        v1.14.6+c4799753c
node2.example.com    Ready     worker    7h        v1.14.6+c4799753c
```

- The **-wide** option provides additional information on all nodes.

```
$ oc get nodes -o wide
```

- The following command lists information about a single node:

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

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

Table 4.1. Node Conditions

Condition	Description
Ready	The node reports its own readiness to the apiserver by returning **True**.
NotReady	One of the underlying components, such as the container runtime or network, is experiencing issues or is not yet configured.
SchedulingDisabled	Pods cannot be scheduled for placement on the node.

- The following command provides more detailed information about a specific node, including the reason for the current condition:

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

For example:

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

Name:               node1.example.com 1
Roles:              worker 2
Labels:             beta.kubernetes.io/arch=amd64 3
                    beta.kubernetes.io/instance-type=m4.large
                    beta.kubernetes.io/os=linux
                    failure-domain.beta.kubernetes.io/region=us-east-2
                    failure-domain.beta.kubernetes.io/zone=us-east-2a
                    kubernetes.io/hostname=ip-10-0-140-16
                    node-role.kubernetes.io/worker=
Annotations:        cluster.k8s.io/machine: openshift-machine-api/ahardin-worker-us-east-2a-
q5dzc 4
                    machineconfiguration.openshift.io/currentConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/desiredConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/state: Done
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Wed, 13 Feb 2019 11:05:57 -0500
Taints:             <none> 5
Unschedulable:      false
Conditions:               6
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason
Message
  ----             ------  -----------------                 ------------------                ------
-------
  OutOfDisk        False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientDisk     kubelet has sufficient disk space available
  MemoryPressure   False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57
-0500   KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientPID      kubelet has sufficient PID available
  Ready            True    Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:07:09 -0500
KubeletReady                 kubelet is posting ready status
Addresses: 7
  InternalIP:   10.0.140.16
  InternalDNS:  ip-10-0-140-16.us-east-2.compute.internal
  Hostname:     ip-10-0-140-16.us-east-2.compute.internal
Capacity:  8
 attachable-volumes-aws-ebs:  39
 cpu:                         2
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      8172516Ki
 pods:                        250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                         1500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      7558116Ki
```

```
 pods:                    250
System Info:  9
 Machine ID:                          63787c9534c24fde9a0cde35c13f1f66
 System UUID:                         EC22BF97-A006-4A58-6AF8-0A38DEEA122A
 Boot ID:                             f24ad37d-2594-46b4-8830-7f7555918325
 Kernel Version:                      3.10.0-957.5.1.el7.x86_64
 OS Image:                            Red Hat Enterprise Linux CoreOS 410.8.20190520.0 (Ootpa)
 Operating System:                    linux
 Architecture:                        amd64
 Container Runtime Version:           cri-o://1.13.9-1.rhaos4.1.gitd70609a.el8
 Kubelet Version:                     v1.14.6+c4799753c
 Kube-Proxy Version:                  v1.14.6+c4799753c
PodCIDR:                              10.128.4.0/24
ProviderID:                           aws:///us-east-2a/i-04e87b31dc6b3e171
Non-terminated Pods:                  (13 in total)  10
  Namespace                           Name                                CPU Requests  CPU Limits
Memory Requests  Memory Limits
  ---------                           ----                                ------------  ----------  ---------------  -------
------
  openshift-cluster-node-tuning-operator  tuned-hdl5q                       0 (0%)        0 (0%)      0
(0%)          0 (0%)
  openshift-dns                       dns-default-l69zr                   0 (0%)        0 (0%)      0 (0%)
0 (0%)
  openshift-image-registry            node-ca-9hmcg                       0 (0%)        0 (0%)      0
(0%)          0 (0%)
  openshift-ingress                   router-default-76455c45c-c5ptv      0 (0%)        0 (0%)      0
(0%)          0 (0%)
  openshift-machine-config-operator   machine-config-daemon-cvqw9          20m (1%)      0
(0%)      50Mi (0%)       0 (0%)
  openshift-marketplace               community-operators-f67fh           0 (0%)        0 (0%)
0 (0%)          0 (0%)
  openshift-monitoring                alertmanager-main-0                 50m (3%)      50m (3%)
210Mi (2%)      10Mi (0%)
  openshift-monitoring                grafana-78765ddcc7-hnjmm            100m (6%)     200m
(13%)  100Mi (1%)      200Mi (2%)
  openshift-monitoring                node-exporter-l7q8d                 10m (0%)      20m (1%)
20Mi (0%)       40Mi (0%)
  openshift-monitoring                prometheus-adapter-75d769c874-hvb85  0 (0%)       0
(0%)      0 (0%)          0 (0%)
  openshift-multus                    multus-kw8w5                        0 (0%)        0 (0%)      0 (0%)
0 (0%)
  openshift-sdn                       ovs-t4dsn                           100m (6%)     0 (0%)      300Mi
(4%)      0 (0%)
  openshift-sdn                       sdn-g79hg                           100m (6%)     0 (0%)      200Mi
(2%)      0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                    Requests     Limits
  --------                    --------     ------
  cpu                         380m (25%)   270m (18%)
  memory                      880Mi (11%)  250Mi (3%)
  attachable-volumes-aws-ebs  0            0
Events:  11
  Type     Reason                   Age                From                    Message
  ----     ------                   ----               ----                    -------
  Normal   NodeHasSufficientPID     6d (x5 over 6d)    kubelet, m01.example.com  Node
```

```
m01.example.com status is now: NodeHasSufficientPID
  Normal  NodeAllocatableEnforced  6d                 kubelet, m01.example.com  Updated Node
Allocatable limit across pods
  Normal  NodeHasSufficientMemory  6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientMemory
  Normal  NodeHasNoDiskPressure    6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasNoDiskPressure
  Normal  NodeHasSufficientDisk    6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientDisk
  Normal  NodeHasSufficientPID     6d                 kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientPID
  Normal  Starting                 6d                 kubelet, m01.example.com  Starting kubelet.
  ...
```

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

4.1.2. Listing pods on a node in your cluster

You can list all the pods on a specific node.

Procedure

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

```
$ oc describe node <node1> <node2>
```

  For example:

```
$ oc describe node ip-10-0-128-218.ec2.internal
```

- To list all or selected pods on selected nodes:

```
$ oc describe --selector=<node_selector>
$ oc describe -l=<pod_selector>
```

For example:

```
$ oc describe node  --selector=beta.kubernetes.io/os
$ oc describe node -l node-role.kubernetes.io/worker
```

4.1.3. Viewing memory and CPU usage statistics on your nodes

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

Prerequisites

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

Procedure

- To view the usage statistics:

```
$ oc adm top nodes

NAME                                CPU(cores)   CPU%      MEMORY(bytes)   MEMORY%
ip-10-0-12-143.ec2.compute.internal   1503m        100%      4533Mi          61%
ip-10-0-132-16.ec2.compute.internal   76m          5%        1391Mi          18%
ip-10-0-140-137.ec2.compute.internal  398m         26%       2473Mi          33%
ip-10-0-142-44.ec2.compute.internal   656m         43%       6119Mi          82%
ip-10-0-146-165.ec2.compute.internal  188m         12%       3367Mi          45%
ip-10-0-19-62.ec2.compute.internal    896m         59%       5754Mi          77%
ip-10-0-44-193.ec2.compute.internal   632m         42%       5349Mi          72%
```

- To view the usage statistics for nodes with labels:

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

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

4.2. WORKING WITH NODES

As an administrator, you can perform a number of tasks to make your clusters more efficient.

4.2.1. Understanding how to evacuate pods on nodes

Evacuating pods allows you to migrate all or selected pods from a given node or nodes.

You can only evacuate pods backed by a replication controller. The replication controllers create new pods on other nodes and removes the existing pods from the specified node(s).

Bare pods, meaning those not backed by a replication controller, are unaffected by default. You can evacuate a subset of pods by specifying a pod-selector. Pod selectors are based on labels, so all the pods with the specified label will be evacuated.

NOTE

Nodes must first be marked unschedulable to perform pod evacuation.

```
$ oc adm cordon <node1>
NAME       STATUS                        ROLES    AGE      VERSION
<node1>    NotReady,SchedulingDisabled   worker   1d       v1.14.6+c4799753c
```

Use **oc adm uncordon** to mark the node as schedulable when done.

```
$ oc adm uncordon <node1>
```

- The following command evacuates all or selected pods on one or more nodes:

```
$ oc adm drain <node1> <node2> [--pod-selector=<pod_selector>]
```

- The following command forces deletion of bare pods using the **--force** option. When set to **true**, deletion continues even if there are pods not managed by a replication controller, ReplicaSet, job, daemonset, or StatefulSet:

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

- The following command sets a period of time in seconds for each pod to terminate gracefully, use **--grace-period**. If negative, the default value specified in the pod will be used:

```
$ oc adm drain <node1> <node2> --grace-period=-1
```

- The following command ignores DaemonSet-managed pods using the **--ignore-daemonsets** flag set to **true**:

```
$ oc adm drain <node1> <node2> --ignore-daemonsets=true
```

- The following command sets the length of time to wait before giving up using the **--timeout** flag. A value of **0** sets an infinite length of time:

```
$ oc adm drain <node1> <node2> --timeout=5s
```

- The following command deletes pods even if there are pods using emptyDir using the **--delete-local-data** flag set to **true**. Local data is deleted when the node is drained:

```
$ oc adm drain <node1> <node2> --delete-local-data=true
```

- The following command lists objects that will be migrated without actually performing the evacuation, using the **--dry-run** option set to **true**:

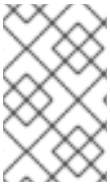
```
$ oc adm drain <node1> <node2>  --dry-run=true
```

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

4.2.2. Understanding how to update labels on nodes

You can update any label on a node.

Node labels are not persisted after a node is deleted even if the node is backed up by a Machine.



NOTE

Any change to a MachineSet is not applied to existing machines owned by the MachineSet. For example, labels edited or added to an existing MachineSet are not propagated to existing machines and Nodes associated with the MachineSet.

- The following command adds or updates labels on a node:

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

For example:

```
$ oc label nodes webconsole-7f7f6 unhealthy=true
```

- The following command updates all pods in the namespace:

```
$ oc label pods --all <key_1>=<value_1>
```

For example:

```
$ oc label pods --all status=unhealthy
```

4.2.3. Understanding how to mark 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.

- The following command marks a node or nodes as unschedulable:

```
$ oc adm cordon <node>
```

For example:

```
$ oc adm cordon node1.example.com
node/node1.example.com cordoned

NAME                LABELS                                        STATUS
node1.example.com   kubernetes.io/hostname=node1.example.com
Ready,SchedulingDisabled
```

- The following command marks a currently unschedulable node or nodes as schedulable:

```
$ oc adm uncordon <node1>
```

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

4.2.4. Configuring master nodes as schedulable

As of OpenShift Container Platform 4.2, you can configure master nodes to be schedulable, meaning that new Pods are allowed for placement on the master nodes. By default, master nodes are not schedulable. However, if your cluster does not contain any worker nodes, then master nodes are marked schedulable by default.

You can allow or disallow master nodes to be schedulable by configuring the **mastersSchedulable** field.

Procedure

1. Edit the **schedulers.config.openshift.io** resource.

```
$ oc edit schedulers.config.openshift.io cluster
```

2. Configure the **mastersSchedulable** field.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: "2019-09-10T03:04:05Z"
  generation: 1
  name: cluster
  resourceVersion: "433"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: a636d30a-d377-11e9-88d4-0a60097bee62
spec:
  mastersSchedulable: false (1)
  policy:
    name: ""
status: {}
```

(1) Set to **true** to allow master nodes to be schedulable, or **false** to disallow master nodes to be schedulable.

3. Save the file to apply the changes.

4.2.5. Deleting nodes from a cluster

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

Procedure

To delete a node from the OpenShift Container Platform cluster edit the appropriate MachineSet:

1. View the MachineSets that are in the cluster:

```
$ oc get machinesets -n openshift-machine-api
```

The MachineSets are listed in the form of <clusterid>-worker-<aws-region-az>.

2. Scale the MachineSet:

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

For more information on scaling your cluster using a MachineSet, see Manually scaling a MachineSet.

4.2.6. Adding kernel arguments to Nodes

In some special cases, you might want to add kernel arguments to a set of nodes in your cluster. This should only be done with caution and clear understanding of the implications of the arguments you set.

WARNING

Improper use of kernel arguments can result in your systems becoming unbootable.

Examples of kernel arguments you could set include:

- **selinux=0**: Disables Security Enhanced Linux (SELinux). While not recommended for production, disabling SELinux can improve performance by 2% - 3%.
- **nosmt**: Disables symmetric multithreading (SMT) in the kernel. Multithreading allows multiple logical threads for each CPU. You could consider **nosmt** in multi-tenant environments to reduce risks from potential cross-thread attacks. By disabling SMT, you essentially choose security over performance.

See Kernel.org kernel parameters for a list and descriptions of kernel arguments.

In the following procedure, you create a MachineConfig that identifies:

- A set of machines to which you want to add the kernel argument. In this case, machines with a worker role.
- Kernel arguments that are appended to the end of the existing kernel arguments.
- A label that indicates where in the list of MachineConfigs the change is applied.

Prerequisites

- Have administrative privilege to a working OpenShift Container Platform cluster.

Procedure

1. List existing MachineConfigs for your OpenShift Container Platform cluster to determine how to label your MachineConfig:

```
$ oc get MachineConfig
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                          577c2d527b09cd7a481a162c50592139caa15e20
2.2.0             30m
```

```
00-worker                                          577c2d527b09cd7a481a162c50592139caa15e20
2.2.0             30m
01-master-container-runtime
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
01-master-kubelet
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
01-worker-container-runtime
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
01-worker-kubelet
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
99-master-1131169f-dae9-11e9-b5dd-12a845e8ffd8-registries
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
99-master-ssh                                                 2.2.0             30m
99-worker-114e8ac7-dae9-11e9-b5dd-12a845e8ffd8-registries
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
99-worker-ssh                                                 2.2.0             30m
rendered-master-b3729e5f6124ca3678188071343115d0
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
rendered-worker-18ff9506c718be1e8bd0a066850065b7
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             30m
```

2. Create a MachineConfig file that identifies the kernel argument (for example, **05-worker-kernelarg-selinuxoff.yaml**)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 05-worker-kernelarg-selinuxoff (2)
spec:
  config:
    ignition:
      version: 2.2.0
  kernelArguments:
    - selinux=0 (3)
```

(1) Applies the new kernel argument only to worker nodes.

(2) Named to identify where it fits among the MachineConfigs (05) and what it does (adds a kernel argument to turn off SELinux).

(3) Identifies the exact kernel argument as **selinux=0**.

3. Create the new MachineConfig:

```
$ oc create -f 05-worker-kernelarg-selinuxoff.yaml
```

4. Check the MachineConfigs to see that the new one was added:

```
$ oc get MachineConfig
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                          577c2d527b09cd7a481a162c50592139caa15e20
```

```
2.2.0             31m
00-worker                                          577c2d527b09cd7a481a162c50592139caa15e20
2.2.0             31m
01-master-container-runtime
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
01-master-kubelet
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
01-worker-container-runtime
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
01-worker-kubelet
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m

05-worker-kernelarg-selinuxoff                                                 2.2.0
105s

99-master-1131169f-dae9-11e9-b5dd-12a845e8ffd8-registries
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
99-master-ssh                                                  2.2.0             30m
99-worker-114e8ac7-dae9-11e9-b5dd-12a845e8ffd8-registries
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
99-worker-ssh                                                  2.2.0             31m
rendered-master-b3729e5f6124ca3678188071343115d0
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
rendered-worker-18ff9506c718be1e8bd0a066850065b7
577c2d527b09cd7a481a162c50592139caa15e20   2.2.0             31m
```

5. Check the nodes:

```
$ oc get node
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.14.6+90fadebfa
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.14.6+90fadebfa
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.14.6+90fadebfa
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.14.6+90fadebfa
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.14.6+90fadebfa
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.14.6+90fadebfa
```

You can see that scheduling on each worker node is disabled as the change is being applied.

6. Check that the kernel argument worked by going to one of the worker nodes and listing the kernel command line arguments (in **/proc/cmdline** on the host):

```
$ oc debug node/ip-10-0-141-105.ec2.internal
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 selinux=0

sh-4.2# exit
```

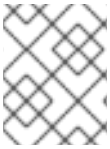
You should see the **selinux=0** argument added to the other kernel arguments.

4.2.7. Additional resources

For more information on scaling your cluster using a MachineSet, see Manually scaling a MachineSet .

4.3. MANAGING NODES

OpenShift Container Platform uses a KubeletConfig Custom Resource to manage the configuration of nodes. By creating an instance of a KubeletConfig, a managed MachineConfig is created to override setting on the node.



NOTE

Logging in to remote machines for the purpose of changing their configuration is not supported.

4.3.1. Modifying Nodes

To make configuration changes to a cluster, or MachinePool, you must create a Custom Resource Definition, or KubeletConfig instance. OpenShift Container Platform uses the Machine Config Controller to watch for changes introduced through the CRD applies the changes to the cluster.

Procedure

1. Obtain the label associated with the static CRD, Machine Config Pool, for the type of node you want to configure. Perform one of the following steps:

   a. Check current labels of the desired machineconfigpool.
   For example:

```
$ oc get machineconfigpool --show-labels
NAME     CONFIG                                    UPDATED   UPDATING   DEGRADED
LABELS
master   rendered-master-e05b81f5ca4db1d249a1bf32f9ec24fd   True      False
False    operator.machineconfiguration.openshift.io/required-for-upgrade=
worker   rendered-worker-f50e78e1bc06d8e82327763145bfcf62   True      False
False
```

   b. Add a custom label to the desired machineconfigpool.
   For example:

```
$ oc label machineconfigpool worker custom-kubelet=enabled
```

2. Create a KubeletConfig Custom Resource (CR) for your configuration change.
   For example:

   Sample configuration for a custom-config CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config (1)
spec:
  machineConfigPoolSelector:
```

```
    matchLabels:
      custom-kubelet: enabled 2
  kubeletConfig: 3
    podsPerCore: 10
    maxPods: 250
    systemReserved:
      cpu: 1000m
      memory: 500Mi
    kubeReserved:
      cpu: 1000m
      memory: 500Mi
```

1. Assign a name to CR.
2. Specify the label to apply the configuration change, this is the label you added to the machineconfigpool.
3. Specify the new value(s) you want to change.

3. Create the CR object.

```
$ oc create -f <file-name>
```

For example:

```
$ oc create -f master-kube-config.yaml
```

Most KubeletConfig Options may be set by the user. The following options are not allowed to be overwritten:

- CgroupDriver
- ClusterDNS
- ClusterDomain
- RuntimeRequestTimeout
- StaticPodPath

4.4. MANAGING THE MAXIMUM NUMBER OF PODS PER NODE

In OpenShift Container Platform, you can configure the number of pods that can run on a node based on the number of processor cores on the node, a hard limit or both. If you use both options, the lower of the two limits the number of pods on a node.

Exceeding these values can result in:

- Increased CPU utilization by OpenShift Container Platform.
- Slow pod scheduling.
- Potential out-of-memory scenarios, depending on the amount of memory in the node.

- Exhausting the IP address pool.
- Resource overcommitting, leading to poor user application performance.

NOTE

A pod that is holding a single container actually uses two containers. The second container sets up networking prior to the actual container starting. As a result, a node running 10 pods actually has 20 containers running.

The **podsPerCore** parameter limits the number of pods the node can run based on the number of processor cores on the node. For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node is 40.

The **maxPods** parameter limits the number of pods the node can run to a fixed value, regardless of the properties of the node.

4.4.1. Configuring the maximum number of Pods per Node

Two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**. If you use both options, the lower of the two limits the number of pods on a node.

For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be 40.

Prerequisite

1. Obtain the label associated with the static Machine Config Pool CRD for the type of node you want to configure. Perform one of the following steps:

   a. View the Machine Config Pool:

   ```
   $ oc describe machineconfigpool <name>
   ```

   For example:

   ```
   $ oc describe machineconfigpool worker

   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: small-pods 1
   ```

   1 If a label has been added it appears under **labels**.

   b. If the label is not present, add a key/value pair:

   ```
   $ oc label machineconfigpool worker custom-kubelet=small-pods
   ```

Procedure

1. Create a Custom Resource (CR) for your configuration change.

Sample configuration for a max-pods CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods (2)
  kubeletConfig:
    podsPerCore: 10 (3)
    maxPods: 250 (4)
```

(1) Assign a name to CR.

(2) Specify the label to apply the configuration change.

(3) Specify the number of pods the node can run based on the number of processor cores on the node.

(4) Specify the number of pods the node can run to a fixed value, regardless of the properties of the node.

NOTE

Setting **podsPerCore** to 0 disables this limit.

In the above example, the default value for **podsPerCore** is **10** and the default value for **maxPods** is **250**. This means that unless the node has 25 cores or more, by default, **podsPerCore** will be the limiting factor.

2. List the Machine Config Pool CRDs to see if the change is applied. The **UPDATING** column reports **True** if the change is picked up by the Machine Config Controller:

```
$ oc get machineconfigpools
NAME     CONFIG                    UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     False      False
worker   worker-8cecd1236b33ee3f8a5e   False     True       False
```

Once the change is complete, the **UPDATED** column reports **True**.

```
$ oc get machineconfigpools
NAME     CONFIG                    UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     True       False
worker   worker-8cecd1236b33ee3f8a5e   True      False      False
```

4.5. USING THE NODE TUNING OPERATOR

Learn about the Node Tuning Operator and how you can use it to manage node-level tuning by orchestrating the tuned daemon.

4.5.1. About the Node Tuning Operator

The Node Tuning Operator helps you manage node-level tuning by orchestrating the tuned daemon. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs. The Operator manages the containerized tuned daemon for OpenShift Container Platform as a Kubernetes DaemonSet. It ensures the custom tuning specification is passed to all containerized tuned daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized tuned daemon are rolled back on an event that triggers a profile change or when the containerized tuned daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.

4.5.2. Accessing an example Node Tuning Operator specification

Use this process to access an example Node Tuning Operator specification.

Procedure

1. Run:

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

Note the default CR is meant for delivering standard node-level tuning for the OpenShift Container Platform platform and any custom changes to the default CR will be overwritten by the Operator. For custom tuning, create your own tuned CRs. Newly created CRs will be combined with the default CR and custom tuning applied to OpenShift Container Platform nodes based on node/pod labels and profile priorities.

4.5.3. Custom tuning specification

The custom resource (CR) for the operator has two major sections. The first section, **profile:**, is a list of tuned profiles and their names. The second, **recommend:**, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the operator's namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized tuned daemons are updated.

Profile data

The **profile:** section lists tuned profiles and their names.

```
profile:
- name: tuned_profile_1
  data: |
    # Tuned profile specification
    [main]
    summary=Description of tuned_profile_1 profile
```

```
    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other tuned daemon plugins supported by the containerized tuned

# ...

- name: tuned_profile_n
  data: |
    # Tuned profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

The **profile:** selection logic is defined by the **recommend:** section of the CR:

```
recommend:
- match:                          # optional; if omitted, profile match is assumed unless a profile with a
higher matches first
  <match>                          # an optional array
  priority: <priority>              # profile ordering priority, lower numbers mean higher priority (0 is the
highest priority)
  profile: <tuned_profile_name>       # e.g. tuned_profile_1

# ...

- match:
  <match>
  priority: <priority>
  profile: <tuned_profile_name>       # e.g. tuned_profile_n
```

If **<match>** is omitted, a profile match (for example, **true**) is assumed.

<match> is an optional array recursively defined as follows:

```
- label: <label_name>     # node or pod label name
  value: <label_value>    # optional node or pod label value; if omitted, the presence of <label_name>
is enough to match
  type: <label_type>      # optional node or pod type ("node" or "pod"); if omitted, "node" is assumed
  <match>                # an optional <match> array
```

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** array matches, the entire **<match>** array evaluates to **true**. Therefore, the array acts as logical OR operator.

Example

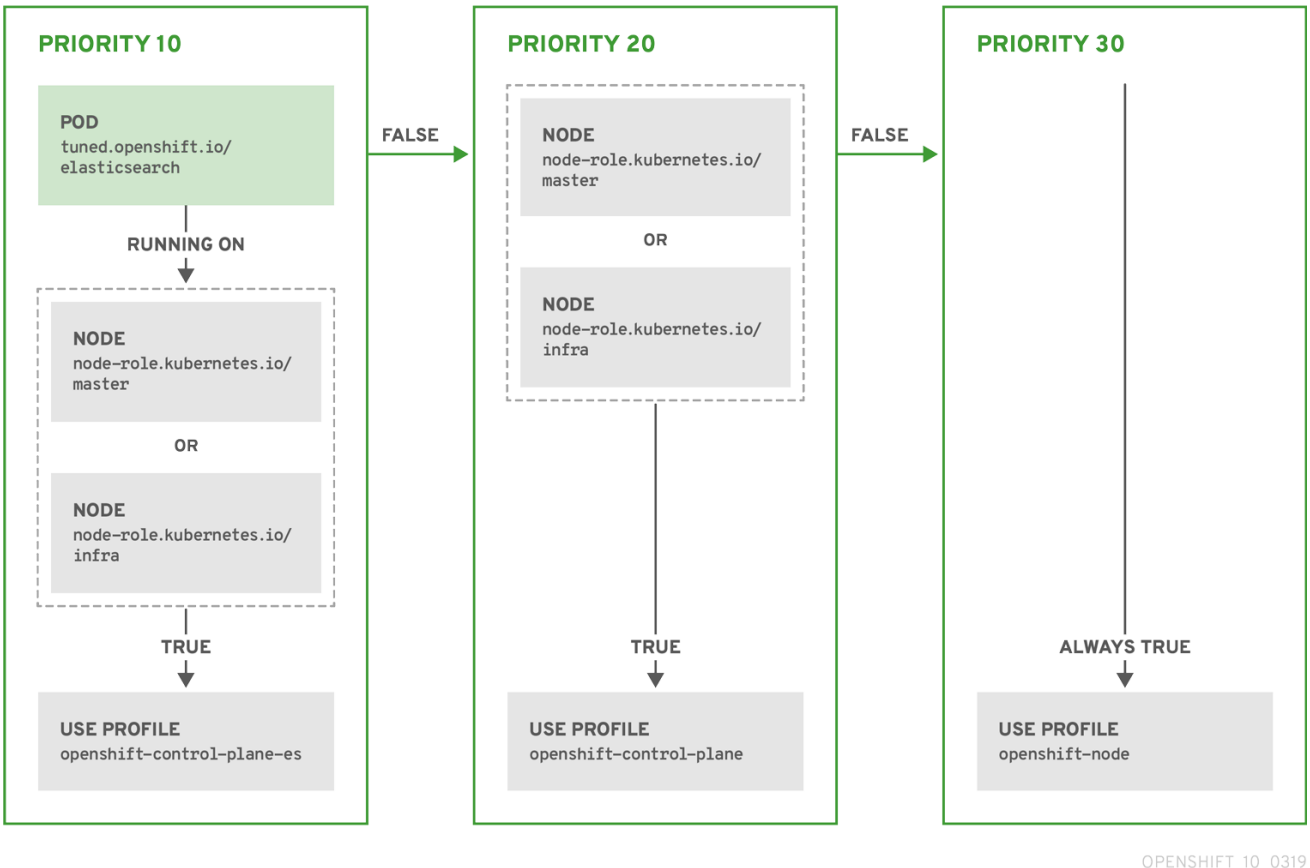
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
```

```
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

The CR above is translated for the containerized tuned daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized tuned daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized tuned pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.



4.5.4. Default profiles set on a cluster

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1alpha1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - name: "openshift"
    data: |
      [main]
      summary=Optimize systems running OpenShift (parent profile)
      include=${f:virt_check:virtual-guest:throughput-performance}
      [selinux]
      avc_cache_threshold=8192
      [net]
      nf_conntrack_hashsize=131072
      [sysctl]
      net.ipv4.ip_forward=1
      kernel.pid_max=>131072
      net.netfilter.nf_conntrack_max=1048576
      net.ipv4.neigh.default.gc_thresh1=8192
      net.ipv4.neigh.default.gc_thresh2=32768
      net.ipv4.neigh.default.gc_thresh3=65536
      net.ipv6.neigh.default.gc_thresh1=8192
      net.ipv6.neigh.default.gc_thresh2=32768
      net.ipv6.neigh.default.gc_thresh3=65536
      [sysfs]
      /sys/module/nvme_core/parameters/io_timeout=4294967295
      /sys/module/nvme_core/parameters/max_retries=10
  - name: "openshift-control-plane"
    data: |
      [main]
      summary=Optimize systems running OpenShift control plane
      include=openshift
      [sysctl]
      # ktune sysctl settings, maximizing i/o throughput
      #
      # Minimal preemption granularity for CPU-bound tasks:
      # (default: 1 msec#  (1 + ilog(ncpus)), units: nanoseconds)
      kernel.sched_min_granularity_ns=10000000
      # The total time the scheduler will consider a migrated process
      # "cache hot" and thus less likely to be re-migrated
      # (system default is 500000, i.e. 0.5 ms)
      kernel.sched_migration_cost_ns=5000000
      # SCHED_OTHER wake-up granularity.
      #
      # Preemption granularity when tasks wake up.  Lower the value to
      # improve wake-up latency and throughput for latency critical tasks.
      kernel.sched_wakeup_granularity_ns=4000000
  - name: "openshift-node"
    data: |
      [main]
```

```
      summary=Optimize systems running OpenShift nodes
      include=openshift
      [sysctl]
      net.ipv4.tcp_fastopen=3
      fs.inotify.max_user_watches=65536
  - name: "openshift-control-plane-es"
    data: |
      [main]
      summary=Optimize systems running ES on OpenShift control-plane
      include=openshift-control-plane
      [sysctl]
      vm.max_map_count=262144
  - name: "openshift-node-es"
    data: |
      [main]
      summary=Optimize systems running ES on OpenShift nodes
      include=openshift-node
      [sysctl]
      vm.max_map_count=262144
  recommend:
  - profile: "openshift-control-plane-es"
    priority: 10
    match:
    - label: "tuned.openshift.io/elasticsearch"
      type: "pod"
      match:
      - label: "node-role.kubernetes.io/master"
      - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node-es"
    priority: 20
    match:
    - label: "tuned.openshift.io/elasticsearch"
      type: "pod"

  - profile: "openshift-control-plane"
    priority: 30
    match:
    - label: "node-role.kubernetes.io/master"
    - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node"
  priority: 40
```

4.5.5. Supported Tuned daemon plug-ins

Excluding the **[main]** section, the following Tuned plug-ins are supported when using custom profiles defined in the **profile:** section of the Tuned CR:

- audio
- cpu
- disk
- eeepc_she

- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm

There is some dynamic tuning functionality provided by some of these plug-ins that is not supported. The following Tuned plug-ins are currently not supported:

- bootloader
- script
- systemd

See Available Tuned Plug-ins and Getting Started with Tuned for more information.

4.6. UNDERSTANDING NODE REBOOTING

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.

4.6.1. Understanding infrastructure node rebooting

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 nodes to run the infrastructure are called **master** nodes.

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 master nodes for infrastructure 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.

4.6.2. Rebooting a node using pod anti-affinity

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

With this in place, if only two infrastructure nodes are available and one is rebooted, the container image 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.

Procedure

To reboot a node using pod anti-affinity:

1. Edit the node specification to configure pod anti-affinity:

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: registry 4
              operator: In 5
              values:
              - default
          topologyKey: kubernetes.io/hostname
```

1. Stanza to configure pod anti-affinity.
2. Defines a preferred rule.
3. Specifies a weight for a preferred rule. The node with the highest weight is preferred.
4. Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.

5 The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be

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

2. Enable the **MatchInterPodAffinity** scheduler predicate in the scheduling policy file. For more information, see Configuring the Default Scheduler .

4.6.3. Understanding how to reboot nodes running routers

In most cases, a pod running an OpenShift Container Platform router exposes 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.

4.7. FREEING NODE RESOURCES USING GARBAGE COLLECTION

As an administrator, you can use OpenShift Container Platform to ensure that your nodes are running efficiently by freeing up resources through garbage collection.

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

- Container garbage collection: Removes terminated containers.
- Image garbage collection: Removes images not referenced by any running pods.

4.7.1. Understanding how terminated containers are removed though garbage collection

Container garbage collection can be performed using eviction thresholds.

When eviction thresholds are set for garbage collection, the node tries to keep any container for any pod accessible from the API. If the pod has been deleted, the containers will be as well. Containers are preserved as long the pod is not deleted and the eviction threshold is not reached. If the node is under disk pressure, it will remove containers and their logs will no longer be accessible using **oc logs**.

- **eviction-soft** - A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period.
- **eviction-hard** - A hard eviction threshold has no grace period, and if observed, OpenShift Container Platform takes immediate action.

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node would constantly oscillate between **true** and **false**. As a consequence, the scheduler could make poor scheduling decisions.

To protect against this oscillation, use the **eviction-pressure-transition-period** flag to control how long

OpenShift Container Platform must wait before transitioning out of a pressure condition. OpenShift Container Platform will not set an eviction threshold as being met for the specified pressure condition for the period specified before toggling the condition back to false.

4.7.2. Understanding how images are removed though 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.

The policy for image garbage collection is based on two conditions:

- The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is **85**.
- The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is **80**.

For image garbage collection, you can modify any of the following variables using a Custom Resource.

Table 4.2. Variables for configuring image garbage collection

Setting	Description
imageMinimumGCAge	The minimum age for an unused image before the image is removed by garbage collection. The default is **2m**.
imageGCHighThresholdPercent	The percent of disk usage, expressed as an integer, which triggers image garbage collection. The default is **85**.
imageGCLowThresholdPercent	The percent of disk usage, expressed as an integer, to which image garbage collection attempts to free. The default is **80**.

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

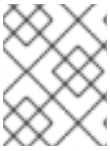
1. A list of images currently running in at least one pod.
2. A list of images available on a host.

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.

4.7.3. Configuring garbage collection for containers and images

As an administrator, you can configure how OpenShift Container Platform performs garbage collection by creating a **kubeletConfig** object for each Machine Config Pool.



NOTE

OpenShift Container Platform supports only one **kubeletConfig** object for each Machine Config Pool.

You can configure any combination of the following:

- soft eviction for containers
- hard eviction for containers
- eviction for images

For soft container eviction you can also configure a grace period before eviction.

Prerequisites

1. Obtain the label associated with the static Machine Config Pool CRD for the type of node you want to configure. Perform one of the following steps:

   a. View the Machine Config Pool:

   ```
   $ oc describe machineconfigpool <name>
   ```

   For example:

   ```
   $ oc describe machineconfigpool worker

   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: small-pods 1
   ```

   1 If a label has been added it appears under **labels**.

   b. If the label is not present, add a key/value pair:

   ```
   $ oc label machineconfigpool worker custom-kubelet=small-pods
   ```

Procedure

1. Create a Custom Resource (CR) for your configuration change.

   Sample configuration for a container garbage collection CR:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: worker-kubeconfig 1
   spec:
   ```

```
machineConfigPoolSelector:
  matchLabels:
    custom-kubelet: small-pods (2)
kubeletConfig:
  evictionSoft: (3)
  evictionSoft:
    memory.available: "500Mi" (4)
    nodefs.available: "10%"
    nodefs.inodesFree: "5%"
    imagefs.available: "15%"
    imagefs.inodesFree: "10%"
  evictionSoftGracePeriod: (5)
    memory.available: "1m30s"
    nodefs.available: "1m30s"
    nodefs.inodesFree: "1m30s"
    imagefs.available: "1m30s"
    imagefs.inodesFree: "1m30s"
  evictionHard:
    memory.available: "200Mi"
    nodefs.available: "5%"
    nodefs.inodesFree: "4%"
    imagefs.available: "10%"
    imagefs.inodesFree: "5%"
  evictionPressureTransitionPeriod: 0s (6)
  imageMinimumGCAge: 5m (7)
  imageGCHighThresholdPercent: 80 (8)
  imageGCLowThresholdPercent: 75 (9)
```

(1) Name for the object.

(2) Selector label.

(3) Type of eviction: **EvictionSoft** and **EvictionHard**.

(4) Eviction thresholds based on a specific eviction trigger signal.

(5) Grace periods for the soft eviction. This parameter does not apply to **eviction-hard**.

(6) The duration to wait before transitioning out of an eviction pressure condition

(7) The minimum age for an unused image before the image is removed by garbage collection.

(8) The percent of disk usage (expressed as an integer) which triggers image garbage collection.

(9) The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free.

2. Create the object:

```
$ oc create -f <file-name>.yaml
```

For example:

```
oc create -f gc-container.yaml

kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

3. Verify that garbage collection is active. The Machine Config Pool you specified in the custom resource appears with **UPDATING** as 'true` until the change is fully implemented:

```
$ oc get machineconfigpool

NAME     CONFIG                                UPDATED   UPDATING
master   rendered-master-546383f80705bd5aeaba93   True      False
worker   rendered-worker-b4c51bb33ccaae6fc4a6a5   False     True
```

4.8. ALLOCATING RESOURCES FOR NODES IN AN OPENSHIFT CONTAINER PLATFORM CLUSTER

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 (such as kubelet, kube-proxy) and the remaining system components (such as **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.

4.8.1. Understanding how to allocate resources for nodes

CPU and memory resources reserved for node components in OpenShift Container Platform are based on two node settings:

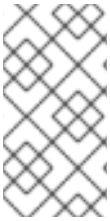
Setting	Description
kube-reserved	Resources reserved for node components. Default is none.
system-reserved	Resources reserved for the remaining system components. Default is none.

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.

4.8.1.1. How OpenShift Container Platform computes allocated resources

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

```
[Allocatable] = [Node Capacity] - [kube-reserved] - [system-reserved] - [Hard-Eviction-Thresholds]
```



NOTE

The withholding of **Hard-Eviction-Thresholds** from allocatable is a change in behavior to improve system reliability now that allocatable is enforced for end-user pods at the node level. The **experimental-allocatable-ignore-eviction** setting is available to preserve legacy behavior, but it will be deprecated in a future release.

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

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*.

4.8.1.2. How nodes enforce resource constraints

The node is able to limit the total amount of resources that pods may consume based on the configured allocatable value. This feature significantly improves the reliability of the node by preventing pods from starving system services (for example: container runtime, node agent, etc.) for resources. It is strongly encouraged that administrators reserve resources based on the desired node utilization target in order to improve node reliability.

The node enforces resource constraints using a new **cgroup** hierarchy that enforces quality of service. All pods are launched in a dedicated cgroup hierarchy separate from system daemons.

Optionally, the node can be made to enforce kube-reserved and system-reserved by specifying those tokens in the enforce-node-allocatable flag. If specified, the corresponding **--kube-reserved-cgroup** or **--system-reserved-cgroup** needs to be provided. In future releases, the node and container runtime will be packaged in a common cgroup separate from **system.slice**. Until that time, we do not recommend users change the default value of enforce-node-allocatable flag.

Administrators should treat system daemons similar to Guaranteed pods. System daemons can burst within their bounding control groups and this behavior needs to be managed as part of cluster deployments. Enforcing system-reserved limits can lead to critical system services being CPU starved or OOM killed on the node. The recommendation is to enforce system-reserved only if operators have profiled their nodes exhaustively to determine precise estimates and are confident in their ability to recover if any process in that group is OOM killed.

As a result, we strongly recommended that users only enforce node allocatable for **pods** by default, and set aside appropriate reservations for system daemons to maintain overall node reliability.

4.8.1.3. Understanding Eviction Thresholds

If a node is under memory pressure, it can impact the entire node and all pods running on it. If a system daemon is using more than its reserved amount of memory, an OOM event may occur that can impact the entire node and all pods running on it. To avoid (or reduce the probability of) system OOMs the node provides out-of-resource handling.

You can reserve some memory using the **--eviction-hard** flag. The node attempts to evict pods whenever memory availability on the node drops below the absolute value or percentage. If system daemons do not exist on a node, pods are limited to the memory **capacity - eviction-hard**. For this reason, resources set aside as a buffer for eviction before reaching out of memory conditions are not available for pods.

The following is an example to illustrate the impact of node allocatable for memory:

- Node capacity is **32Gi**
- --kube-reserved is **2Gi**
- --system-reserved is **1Gi**
- --eviction-hard is set to **100Mi**.

For this node, the effective node allocatable value is **28.9Gi**. If the node and system components use up all their reservation, the memory available for pods is **28.9Gi**, and kubelet will evict pods when it exceeds this usage.

If you enforce node allocatable (**28.9Gi**) via top level cgroups, then pods can never exceed **28.9Gi**. Evictions would not be performed unless system daemons are consuming more than **3.1Gi** of memory.

If system daemons do not use up all their reservation, with the above example, pods would face memcg OOM kills from their bounding cgroup before node evictions kick in. To better enforce QoS under this situation, the node applies the hard eviction thresholds to the top-level cgroup for all pods to be **Node Allocatable + Eviction Hard Thresholds**.

If system daemons do not use up all their reservation, the node will evict pods whenever they consume more than **28.9Gi** of memory. If eviction does not occur in time, a pod will be OOM killed if pods consume **29Gi** of memory.

4.8.1.4. How the scheduler determines resource availability

The scheduler 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.

4.8.2. Configuring allocated resources for nodes

OpenShift Container Platform supports the CPU and memory resource types for allocation. If your administrator enabled the ephemeral storage technology preview, the **ephemeral-resource** resource type is supported as well. For the **cpu** type, the resource quantity is specified in units of cores, such as **200m**, **0.5**, or **1**. For **memory** and **ephemeral-storage**, it is specified in units of bytes, such as **200Ki**, **50Mi**, or **5Gi**.

As an administrator, you can set these using a Custom Resource (CR) through a set of **<resource_type>=<resource_quantity>** pairs (e.g., **cpu=200m,memory=512Mi**).

Prerequisites

1. To help you determine setting for **--system-reserved** and **--kube-reserved** you can introspect the corresponding node's resource usage using the node summary API, which is accessible at ***<master>/api/v1/nodes/<node>/proxy/stats/summary***. Run the following command for your node:

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

   The *REST API Overview* has details about certificate details.

   For example, 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"
        }
      ]
    }
}
```


2. Obtain the label associated with the static Machine Config Pool CRD for the type of node you want to configure. Perform one of the following steps:

   a. View the Machine Config Pool:

   ```
   $ oc describe machineconfigpool <name>
   ```

   For example:

   ```
   $ oc describe machineconfigpool worker

   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: small-pods 1
   ```

   1 If a label has been added it appears under **labels**.

   b. If the label is not present, add a key/value pair:

   ```
   $ oc label machineconfigpool worker custom-kubelet=small-pods
   ```

Procedure

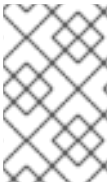
1. Create a Custom Resource (CR) for your configuration change.

Sample configuration for a resource allocation CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-allocatable (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods (2)
  kubeletConfig:
    systemReserved:
      cpu: 500m
      memory: 512Mi
    kubeReserved:
      cpu: 500m
      memory: 512Mi
```

4.9. ADVERTISING HIDDEN RESOURCES FOR NODES IN AN OPENSHIFT CONTAINER PLATFORM CLUSTER

Opaque integer resources allow cluster operators to provide new node-level resources that would be otherwise unknown to the system. Users can consume these resources in pod specifications, similar to CPU and memory. The scheduler performs resource accounting so that no more than the available amount is simultaneously allocated to pods.



NOTE

Opaque integer resources are Alpha currently, and only resource accounting is implemented. There is no resource quota or limit range support for these resources, and they have no impact on QoS.

4.9.1. Understanding opaque resources

Opaque integer resources are called *opaque* because OpenShift Container Platform does not know what the resource is, but will schedule a pod on a node only if enough of that resource is available. They are called *integer resources* because they must be available, or *advertised*, in integer amounts. The API server restricts quantities of these resources to whole numbers. Examples of *valid* quantities are **3**, **3000m**, and **3Ki**.

Opaque integer resources can be used to allocate:

- Last-level cache (LLC)
- Graphics processing unit (GPU) devices
- Field-programmable gate array (FPGA) devices
- Slots for sharing bandwidth to a parallel file system.

For example, if a node has 800 GiB of a special kind of disk storage, you could create a name for the special storage, such as ***opaque-int-resource-special-storage***. You could advertise it in chunks of a certain size, such as 100 GiB. In that case, your node would advertise that it has eight resources of type **opaque-int-resource-special-storage**.

Opaque integer resource names must begin with the prefix **pod.alpha.kubernetes.io/opaque-int-resource-**.

4.9.2. Creating Opaque Integer Resources

There are two steps required to use opaque integer resources. First, the cluster operator must name and advertise a per-node opaque resource on one or more nodes. Second, application developer must request the opaque resource in pods.

To make opaque integer resources available:

Procedure

1. Allocate the resource and assign a name starting with **pod.alpha.kubernetes.io/opaque-int-resource-**
2. Advertise a new opaque integer resource by submitting a PATCH HTTP request to the API server that specifies the available quantity in the **status.capacity** for a node in the cluster. For example, the following HTTP request advertises five **foo** resources on the **openshift-node-1** node.

```
PATCH /api/v1/nodes/openshift-node-1/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: openshift-master:8080

[
  {
    "op": "add",
    "path": "/status/capacity/pod.alpha.kubernetes.io~1opaque-int-resource-foo",
    "value": "5"
  }
]
```

NOTE

The **~1** in the **path** is the encoding for the character /. The operation path value in the JSON-Patch is interpreted as a JSON-Pointer. For more details, refer to IETF RFC 6901, section 3 .

After this operation, the node **status.capacity** includes a new resource. The **status.allocatable** field is updated automatically with the new resource asynchronously.

NOTE

Since the scheduler uses the node **status.allocatable** value when evaluating pod fitness, there might be a short delay between patching the node capacity with a new resource and the first pod that requests the resource to be scheduled on that node.

4.9.3. Consuming Opaque Integer Resources

An application developer can consume the opaque resources by editing the pod configuration.

Procedure

Edit the pod configuration to include the name of the opaque resource as a key in the **spec.containers[].resources.requests** field.

For example:

The following pod requests two CPUs and one **foo** (an opaque resource).

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: myimage
    resources:
      requests:
        cpu: 2
        pod.alpha.kubernetes.io/opaque-int-resource-foo: 1
```

The pod will be scheduled only if all of the resource requests are satisfied (including CPU, memory, and any opaque resources). The pod will remain in the **PENDING** state while the resource request cannot be met by any node.

```
Conditions:
  Type    Status
  PodScheduled  False
...
Events:
  FirstSeen  LastSeen Count From    SubObjectPath Type    Reason     Message
  ---------  -------- ----- ----    ------------- --------  ------     -------
  14s      0s  6 default-scheduler  Warning    FailedScheduling  No nodes are available that match all of
the following predicates:: Insufficient pod.alpha.kubernetes.io/opaque-int-resource-foo (1).
```

4.10. VIEWING NODE AUDIT LOGS

Audit provides a security-relevant chronological set of records documenting the sequence of activities that have affected system by individual users, administrators, or other components of the system.

4.10.1. About the API Audit Log

Audit works at the API server level, logging all requests coming to the server. Each audit log contains two entries:

1. The request line containing:
    a. A Unique ID allowing to match the response line (see #2)
    b. The source IP of the request
    c. The HTTP method being invoked
    d. The original user invoking the operation

e. The impersonated user for the operation (**self** meaning himself)

f. The impersonated group for the operation (**lookup** meaning user's group)

g. The namespace of the request or <none>

h. The URI as requested

2. The response line containing:

a. The unique ID from #1

b. The response code

You can view logs for the master nodes for the OpenShift Container Platform API server or the Kubernetes API server.

Example output for the Kubelet API server:

```
ip-10-0-140-97.ec2.internal {"kind":"Event","apiVersion":"audit.k8s.io/v1beta1","metadata":
{"creationTimestamp":"2019-04-09T19:56:58Z"},"level":"Metadata","timestamp":"2019-04-
09T19:56:58Z","auditID":"6e96c88b-ab6f-44d2-b62e-
d1413efd676b","stage":"ResponseComplete","requestURI":"/api/v1/nodes/audit-2019-04-09T14-07-
27.129.log","verb":"get","user":{"username":"kube:admin","groups":["system:cluster-
admins","system:authenticated"],"extra":{"scopes.authorization.openshift.io":["user:full"]}},"sourceIPs":
["10.0.57.93"],"userAgent":"oc/v1.14.6+c4799753c (linux/amd64) kubernetes/ba88cb2","objectRef":
{"resource":"nodes","name":"audit-2019-04-09T14-07-
27.129.log","apiVersion":"v1"},"responseStatus":{"metadata":
{},"status":"Failure","reason":"NotFound","code":404},"requestReceivedTimestamp":"2019-04-
09T19:56:58.982157Z","stageTimestamp":"2019-04-09T19:56:58.985300Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by
ClusterRoleBinding \"cluster-admins\" of ClusterRole \"cluster-admin\" to Group \"system:cluster-
admins\""}}
```

4.10.2. Configuring the API Audit Log level

You can configure the audit feature to set log level, retention policy, and the type of events to log.

Procedure

1. Set the audit log level:

a. Get the name of the API server Custom Resource (CR):

```
$ oc get APIServer

NAME      AGE
cluster   18h
```

b. Edit the API server CR:

```
$ oc edit APIServer cluster
```

```
apiVersion: config.openshift.io/v1
kind: APIServer
```

```
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: 2019-05-09T18:56:37Z
  generation: 2
  name: cluster
  resourceVersion: "562502"
  selfLink: /apis/config.openshift.io/v1/apiservers/cluster
  uid: 2c4e980b-728c-11e9-967d-0a973adad40e
spec:
  logLevel: "Normal" (1)
  operatorloglevel: "TraceAll" (2)
```

(1) (1) Specify the log level for an overall component.

(2) (2) Specify the log level for the API server Operator.

You can set the log level for both settings to one of the following. The setting can be different for each setting:

- **Normal**. Normal is the default. Normal working log information, including helpful notices for auditing or common operations. Similar to **glog=2**.
- **Debug**. Debug is for troubleshooting problems. A greater quanitity of notices than **Normal**, but contain less information than **Trace**. Common operations might be logged. Similar to **glog=4**.
- **Trace**. Trace is for troubleshooting problems when **Debug** is not verbose enough. Logs every function call as part of a common operation, including tracing execution of a query. Similar to **glog=6**.
- **TraceAll**. TraceAll is troubleshoting at the level of API content/decoding. Contains complete body content. In production clusters, this setting causes performance degradation and results in a significant number of logs. Similar to ` glog=8`.

4.10.3. Viewing the API Audit Log

You can view the basic audit log.

Procedure

To view the basic audit log:

1. View the OpenShift Container Platform API server logs
   a. If necessary, get the node IP and name of the log you want to view:

```
$ oc --insecure-skip-tls-verify adm node-logs --role=master --path=openshift-apiserver/

ip-10-0-140-97.ec2.internal audit-2019-04-09T00-12-19.834.log
ip-10-0-140-97.ec2.internal audit-2019-04-09T11-13-00.469.log
ip-10-0-140-97.ec2.internal audit.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T00-11-49.835.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T11-08-30.469.log
ip-10-0-153-35.ec2.internal audit.log
```

```
ip-10-0-170-165.ec2.internal audit-2019-04-09T00-13-00.128.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T11-10-04.082.log
ip-10-0-170-165.ec2.internal audit.log
```

b. View the OpenShift Container Platform API server log for a specific master node and timestamp or view all the logs for that master:

```
$ oc adm node-logs <node-ip> <log-name> --path=openshift-apiserver/<log-name>
```

For example:

```
$ oc adm node-logs ip-10-0-140-97.ec2.internal audit-2019-04-08T13-09-01.227.log --
path=openshift-apiserver/audit-2019-04-08T13-09-01.227.log
$ oc adm node-logs ip-10-0-140-97.ec2.internal audit.log --path=openshift-
apiserver/audit.log
```

The output appears similar to the following:

```
ip-10-0-140-97.ec2.internal
{"kind":"Event","apiVersion":"audit.k8s.io/v1beta1","metadata":
{"creationTimestamp":"2019-04-09T18:52:03Z"},"level":"Metadata","timestamp":"2019-
04-09T18:52:03Z","auditID":"9708b50d-8956-4c87-b9eb-
a53ba054c13d","stage":"ResponseComplete","requestURI":"/","verb":"get","user":
{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":
["10.128.0.1"],"userAgent":"Go-http-client/2.0","responseStatus":{"metadata":
{},"code":200},"requestReceivedTimestamp":"2019-04-
09T18:52:03.914638Z","stageTimestamp":"2019-04-
09T18:52:03.915080Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by
ClusterRoleBinding \"cluster-status-binding\" of ClusterRole \"cluster-status\" to Group
\"system:unauthenticated\""}}
```

2. View the Kubernetes API server logs:

a. If necessary, get the node IP and name of the log you want to view:

```
$ oc --insecure-skip-tls-verify adm node-logs --role=master --path=kube-apiserver/

ip-10-0-140-97.ec2.internal audit-2019-04-09T14-07-27.129.log
ip-10-0-140-97.ec2.internal audit-2019-04-09T19-18-32.542.log
ip-10-0-140-97.ec2.internal audit.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T19-24-22.620.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T19-51-30.905.log
ip-10-0-153-35.ec2.internal audit.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T18-37-07.511.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T19-21-14.371.log
ip-10-0-170-165.ec2.internal audit.log
```

b. View the Kubernetes API server log for a specific master node and timestamp or view all the logs for that master:

```
$ oc adm node-logs <node-ip> <log-name> --path=kube-apiserver/<log-name>
```

For example:

```
$ oc adm node-logs ip-10-0-140-97.ec2.internal audit-2019-04-09T14-07-27.129.log --
path=kube-apiserver/audit-2019-04-09T14-07-27.129.log
$ oc adm node-logs ip-10-0-170-165.ec2.internal audit.log --path=kube-
apiserver/audit.log
```

The output appears similar to the following:

```
ip-10-0-140-97.ec2.internal
{"kind":"Event","apiVersion":"audit.k8s.io/v1beta1","metadata":
{"creationTimestamp":"2019-04-09T19:56:58Z"},"level":"Metadata","timestamp":"2019-
04-09T19:56:58Z","auditID":"6e96c88b-ab6f-44d2-b62e-
d1413efd676b","stage":"ResponseComplete","requestURI":"/api/v1/nodes/audit-2019-04-
09T14-07-27.129.log","verb":"get","user":{"username":"kube:admin","groups":
["system:cluster-admins","system:authenticated"],"extra":
{"scopes.authorization.openshift.io":["user:full"]}},"sourceIPs":
["10.0.57.93"],"userAgent":"oc/v1.14.6+c4799753c (linux/amd64)
kubernetes/ba88cb2","objectRef":{"resource":"nodes","name":"audit-2019-04-09T14-07-
27.129.log","apiVersion":"v1"},"responseStatus":{"metadata":
{},"status":"Failure","reason":"NotFound","code":404},"requestReceivedTimestamp":"2019-
04-09T19:56:58.982157Z","stageTimestamp":"2019-04-
09T19:56:58.985300Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by
ClusterRoleBinding \"cluster-admins\" of ClusterRole \"cluster-admin\" to Group
\"system:cluster-admins\""}}
```

CHAPTER 5. WORKING WITH CONTAINERS

5.1. UNDERSTANDING CONTAINERS

The basic units of OpenShift Container Platform applications are called *containers*. Linux container technologies are lightweight mechanisms for isolating running processes so that they are limited to interacting with only their designated resources.

Many application instances can be running in containers on a single host without visibility into each others' processes, files, network, and so on. Typically, each container provides a single service (often called a "micro-service"), such as a web server or a database, though containers can be used for arbitrary workloads.

The Linux kernel has been incorporating capabilities for container technologies for years. OpenShift Container Platform and Kubernetes add the ability to orchestrate containers across multi-host installations.

5.2. USING INIT CONTAINERS TO PERFORM TASKS BEFORE A POD IS DEPLOYED

OpenShift Container Platform provides *Init Containers*, which are specialized containers that run before application containers and can contain utilities or setup scripts not present in an app image.

5.2.1. Understanding Init Containers

You can use an Init Container resource to perform tasks before the rest of a pod is deployed.

A pod can have Init Containers in addition to application containers. Init containers allow you to reorganize setup scripts and binding code.

An Init Container can:

- Contain and run utilities that are not desirable to include in the app Container image for security reasons.
- Contain utilities or custom code for setup that is not present in an app image. For example, there is no requirement to make an image FROM another image just to use a tool like sed, awk, python, or dig during setup.
- Use Linux namespaces so that they have different filesystem views from app containers, such as access to Secrets that application containers are not able to access.

Each Init Container must complete successfully before the next one is started. So, Init Containers provide an easy way to block or delay the startup of app containers until some set of preconditions are met.

For example, the following are some ways you can use Init Containers:

- Wait for a service to be created with a shell command like:

```
for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; done; exit 1
```

- Register this Pod with a remote server from the downward API with a command like:

```
$ curl -X POST
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d
‘instance=$()&ip=$()’
```

- Wait for some time before starting the app Container with a command like **sleep 60**.
- Clone a git repository into a volume.
- Place values into a configuration file and run a template tool to dynamically generate a configuration file for the main app Container. For example, place the POD_IP value in a configuration and generate the main app configuration file using Jinja.

See the Kubernetes documentation for more information.

5.2.2. Creating Init Containers

The following example outlines a simple Pod which has two Init Containers. The first waits for **myservice** and the second waits for **mydb**. Once both containers complete, the Pod begins.

Procedure

1. Create a YAML file for the Init Container:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2;
done;']
  - name: init-mydb
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

2. Create a YAML file for the **myservice** service.

```
kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

3. Create a YAML file for the **mydb** service.

```
kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

4. Run the following command to create the **myapp-pod**:

```
$ oc create -f myapp.yaml

pod/myapp-pod created
```

5. View the status of the pod:

```
$ oc get pods
NAME                    READY    STATUS           RESTARTS   AGE
myapp-pod                 0/1      Init:0/2          0         5s
```

Note that the pod status indicates it is waiting

6. Run the following commands to create the services:

```
$ oc create -f mydb.yaml
$ oc create -f myservice.yaml
```

7. View the status of the pod:

```
$ oc get pods
NAME                    READY    STATUS           RESTARTS   AGE
myapp-pod                 1/1      Running          0         2m
```

5.3. USING VOLUMES TO PERSIST CONTAINER DATA

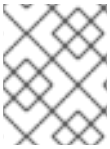
Files in a container are ephemeral. As such, when a container crashes or stops, the data is lost. You can use *volumes* to persist the data used by the containers in a pod. A volume is directory, accessible to the Containers in a Pod, where data is stored for the life of the pod.

5.3.1. Understanding volumes

Volumes are mounted file systems available to pods and their containers which may be backed by a number of host-local or network attached storage endpoints. Containers are not persistent by default; on restart, their contents are cleared.

To ensure that the file system on the volume contains no errors and, if errors are present, to repair them when possible, OpenShift Container Platform invokes the **fsck** utility prior to the **mount** utility. This occurs when either adding a volume or updating an existing volume.

The simplest volume type is **emptyDir**, which is a temporary directory on a single machine. Administrators may also allow you to request a persistent volume that is automatically attached to your pods.



NOTE

emptyDir volume storage may be restricted by a quota based on the pod's FSGroup, if the FSGroup parameter is enabled by your cluster administrator.

5.3.2. Working with volumes using the OpenShift Container Platform CLI

You can use the CLI command **oc set volume** to add and remove volumes and volume mounts for any object that has a pod template like replication controllers or DeploymentConfigs. You can also list volumes in pods or any object that has a pod template.

The **oc set volume** command uses the following general syntax:

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

Object selection

Specify one of the following for **object_seletion** in the **oc set volume** command:

Table 5.1. Object Selection

Syntax	Description	Example
<object_type> <name>	Selects ***<name>*** of type ***<object_type>***.	**deploymentConfig registry**
<object_type>/<name>	Selects ***<name>*** of type ***<object_type>***.	**deploymentConfig/registry**
<object_type>*--selector=*<object_label_selector>	Selects resources of type ***<object_type>*** that matched the given label selector.	**deploymentConfig--selector="name=registry"**
***<object_type>* --all**	Selects all resources of type ***<object_type>***.	**deploymentConfig --all**
-f or **--filename=*<file_name>***	File name, directory, or URL to file to use to edit the resource.	**-f registry-deployment-config.json**

Operation

Specify **--add**, **--remove**, or **--list** for **operation** in the **oc set volume** command.

Mandatory parameters

Any ***<mandatory_parameters>*** are specific to the selected operation and are discussed in later sections.

Options

Any ***<options>*** are specific to the selected operation and are discussed in later sections.

5.3.3. Listing volumes and volume mounts in a pod

You can list volumes and volume mounts in pods or pod templates:

Procedure

To list volumes:

```
$ oc set volume <object_type>/<name> --list [options]
```

List volume supported options:

Option	Description	Default
--name	Name of the volume.	
-c, --containers	Select containers by name. It can also take wildcard **'*'** that matches any character.	**'*'**

For example:

- To list all volumes for pod **p1**:

  ```
  $ oc set volume pod/p1 --list
  ```

- To list volume **v1** defined on all DeploymentConfigs:

  ```
  $ oc set volume dc --all --name=v1
  ```

5.3.4. Adding volumes to a pod

You can add volumes and volume mounts to a pod.

Procedure

To add a volume, a volume mount, or both to pod templates:

```
$ oc set volume <object_type>/<name> --add [options]
```

Table 5.2. Supported Options for Adding Volumes

Option	Description	Default
--name	Name of the volume.	Automatically generated, if not specified.

Option	Description	Default
-t, --type	Name of the volume source. Supported values: **emptyDir**, **hostPath**, **secret**, **configmap**, **persistentVolumeClaim** or **projected**.	**emptyDir**
-c, --containers	Select containers by name. It can also take wildcard **'*'** that matches any character.	**'*'**
-m, --mount-path	Mount path inside the selected containers.	
--path	Host path. Mandatory parameter for **--type=hostPath**.	
--secret-name	Name of the secret. Mandatory parameter for **--type=secret**.	
--configmap-name	Name of the configmap. Mandatory parameter for **--type=configmap**.	
--claim-name	Name of the persistent volume claim. Mandatory parameter for **--type=persistentVolumeClaim**.	
--source	Details of volume source as a JSON string. Recommended if the desired volume source is not supported by **--type**.	
-o, --output	Display the modified objects instead of updating them on the server. Supported values: **json**, **yaml**.	
--output-version	Output the modified objects with the given version.	**api-version**

For example:

- To add a new volume source **emptyDir** to DeploymentConfig **registry**:

```
$ oc set volume dc/registry --add
```

- To add volume **v1** with secret **$ecret** for replication controller **r1** and mount inside the containers at */data*:

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='$ecret' --mount-
path=/data
```

- To add existing persistent volume **v1** with claim name **pvc1** to deployment configuration ***dc.json*** on disk, mount the volume on container **c1** at ***/data***, and update the DeploymentConfig on the server:

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
  --claim-name=pvc1 --mount-path=/data --containers=c1
```

- To add a volume **v1** based on Git repository **https://github.com/namespace1/project1** with revision **5125c45f9f563** for all replication controllers:

```
$ oc set volume rc --all --add --name=v1 \
  --source='{"gitRepo": {
              "repository": "https://github.com/namespace1/project1",
              "revision": "5125c45f9f563"
            }}'
```

5.3.5. Updating volumes and volume mounts in a pod

You can modify the volumes and volume mounts in a pod.

Procedure

Updating existing volumes using the **--overwrite** option:

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

For example:

- To replace existing volume **v1** for replication controller **r1** with existing persistent volume claim **pvc1**:

```
$ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --claim-
name=pvc1
```

- To change DeploymentConfig **d1** mount point to ***/opt*** for volume **v1**:

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

5.3.6. Removing volumes and volume mounts from a pod

You can remove a volume or volume mount from a pod.

Procedure

To remove a volume from pod templates:

```
$ oc set volume <object_type>/<name> --remove [options]
```

Table 5.3. Supported Options for Removing Volumes

Option	Description	Default
--name	Name of the volume.	
-c, --containers	Select containers by name. It can also take wildcard **'*'** that matches any character.	**'*'**
--confirm	Indicate that you want to remove multiple volumes at once.	
-o, --output	Display the modified objects instead of updating them on the server. Supported values: **json**, **yaml**.	
--output-version	Output the modified objects with the given version.	**api-version**

For example:

- To remove a volume **v1** from DeploymentConfig **d1**:

```
$ oc set volume dc/d1 --remove --name=v1
```

- To unmount volume **v1** from container **c1** for DeploymentConfig **d1** and remove the volume **v1** if it is not referenced by any containers on **d1**:

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- To remove all volumes for replication controller **r1**:

```
$ oc set volume rc/r1 --remove --confirm
```

5.3.7. Configuring volumes for multiple uses in a pod

You can configure a volume to allows you to share one volume for multiple uses in a single pod using the **volumeMounts.subPath** property to specify a **subPath** inside a volume instead of the volume's root.

Procedure

1. View the list of files in the volume, run the **oc rsh** command:

```
$ oc rsh <pod>
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. Specify the **subPath**:

Example subPath Usage

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
        subPath: mysql (1)
    - name: php
      image: php
      volumeMounts:
      - mountPath: /var/www/html
        name: site-data
        subPath: html (2)
    volumes:
    - name: site-data
      persistentVolumeClaim:
        claimName: my-site-data
```

1. Databases are stored in the **mysql** folder.
2. HTML content is stored in the **html** folder.

5.4. MAPPING VOLUMES USING PROJECTED VOLUMES

A *projected volume* maps several existing volume sources into the same directory.

The following types of volume sources can be projected:

- Secrets
- Config Maps
- Downward API

NOTE

All sources are required to be in the same namespace as the pod.

5.4.1. Understanding projected volumes

Projected volumes can map any combination of these volume sources into a single directory, allowing the user to:

- automatically populate a single volume with the keys from multiple secrets, configmaps, and with downward API information, so that I can synthesize a single directory with various sources of information;

- populate a single volume with the keys from multiple secrets, configmaps, and with downward API information, explicitly specifying paths for each item, so that I can have full control over the contents of that volume.

The following general scenarios show how you can use projected volumes.

ConfigMap, Secrets, Downward API.

Projected volumes allow you to deploy containers with configuration data that includes passwords. An application using these resources could be deploying OpenStack on Kubernetes. The configuration data might have to be assembled differently depending on if the services are going to be used for production or for testing. If a pod is labeled with production or testing, the downward API selector **metadata.labels** can be used to produce the correct OpenStack configs.

ConfigMap + Secrets.

Projected volumes allow you to deploy containers involving configuration data and passwords. For example, you might execute a configmap with some sensitive encrypted tasks that are decrypted using a vault password file.

ConfigMap + Downward API.

Projected volumes allow you to generate a config including the pod name (available via the **metadata.name** selector). This application can then pass the pod name along with requests in order to easily determine the source without using IP tracking.

Secrets + Downward API.

Projected volumes allow you to use a secret as a public key to encrypt the namespace of the pod (available via the **metadata.namespace** selector). This example allows the operator to use the application to deliver the namespace information securely without using an encrypted transport.

5.4.1.1. Example Pod Specifications

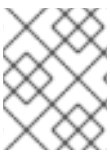
The following are examples of pod specifications for creating projected volumes.

Pod with a secret, a downward API, and a configmap

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts: 1
    - name: all-in-one
      mountPath: "/projected-volume" 2
      readOnly: true 3
  volumes: 4
  - name: all-in-one 5
    projected:
      defaultMode: 0400 6
      sources:
      - secret:
          name: mysecret 7
          items:
            - key: username
```

```
          path: my-group/my-username 8
    - downwardAPI: 9
        items:
          - path: "labels"
            fieldRef:
              fieldPath: metadata.labels
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: container-test
              resource: limits.cpu
    - configMap: 10
        name: myconfigmap
        items:
          - key: config
            path: my-group/my-config
            mode: 0777 11
```

1. Add a **volumeMounts** section for each container that needs the secret.
2. Specify a path to an unused directory where the secret will appear.
3. Set **readOnly** to **true**.
4. Add a **volumes** block to list each projected volume source.
5. Specify any name for the volume.
6. Set the execute permission on the files.
7. Add a secret. Enter the name of the secret object. Each secret you want to use must be listed.
8. Specify the path to the secrets file under the **mountPath**. Here, the secrets file is in */projected-volume/my-group/my-config*.
9. Add a Downward API source.
10. Add a ConfigMap source.
11. Set the mode for the specific projection



NOTE

If there are multiple containers in the pod, each container needs a **volumeMounts** section, but only one **volumes** section is needed.

Pod with multiple secrets with a non-default permission mode set

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
```

```
  image: busybox
  volumeMounts:
  - name: all-in-one
    mountPath: "/projected-volume"
    readOnly: true
volumes:
- name: all-in-one
  projected:
    defaultMode: 0755
    sources:
    - secret:
        name: mysecret
        items:
          - key: username
            path: my-group/my-username
    - secret:
        name: mysecret2
        items:
          - key: password
            path: my-group/my-password
            mode: 511
```

NOTE

The **defaultMode** can only be specified at the projected level and not for each volume source. However, as illustrated above, you can explicitly set the **mode** for each individual projection.

5.4.1.2. Pathing Considerations

Collisions Between Keys when Configured Paths are Identical

If you configure any keys with the same path, the pod spec will not be accepted as valid. In the following example, the specified path for **mysecret** and **myconfigmap** are the same:

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/data
```

```
- configMap:
    name: myconfigmap
    items:
      - key: config
        path: my-group/data
```

Consider the following situations related to the volume file paths.

Collisions Between Keys without Configured Paths

The only run-time validation that can occur is when all the paths are known at pod creation, similar to the above scenario. Otherwise, when a conflict occurs the most recent specified resource will overwrite anything preceding it (this is true for resources that are updated after pod creation as well).

Collisions when One Path is Explicit and the Other is Automatically Projected

In the event that there is a collision due to a user specified path matching data that is automatically projected, the latter resource will overwrite anything preceding it as before

5.4.2. Configuring a Projected Volume for a Pod

When creating projected volumes, consider the volume file path situations described in *Understanding projected vol`umes*.

The following example shows how to use a projected volume to mount an existing Secret volume source. The steps can be used to create a user name and password Secrets from local files. You then create a pod that runs one container, using a projected volume to mount the Secrets into the same shared directory.

Procedure

To use a projected volume to mount an existing Secret volume source.

1. Create files containing the secrets, entering the following, replacing the password and user information as appropriate:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
```

The **user** and **pass** values can be any valid string that is **base64** encoded. The examples used here are base64 encoded values **user: admin**, **pass:1f2d1e2e67df**.

```
$ echo -n "admin" | base64
YWRtaW4=
$ echo -n "1f2d1e2e67df" | base64
MWYyZDFlMmU2N2Rm
```

2. Use the following command to create the secrets:

```
$ oc create -f <secrets-filename>
```

For example:

```
$ oc create -f secret.yaml
secret "mysecret" created
```

3. You can check that the secret was created using the following commands:

```
$ oc get secret <secret-name>
$ oc get secret <secret-name> -o yaml
```

For example:

```
$ oc get secret mysecret
NAME       TYPE      DATA      AGE
mysecret   Opaque    2         17h
```

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
  name: mysecret
  namespace: default
  resourceVersion: "2107"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque
```

4. Create a pod configuration file similar to the following that includes a **volumes** section:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
```

```
projected:
  sources:
  - secret:      1
      name: user
  - secret:      2
      name: pass
```

1 2 The name of the secret you created.

5. Create the pod from the configuration file:

```
$ oc create -f <your_yaml_file>.yaml
```

For example:

```
$ oc create -f secret-pod.yaml
pod "test-projected-volume" created
```

6. Verify that the pod container is running, and then watch for changes to the Pod:

```
$ oc get pod <name>
```

The output should appear similar to the following:

```
$ oc get pod test-projected-volume
NAME                    READY     STATUS    RESTARTS   AGE
test-projected-volume   1/1       Running   0          14s
```

7. In another terminal, use the **oc exec** command to open a shell to the running container:

```
$ oc exec -it <pod> <command>
```

For example:

```
$ oc exec -it test-projected-volume -- /bin/sh
```

8. In your shell, verify that the **projected-volumes** directory contains your projected sources:

```
/ # ls
bin              home             root             tmp
dev              proc             run              usr
etc              projected-volume  sys              var
```

5.5. ALLOWING CONTAINERS TO CONSUME API OBJECTS

The *Downward API* is a mechanism that allows containers to consume information about API objects without coupling to OpenShift Container Platform. Such information includes the pod's name, namespace, and resource values. Containers can consume information from the downward API using environment variables or a volume plug-in.

5.5.1. Expose Pod information to Containers using the Downward API

The Downward API contains such information as the pod's name, project, and resource values. Containers can consume information from the downward API using environment variables or a volume plug-in.

Fields within the pod are selected using the **FieldRef** API type. **FieldRef** has two fields:

Field	Description
fieldPath	The path of the field to select, relative to the pod.
apiVersion	The API version to interpret the **fieldPath** selector within.

Currently, the valid selectors in the v1 API include:

Selector	Description
metadata.name	The pod's name. This is supported in both environment variables and volumes.
metadata.namespace	The pod's namespace.This is supported in both environment variables and volumes.
metadata.labels	The pod's labels. This is only supported in volumes and not in environment variables.
metadata.annotations	The pod's annotations. This is only supported in volumes and not in environment variables.
status.podIP	The pod's IP. This is only supported in environment variables and not volumes.

The **apiVersion** field, if not specified, defaults to the API version of the enclosing pod template.

5.5.2. Understanding how to consume container values using the downward API

You containers can consume API values using environment variables or a volume plug-in. Depending on the method you choose, containers can consume:

- Pod name
- Pod project/namespace
- Pod annotations
- Pod labels

Annotations and labels are available using only a volume plug-in.

5.5.2.1. Consuming container values using environment variables

When using a container's environment variables, use the **EnvVar** type's **valueFrom** field (of type **EnvVarSource**) to specify that the variable's value should come from a **FieldRef** source instead of the literal value specified by the **value** field.

Only constant attributes of the pod can be consumed this way, as environment variables cannot be updated once a process is started in a way that allows the process to be notified that the value of a variable has changed. The fields supported using environment variables are:

- Pod name
- Pod project/namespace

Procedure

To use environment variables

1. Create a ***pod.yaml*** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_POD_NAME** and **MY_POD_NAMESPACE** values:

```
$ oc logs -p dapi-env-test-pod
```

5.5.2.2. Consuming container values using a volume plug-in

You containers can consume API values using a volume plug-in.

Containers can consume:

- Pod name
- Pod project/namespace

- Pod annotations
- Pod labels

Procedure

To use the volume plug-in:

1. Create a ***volume-pod.yaml*** file:

```
kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  containers:
    - name: volume-test-container
      image: gcr.io/google_containers/busybox
      command: ["sh", "-c", "cat /tmp/etc/pod_labels /tmp/etc/pod_annotations"]
      volumeMounts:
        - name: podinfo
          mountPath: /tmp/etc
          readOnly: false
  volumes:
  - name: podinfo
    downwardAPI:
      defaultMode: 420
      items:
      - fieldRef:
          fieldPath: metadata.name
        path: pod_name
      - fieldRef:
          fieldPath: metadata.namespace
        path: pod_namespace
      - fieldRef:
          fieldPath: metadata.labels
        path: pod_labels
      - fieldRef:
          fieldPath: metadata.annotations
        path: pod_annotations
  restartPolicy: Never
```

2. Create the pod from the ***volume-pod.yaml*** file:

```
$ oc create -f volume-pod.yaml
```

3. Check the container's logs and verify the presence of the configured fields:

```
$ oc logs -p dapi-volume-test-pod
```

```
cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api
```

5.5.3. Understanding how to consume container resources using the downward API

When creating pods, you can use the downward API to inject information about computing resource requests and limits so that image and application authors can correctly create an image for specific environments.

You can do this using environment variable or a volume plug-in.

5.5.3.1. Consuming container resources using environment variables

When creating pods, you can use the downward API to inject information about computing resource requests and limits using environment variables.

Procedure

To use environment variables:

1. When creating a pod configuration, specify environment variables that correspond to the contents of the **resources** field in the **spec.container** field:

```
....
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox:1.24
      command: [ "/bin/sh", "-c", "env" ]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      env:
        - name: MY_CPU_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.cpu
        - name: MY_CPU_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.cpu
        - name: MY_MEM_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.memory
        - name: MY_MEM_LIMIT
          valueFrom:
```

```
            resourceFieldRef:
              resource: limits.memory
....
```

If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

2. Create the pod from the ***pod.yaml*** file:

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

5.5.3.2. Consuming container resources using a volume plug-in

When creating pods, you can use the downward API to inject information about computing resource requests and limits using a volume plug-in.

Procedure

To use the Volume Plug-in:

1. When creating a pod configuration, use the **spec.volumes.downwardAPI.items** field to describe the desired resources that correspond to the **spec.resources** field:

```
....
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat /etc/cpu_limit;
fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e /etc/mem_limit ]]; then cat
/etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat /etc/mem_request; fi; sleep 5; done"]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        items:
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.cpu
          - path: "cpu_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.cpu
          - path: "mem_limit"
            resourceFieldRef:
```

```
            containerName: client-container
            resource: limits.memory
          - path: "mem_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.memory
....
```

If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

2. Create the pod from the ***volume-pod.yaml*** file:

```
$ oc create -f volume-pod.yaml
```

5.5.4. Consuming secrets using the downward API

When creating pods, you can use the downward API to inject Secrets so image and application authors can create an image for specific environments.

Procedure

1. Create a ***secret.yaml*** file:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: cGFzc3dvcmQ=
  username: ZGV2ZWxvcGVy
type: kubernetes.io/basic-auth
```

2. Create a **Secret** from the secret.yaml file:

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

3. Create a ***pod.yaml*** file that references the **username** field from the above **Secret**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_SECRET_USERNAME
          valueFrom:
            secretKeyRef:
```

```
            name: mysecret
            key: username
  restartPolicy: Never
```

4. Create the pod from the ***pod.yaml*** file:

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

5. Check the container's logs for the **MY_SECRET_USERNAME** value:

```
$ oc logs -p dapi-env-test-pod
```

5.5.5. Consuming configuration maps using the downward API

When creating pods, you can use the downward API to inject configuration map values so image and application authors can create an image for specific environments.

Procedure

1. Create a ***configmap.yaml*** file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

2. Create a **ConfigMap** from the ***configmap.yaml*** file:

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

3. Create a ***pod.yaml*** file that references the above **ConfigMap**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_CONFIGMAP_VALUE
          valueFrom:
            configMapKeyRef:
              name: myconfigmap
              key: mykey
  restartPolicy: Always
```

4. Create the pod from the ***pod.yaml*** file:

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

5. Check the container's logs for the **MY_CONFIGMAP_VALUE** value:

```
$ oc logs -p dapi-env-test-pod
```

5.5.6. Referencing environment variables

When creating pods, you can reference the value of a previously defined environment variable by using the **$()** syntax. If the environment variable reference can not be resolved, the value will be left as the provided string.

Procedure

1. Create a ***pod.yaml*** file that references an existing **environment variable**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_EXISTING_ENV
          value: my_value
        - name: MY_ENV_VAR_REF_ENV
          value: $(MY_EXISTING_ENV)
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_ENV_VAR_REF_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

5.5.7. Escaping environment variable references

When creating a pod, you can escape an environment variable reference by using a double dollar sign. The value will then be set to a single dollar sign version of the provided value.

Procedure

1. Create a ***pod.yaml*** file that references an existing **environment variable**:

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

```
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$(SOME_OTHER_ENV)
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_NEW_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

5.6. COPYING FILES TO OR FROM AN OPENSHIFT CONTAINER PLATFORM CONTAINER

You can use the CLI to copy local files to or from a remote directory in a container using the **rsync** command.

5.6.1. Understanding how to copy files

The **oc rsync** command, or remote sync, is a useful tool for copying database archives to and from your pods for backup and restore purposes. You can also use **oc rsync** to copy source code changes into a running pod for development debugging, when the running pod supports hot reload of source files.

```
$ oc rsync <source> <destination> [-c <container>]
```

5.6.1.1. Requirements

Specifying the Copy Source

The source argument of the **oc rsync** command must point to either a local directory or a pod directory. Individual files are not supported.

When specifying a pod directory the directory name must be prefixed with the pod name:

```
<pod name>:<dir>
```

If the directory name ends in a path separator (/), only the contents of the directory are copied to the destination. Otherwise, the directory and its contents are copied to the destination.

Specifying the Copy Destination

The destination argument of the **oc rsync** command must point to a directory. If the directory does not exist, but **rsync** is used for copy, the directory is created for you.

Deleting Files at the Destination

The **--delete** flag may be used to delete any files in the remote directory that are not in the local directory.

Continuous Syncing on File Change

Using the **--watch** option causes the command to monitor the source path for any file system changes, and synchronizes changes when they occur. With this argument, the command runs forever.

Synchronization occurs after short quiet periods to ensure a rapidly changing file system does not result in continuous synchronization calls.

When using the **--watch** option, the behavior is effectively the same as manually invoking **oc rsync** repeatedly, including any arguments normally passed to **oc rsync**. Therefore, you can control the behavior via the same flags used with manual invocations of **oc rsync**, such as **--delete**.

5.6.2. Copying files to and from containers

Support for copying local files to or from a container is built into the CLI.

Prerequisites

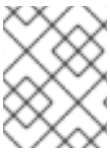
When working with **oc sync**, note the following:

rsync must be installed

The **oc rsync** command uses the local **rsync** tool if present on the client machine and the remote container.

If **rsync** is not found locally or in the remote container, a **tar** archive is created locally and sent to the container where the **tar** utility is used to extract the files. If **tar** is not available in the remote container, the copy will fail.

The **tar** copy method does not provide the same functionality as **oc rsync**. For example, **oc rsync** creates the destination directory if it does not exist and only sends files that are different between the source and the destination.



NOTE

In Windows, the **cwRsync** client should be installed and added to the PATH for use with the **oc rsync** command.

Procedure

- To copy a local directory to a pod directory:

```
oc rsync <local-dir> <pod-name>:/<remote-dir>
```

For example:

```
$ oc rsync /home/user/source devpod1234:/src

WARNING: cannot use rsync: rsync not available in container
status.txt
```

- To copy a pod directory to a local directory:

```
$ oc rsync devpod1234:/src /home/user/source
```

```
oc rsync devpod1234:/src/status.txt /home/user/
WARNING: cannot use rsync: rsync not available in container
status.txt
```

5.6.3. Using advanced Rsync features

The **oc rsync** command exposes fewer command line options than standard **rsync**. In the case that you wish to use a standard **rsync** command line option which is not available in **oc rsync** (for example the **--exclude-from=FILE** option), it may be possible to use standard **rsync** 's **--rsh** (**-e**) option or **RSYNC_RSH** environment variable as a workaround, as follows:

```
$ rsync --rsh='oc rsh' --exclude-from=FILE SRC POD:DEST
```

or:

```
$ export RSYNC_RSH='oc rsh'
$ rsync --exclude-from=FILE SRC POD:DEST
```

Both of the above examples configure standard **rsync** to use **oc rsh** as its remote shell program to enable it to connect to the remote pod, and are an alternative to running **oc rsync**.

5.7. EXECUTING REMOTE COMMANDS IN AN OPENSHIFT CONTAINER PLATFORM CONTAINER

You can use the CLI to execute remote commands in an OpenShift Container Platform container.

5.7.1. Executing remote commands in containers

Support for remote container command execution is built into the CLI.

Procedure

To run a command in a container:

```
$ oc exec <pod> [-c <container>] <command> [<arg_1> ... <arg_n>]
```

For example:

```
$ oc exec mypod date
Thu Apr  9 02:21:53 UTC 2015
```

IMPORTANT

For security purposes, the **oc exec** command does not work when accessing privileged containers except when the command is executed by a **cluster-admin** user.

5.7.2. Protocol for initiating a remote command from a client

Clients initiate the execution of a remote command in a container by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the project of the target pod.
- **<pod>** is the name of the target pod.
- **<container>** is the name of the target container.
- **<command>** is the desired command to be executed.

For example:

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

Additionally, the client can add parameters to the request to indicate if:

- the client should send input to the remote container's command (stdin).
- the client's terminal is a TTY.
- the remote container's command should send output from stdout to the client.
- the remote container's command should send output from stderr to the client.

After sending an **exec** request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses **SPDY**.

The client creates one stream each for stdin, stdout, and stderr. To distinguish among the streams, the client sets the **streamType** header on the stream to one of **stdin**, **stdout**, or **stderr**.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the remote command execution request.

5.8. USING PORT FORWARDING TO ACCESS APPLICATIONS IN A CONTAINER

OpenShift Container Platform supports port forwarding to pods.

5.8.1. Understanding port forwarding

You can use the CLI to forward one or more local ports to a pod. This allows you to listen on a given or random port locally, and have data forwarded to and from given ports in the pod.

Support for port forwarding is built into the CLI:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

The CLI listens on each local port specified by the user, forwarding via the protocol described below.

Ports may be specified using the following formats:

5000	The client listens on port 5000 locally and forwards to 5000 in the pod.
6000:5000	The client listens on port 6000 locally and forwards to 5000 in the pod.
:5000 or **0:5000**	The client selects a free local port and forwards to 5000 in the pod.

OpenShift Container Platform handles port-forward requests from clients. Upon receiving a request, OpenShift Container Platform upgrades the response and waits for the client to create port-forwarding streams. When OpenShift Container Platform receives a new stream, it copies data between the stream and the pod's port.

Architecturally, there are options for forwarding to a pod's port. The supported OpenShift Container Platform implementation invokes **nsenter** directly on the node host to enter the pod's network namespace, then invokes **socat** to copy data between the stream and the pod's port. However, a custom implementation could include running a *helper* pod that then runs **nsenter** and **socat**, so that those binaries are not required to be installed on the host.

5.8.2. Using port forwarding

You can use the CLI to port-forward one or more local ports to a pod.

Procedure

Use the following command to listen on the specified port in a pod:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

For example:

- Use the following command to listen on ports **5000** and **6000** locally and forward data to and from ports **5000** and **6000** in the pod:

```
$ oc port-forward <pod> 5000 6000

Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
Forwarding from 127.0.0.1:6000 -> 6000
Forwarding from [::1]:6000 -> 6000
```

- Use the following command to listen on port **8888** locally and forward to **5000** in the pod:

```
$ oc port-forward <pod> 8888:5000

Forwarding from 127.0.0.1:8888 -> 5000
Forwarding from [::1]:8888 -> 5000
```

- Use the following command to listen on a free port locally and forward to **5000** in the pod:

```
$ oc port-forward <pod> :5000

Forwarding from 127.0.0.1:42390 -> 5000
```

```
Forwarding from [::1]:42390 -> 5000
```

Or:

```
$ oc port-forward <pod> 0:5000
```

5.8.3. Protocol for initiating port forwarding from a client

Clients initiate port forwarding to a pod by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the namespace of the target pod.
- **<pod>** is the name of the target pod.

For example:

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

After sending a port forward request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses SPDY.

The client creates a stream with the **port** header containing the target port in the pod. All data written to the stream is delivered via the Kubelet to the target pod and port. Similarly, all data sent from the pod for that forwarded connection is delivered back to the same stream in the client.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the port forwarding request.

5.9. USING SYSCTLS IN CONTAINERS

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; you can manually enable other *unsafe* sysctls on the node to be available to the user.

5.9.1. About 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, run:

```
$ sudo sysctl -a
```

5.9.1.1. Namespaced versus node-level sysctls

A number of sysctls are *namespaced* in the Linux kernels. This means that you can set them 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.****

Additionally, most of the sysctls in the **net.*** group are known to be namespaced. Their namespace adoption differs based on the kernel version and distributor.

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, such as by modifying the ***/etc/sysctls.conf*** file, or by 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 taints and toleration feature to mark the nodes.

5.9.1.2. Safe versus unsafe sysctls

Sysctls are grouped into *safe* and *unsafe* sysctls.

For a sysctl to be considered safe, it must use proper namespacing and must be properly isolated between pods on the same node. This means that if you set a sysctl for one pod it must not:

- Influence any other pod on the node
- Harm the node's health
- Gain CPU or memory resources outside of the resource limits of a pod

OpenShift Container Platform supports, or whitelists, the following sysctls in the safe set:

- ***kernel.shm_rmid_forced***
- ***net.ipv4.ip_local_port_range***
- ***net.ipv4.tcp_syncookies***

All safe sysctls are enabled by default. You can use a sysctl in a pod by modifying the pod specification.

Any sysctl not whitelisted by OpenShift Container Platform is considered unsafe for OpenShift Container Platform. Note that being namespaced alone is not sufficient for the sysctl to be considered safe.

All unsafe sysctls are disabled by default, and the cluster administrator must manually enable them on a per-node basis. Pods with disabled unsafe sysctls are scheduled but do not launch.

```
$ oc get pod

NAME       READY   STATUS          RESTARTS   AGE
hello-pod  0/1     SysctlForbidden  0         14s
```

5.9.2. Setting sysctls for a pod

You can set sysctls on pods using the pod's **securityContext**. The **securityContext** applies to all containers in the same pod.

Safe sysctls are allowed by default. A pod with unsafe sysctls fails to launch on any node unless the cluster administrator explicitly enables unsafe sysctls for that node. As with node-level sysctls, use the taints and toleration feature or labels on nodes to schedule those pods onto the right nodes.

The following example uses the pod **securityContext** to set a safe sysctl **kernel.shm_rmid_forced** and two unsafe sysctls, **net.ipv4.route.min_pmtu** and **kernel.msgmax**. There is no distinction between *safe* and *unsafe* sysctls in the specification.

WARNING

To avoid destabilizing your operating system, modify sysctl parameters only after you understand their effects.

Procedure

To use safe and unsafe sysctls:

1. Modify the YAML file that defines the pod and add the **securityContext** spec, as shown in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
spec:
  securityContext:
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.ipv4.route.min_pmtu
      value: "552"
```

```
    - name: kernel.msgmax
      value: "65536"
  ...
```

2. Create the pod:

```
$ oc apply -f <file-name>.yaml
```

If the unsafe sysctls are not allowed for the node, the pod is scheduled, but does not deploy:

```
$ oc get pod

NAME       READY   STATUS            RESTARTS   AGE
hello-pod  0/1     SysctlForbidden   0          14s
```

5.9.3. Enabling unsafe sysctls

A cluster administrator can allow certain unsafe sysctls for very special situations such as high-performance or real-time application tuning.

If you want to use unsafe sysctls, a cluster administrator must enable them individually for a specific type of node. The sysctls must be namespaced.

WARNING

Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems, such as improper behavior of containers, resource shortage, or breaking a node.

Procedure

1. Add a label to the MachineConfigPool where the containers where containers with the unsafe sysctls will run:

```
$ oc edit machineconfigpool worker
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: sysctl 1
```

1 Add a **key: pair** label.

2. Create a KubeletConfig Custom Resource (CR):

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl 1
  kubeletConfig:
    allowedUnsafeSysctls: 2
      - "kernel.msg*"
      - "net.ipv4.route.min_pmtu"
```

1 Specify the label from the MachineConfigPool.

2 List the unsafe sysctls you want to allow.

3. Create the object:

```
$ oc apply -f set-sysctl-worker.yaml
```

A new MachineConfig named in the **99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet** format is created.

4. Wait for the cluster to reboot usng the **machineconfigpool** object **status** fields:
For example:

```
status:
  conditions:
    - lastTransitionTime: '2019-08-11T15:32:00Z'
      message: >-
        All nodes are updating to
        rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
      reason: ''
      status: 'True'
      type: Updating
```

A message similar to the following appears when the cluster is ready:

```
    - lastTransitionTime: '2019-08-11T16:00:00Z'
      message: >-
        All nodes are updated with
        rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
      reason: ''
      status: 'True'
      type: Updated
```

5. When the cluster is ready, check for the merged **KubeletConfig** in the new MachineConfig:

```
$ oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep
ownerReference -A7
```

```
        "ownerReferences": [
```

```
        {
          "apiVersion": "machineconfiguration.openshift.io/v1",
          "blockOwnerDeletion": true,
          "controller": true,
          "kind": "KubeletConfig",
          "name": "custom-kubelet",
          "uid": "3f64a766-bae8-11e9-abe8-0a1a2a4813f2"
```


You can now add unsafe sysctls to pods as needed.

5.10. MONITORING CONTAINER HEALTH

In software systems, components can become unhealthy due to transient issues such as temporary connectivity loss, configuration errors, or problems with external dependencies. OpenShift Container Platform applications have a number of options to detect and handle unhealthy containers.

5.10.1. Understanding health checks

A probe is a Kubernetes action that periodically performs diagnostics on a running container. Currently, two types of probes exist, each serving a different purpose.

Readiness Probe

A Readiness check determines if the container in which it is scheduled is ready to service requests. If the readiness probe fails a container, the endpoints controller ensures the container has its IP address removed from the endpoints of all services. A readiness probe can be used to signal to the endpoints controller that even though a container is running, it should not receive any traffic from a proxy.

For example, a Readiness check can control which Pods are used. When a Pod is not ready, it is removed.

Liveness Probe

A Liveness checks determines if the container in which it is scheduled is still running. If the liveness probe fails due to a condition such as a deadlock, the kubelet kills the container The container then responds based on its restart policy.

For example, a liveness probe on a node with a **restartPolicy** of **Always** or **OnFailure** kills and restarts the Container on the node.

Sample Liveness Check

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness-http
    image: k8s.gcr.io/liveness (1)
    args:
    - /server
    livenessProbe: (2)
      httpGet: (3)
```

```
    # host: my-host
    # scheme: HTTPS
    path: /healthz
    port: 8080
    httpHeaders:
    - name: X-Custom-Header
      value: Awesome
  initialDelaySeconds: 15 4
  timeoutSeconds: 1 5
 name: liveness 6
```

1. Specifies the image to use for the liveness probe.
2. Specifies the type of heath check.
3. Specifies the type of Liveness check:
   - HTTP Checks. Specify **httpGet**.
   - Container Execution Checks. Specify **exec**.
   - TCP Socket Check. Specify **tcpSocket**.
4. Specifies the number of seconds before performing the first probe after the container starts.
5. Specifies the number of seconds between probes.

Sample Liveness check output wth unhealthy container

```
$ oc describe pod pod1

....

FirstSeen LastSeen    Count   From            SubobjectPath          Type       Reason      Message
--------- --------    -----   ----            -------------          --------   ------      -------
37s       37s     1   {default-scheduler }                       Normal     Scheduled   Successfully assigned
liveness-exec to worker0
36s       36s     1   {kubelet worker0}   spec.containers{liveness}   Normal     Pulling     pulling image
"k8s.gcr.io/busybox"
36s       36s     1   {kubelet worker0}   spec.containers{liveness}   Normal     Pulled      Successfully
pulled image "k8s.gcr.io/busybox"
36s       36s     1   {kubelet worker0}   spec.containers{liveness}   Normal     Created     Created
container with docker id 86849c15382e; Security:[seccomp=unconfined]
36s       36s     1   {kubelet worker0}   spec.containers{liveness}   Normal     Started     Started
container with docker id 86849c15382e
2s        2s      1   {kubelet worker0}   spec.containers{liveness}   Warning    Unhealthy   Liveness
probe failed: cat: can't open '/tmp/healthy': No such file or directory
```

5.10.1.1. Understanding the types of health checks

Liveness checks and Readiness checks can be configured in three ways:

HTTP Checks

The kubelet uses a web hook to determine the healthiness of the container. The check is deemed successful if the HTTP response code is between 200 and 399.

A HTTP check is ideal for applications that return HTTP status codes when completely initialized.

Container Execution Checks

The kubelet executes a command inside the container. Exiting the check with status 0 is considered a success.

TCP Socket Checks

The kubelet attempts to open a socket to the container. The container is only considered healthy if the check can establish a connection. A TCP socket check is ideal for applications that do not start listening until initialization is complete.

5.10.2. Configuring health checks

To configure health checks, create a pod for each type of check you want.

Procedure

To create health checks:

1. Create a Liveness Container Execution Check:

   a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - args:
    image: k8s.gcr.io/liveness
    livenessProbe:
      exec: 1
        command: 2
        - cat
        - /tmp/health
      initialDelaySeconds: 15 3
...
```

   1. Specify a Liveness check and the type of Liveness check.
   2. Specify the commands to use in the container.
   3. Specify the number of seconds before performing the first probe after the container starts.

   b. Verify the state of the health check pod:

```
$ oc describe pod liveness-exec
```

```
Events:
  Type    Reason     Age   From                                Message
  ----    ------     ----  ----                                -------
  Normal  Scheduled  9s    default-scheduler                   Successfully assigned
openshift-logging/liveness-exec to ip-10-0-143-40.ec2.internal
  Normal  Pulling    2s    kubelet, ip-10-0-143-40.ec2.internal  pulling image
"k8s.gcr.io/liveness"
  Normal  Pulled     1s    kubelet, ip-10-0-143-40.ec2.internal  Successfully pulled image
"k8s.gcr.io/liveness"
  Normal  Created    1s    kubelet, ip-10-0-143-40.ec2.internal  Created container
  Normal  Started    1s    kubelet, ip-10-0-143-40.ec2.internal  Started container
```

NOTE

The **timeoutSeconds** parameter has no effect on the Readiness and Liveness probes for Container Execution Checks. You can implement a timeout inside the probe itself, as OpenShift Container Platform cannot time out on an exec call into the container. One way to implement a timeout in a probe is by using the **timeout** parameter to run your liveness or readiness probe:

```
spec:
  containers:
    livenessProbe:
      exec:
        command:
          - /bin/bash
          - '-c'
          - timeout 60 /opt/eap/bin/livenessProbe.sh 1
      timeoutSeconds: 1
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
```

1 Timeout value and path to the probe script.

c. Create the check:

```
$ oc create -f <file-name>.yaml
```

2. Create a Liveness TCP Socket Check:

a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-tcp
spec:
  containers:
  - name: contaier1 1
    image: k8s.gcr.io/liveness
```

```
    ports:
    - containerPort: 8080 (2)
    livenessProbe: (3)
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15 (4)
      timeoutSeconds: 1 (5)
```

(1) (2) Specify the container name and port for the check to connect to.

(3) Specify the Liveness heath check and the type of Liveness check.

(4) Specify the number of seconds before performing the first probe after the container starts.

(5) Specify the number of seconds between probes.

b. Create the check:

```
$ oc create -f <file-name>.yaml
```

3. Create an Readiness HTTP Check:

a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: readiness
  name: readiness-http
spec:
  containers:
  - args:
    image: k8s.gcr.io/readiness (1)
    readinessProbe: (2)
    httpGet:
    # host: my-host (3)
    # scheme: HTTPS (4)
      path: /healthz
      port: 8080
    initialDelaySeconds: 15 (5)
    timeoutSeconds: 1 (6)
```

(1) Specify the image to use for the liveness probe.

(2) Specify the Readiness heath check and the type of Readiness check.

(3) Specify a host IP address. When **host** is not defined, the **PodIP** is used.

(4) Specify **HTTP** or **HTTPS**. When **scheme** is not defined, the **HTTP** scheme is used.

(5) Specify the number of seconds before performing the first probe after the container starts.

6 Specify the number of seconds between probes.

b. Create the check:

```
$ oc create -f <file-name>.yaml
```

CHAPTER 6. WORKING WITH CLUSTERS

6.1. VIEWING SYSTEM EVENT INFORMATION IN AN OPENSHIFT CONTAINER PLATFORM CLUSTER

Events in OpenShift Container Platform are modeled based on events that happen to API objects in an OpenShift Container Platform cluster.

6.1.1. Understanding events

Events allow OpenShift Container Platform to record information about real-world events in a resource-agnostic manner. They also allow developers and administrators to consume information about system components in a unified way.

6.1.2. Viewing events using the CLI

You can get a list of events in a given project using the CLI.

Procedure

- To view events in a project use the following command:

```
$ oc get events [-n <project>] 1
```

1 The name of the project.

For example:

```
$ oc get events -n openshift-config

LAST SEEN   TYPE      REASON                  OBJECT                    MESSAGE
97m         Normal    Scheduled               pod/dapi-env-test-pod     Successfully assigned
openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
97m         Normal    Pulling                 pod/dapi-env-test-pod     pulling image
"gcr.io/google_containers/busybox"
97m         Normal    Pulled                  pod/dapi-env-test-pod     Successfully pulled image
"gcr.io/google_containers/busybox"
97m         Normal    Created                 pod/dapi-env-test-pod     Created container
9m5s        Warning   FailedCreatePodSandBox  pod/dapi-volume-test-pod  Failed create
pod sandbox: rpc error: code = Unknown desc = failed to create pod network sandbox
k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9e22
): Multus: Err adding pod to network "openshift-sdn": cannot set "openshift-sdn" ifname to
"eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or directory
8m31s       Normal    Scheduled               pod/dapi-volume-test-pod  Successfully assigned
openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal
```

- To view events in your project from the OpenShift Container Platform console.

  1. Launch the OpenShift Container Platform console.

  2. Click **Home → Events** and select your project.

3. Move to resource that you want to see events. For example: **Home → Projects →** <project-name> **→** <resource-name>.
   Many objects, such as pods and deployments, have their own **Events** tab as well, which shows events related to that object.

6.1.3. List of events

This section describes the events of OpenShift Container Platform.

Table 6.1. Configuration Events

Name	Description
FailedValidation	Failed pod configuration validation.

Table 6.2. Container Events

Name	Description
BackOff	Back-off restarting failed the container.
Created	Container created.
Failed	Pull/Create/Start failed.
Killing	Killing the container.
Started	Container started.
Preempting	Preempting other pods.
ExceededGracePeriod	Container runtime did not stop the pod within specified grace period.

Table 6.3. Health Events

Name	Description
Unhealthy	Container is unhealthy.

Table 6.4. Image Events

Name	Description
BackOff	Back off Ctr Start, image pull.
ErrImageNeverPull	The image's **NeverPull Policy** is violated.

Name	Description
Failed	Failed to pull the image.
InspectFailed	Failed to inspect the image.
Pulled	Successfully pulled the image or the container image is already present on the machine.
Pulling	Pulling the image.

Table 6.5. Image Manager Events

Name	Description
FreeDiskSpaceFailed	Free disk space failed.
InvalidDiskCapacity	Invalid disk capacity.

Table 6.6. Node Events

Name	Description
FailedMount	Volume mount failed.
HostNetworkNotSupported	Host network not supported.
HostPortConflict	Host/port conflict.
InsufficientFreeCPU	Insufficient free CPU.
InsufficientFreeMemory	Insufficient free memory.
KubeletSetupFailed	Kubelet setup failed.
NilShaper	Undefined shaper.
NodeNotReady	Node is not ready.
NodeNotSchedulable	Node is not schedulable.

Name	Description
NodeReady	Node is ready.
NodeSchedulable	Node is schedulable.
NodeSelectorMismatching	Node selector mismatch.
OutOfDisk	Out of disk.
Rebooted	Node rebooted.
Starting	Starting kubelet.
FailedAttachVolume	Failed to attach volume.
FailedDetachVolume	Failed to detach volume.
VolumeResizeFailed	Failed to expand/reduce volume.
VolumeResizeSuccessful	Successfully expanded/reduced volume.
FileSystemResizeFailed	Failed to expand/reduce file system.
FileSystemResizeSuccessful	Successfully expanded/reduced file system.
FailedUnMount	Failed to unmount volume.
FailedMapVolume	Failed to map a volume.
FailedUnmapDevice	Failed unmaped device.
AlreadyMountedVolume	Volume is already mounted.
SuccessfulDetachVolume	Volume is successfully detached.

Name	Description
SuccessfulMountVolume	Volume is successfully mounted.
SuccessfulUnMountVolume	Volume is successfully unmounted.
ContainerGCFailed	Container garbage collection failed.
ImageGCFailed	Image garbage collection failed.
FailedNodeAllocatableEnforcement	Failed to enforce System Reserved Cgroup limit.
NodeAllocatableEnforced	Enforced System Reserved Cgroup limit.
UnsupportedMountOption	Unsupported mount option.
SandboxChanged	Pod sandbox changed.
FailedCreatePodSandBox	Failed to create pod sandbox.
FailedPodSandBoxStatus	Failed pod sandbox status.

Table 6.7. Pod Worker Events

Name	Description
FailedSync	Pod sync failed.

Table 6.8. System Events

Name	Description
SystemOOM	There is an OOM (out of memory) situation on the cluster.

Table 6.9. Pod Events

Name	Description
FailedKillPod	Failed to stop a pod.
FailedCreatePodContainer	Failed to create a pod contianer.
Failed	Failed to make pod data directories.
NetworkNotReady	Network is not ready.
FailedCreate	Error creating: **<error-msg>**.
SuccessfulCreate	Created pod: **<pod-name>**.
FailedDelete	Error deleting: **<error-msg>**.
SuccessfulDelete	Deleted pod: **<pod-id>**.

Table 6.10. Horizontal Pod AutoScaler Events

Name	Description
SelectorRequired	Selector is required.
InvalidSelector	Could not convert selector into a corresponding internal selector object.
FailedGetObjectMetric	HPA was unable to compute the replica count.
InvalidMetricSourceType	Unknown metric source type.
ValidMetricFound	HPA was able to successfully calculate a replica count.
FailedConvertHPA	Failed to convert the given HPA.
FailedGetScale	HPA controller was unable to get the target's current scale.
SucceededGetScale	HPA controller was able to get the target's current scale.

Name	Description
FailedComputeMetricsReplicas	Failed to compute desired number of replicas based on listed metrics.
FailedRescale	New size: **<size>**; reason: **<msg>**; error: **<error-msg>**.
SuccessfulRescale	New size: **<size>**; reason: **<msg>**.
FailedUpdateStatus	Failed to update status.

Table 6.11. Network Events (openshift-sdn)

Name	Description
Starting	Starting OpenShift-SDN.
NetworkFailed	The pod's network interface has been lost and the pod will be stopped.

Table 6.12. Network Events (kube-proxy)

Name	Description
NeedPods	The service-port **<serviceName>:<port>** needs pods.

Table 6.13. Volume Events

Name	Description
FailedBinding	There are no persistent volumes available and no storage class is set.
VolumeMismatch	Volume size or class is different from what is requested in claim.
VolumeFailedRecycle	Error creating recycler pod.
VolumeRecycled	Occurs when volume is recycled.
RecyclerPod	Occurs when pod is recycled.
VolumeDelete	Occurs when volume is deleted.

Name	Description
VolumeFailedDelete	Error when deleting the volume.
ExternalProvisioning	Occurs when volume for the claim is provisioned either manually or via external software.
ProvisioningFailed	Failed to provision volume.
ProvisioningCleanupFailed	Error cleaning provisioned volume.
ProvisioningSucceeded	Occurs when the volume is provisioned successfully.
WaitForFirstConsumer	Delay binding until pod scheduling.

Table 6.14. Lifecycle hooks

Name	Description
FailedPostStartHook	Handler failed for pod start.
FailedPreStopHook	Handler failed for pre-stop.
UnfinishedPreStopHook	Pre-stop hook unfinished.

Table 6.15. Deployments

Name	Description
DeploymentCancellationFailed	Failed to cancel deployment.
DeploymentCancelled	Cancelled deployment.
DeploymentCreated	Created new replication controller.
IngressIPRangeFull	No available Ingress IP to allocate to service.

Table 6.16. Scheduler Events

Name	Description
FailedScheduling	Failed to schedule pod: **<pod-namespace>/<pod-name>**. This event is raised for multiple reasons, for example: **AssumePodVolumes** failed, Binding rejected etc.
Preempted	By **<preemptor-namespace>/<preemptor-name>** on node **<node-name>**.
Scheduled	Successfully assigned **<pod-name>** to **<node-name>**.

Table 6.17. DaemonSet Events

Name	Description
SelectingAll	This daemon set is selecting all pods. A non-empty selector is required.
FailedPlacement	Failed to place pod on **<node-name>**.
FailedDaemonPod	Found failed daemon pod **<pod-name>** on node **<node-name>**, will try to kill it.

Table 6.18. LoadBalancer Service Events

Name	Description
CreatingLoadBalancerFailed	Error creating load balancer.
DeletingLoadBalancer	Deleting load balancer.
EnsuringLoadBalancer	Ensuring load balancer.
EnsuredLoadBalancer	Ensured load balancer.
UnAvailableLoadBalancer	There are no available nodes for **LoadBalancer** service.
LoadBalancerSourceRanges	Lists the new **LoadBalancerSourceRanges**. For example, **<old-source-range>** → **<new-source-range>**.
LoadbalancerIP	Lists the new IP address. For example, **<old-ip>** → **<new-ip>**.
ExternalIP	Lists external IP address. For example, **Added: <external-ip>**.

Name	Description
UID	Lists the new UID. For example, **<old-service-uid> → <new-service-uid>**.
ExternalTrafficPolicy	Lists the new **ExternalTrafficPolicy**. For example,**<old-policy> → <new-ploicy>**.
HealthCheckNodePort	Lists the new **HealthCheckNodePort**. For example,**<old-node-port> → new-node-port>**.
UpdatedLoadBalancer	Updated load balancer with new hosts.
LoadBalancerUpdateFailed	Error updating load balancer with new hosts.
DeletingLoadBalancer	Deleting load balancer.
DeletingLoadBalancerFailed	Error deleting load balancer.
DeletedLoadBalancer	Deleted load balancer.

6.2. ESTIMATING THE NUMBER OF PODS YOUR OPENSHIFT CONTAINER PLATFORM NODES CAN HOLD

As a cluster administrator, you can use the cluster capacity tool to view the number of pods that can be scheduled to increase the current resources before they become exhausted, and to ensure any future pods can be scheduled. This capacity comes from an individual node host in a cluster, and includes CPU, memory, disk space, and others.

6.2.1. Understanding the OpenShift Container Platform cluster capacity tool

The cluster capacity tool simulates a sequence of scheduling decisions to determine how many instances of an input pod can be scheduled on the cluster before it is exhausted of resources to provide a more accurate estimation.

NOTE

The remaining allocatable capacity is a rough estimation, because it does not count all of the resources being distributed among nodes. It analyzes only the remaining resources and estimates the available capacity that is still consumable in terms of a number of instances of a pod with given requirements that can be scheduled in a cluster.

Also, pods might only have scheduling support on particular sets of nodes based on its selection and affinity criteria. As a result, the estimation of which remaining pods a cluster can schedule can be difficult.

You can run the cluster capacity analysis tool as a stand-alone utility from the command line, or as a job in a pod inside an OpenShift Container Platform cluster. Running it as job inside of a pod enables you to run it multiple times without intervention.

6.2.2. Running the cluster capacity tool on the command line

You can run the OpenShift Container Platform cluster capacity tool from the command line to estimate the number of pods that can be scheduled onto your cluster.

Prerequisites

- Download and install the **cluster-capacity** tool.
- Create a sample pod specification file, which the tool uses for estimating resource usage. The **podspec** specifies its resource requirements as **limits** or **requests**. The cluster capacity tool takes the pod's resource requirements into account for its estimation analysis.
  An example of the pod specification input is:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

Procedure

To run the tool on the command line:

1. Run the following command:

```
$ ./cluster-capacity --kubeconfig <path-to-kubeconfig> \ 1
    --podspec <path-to-pod-spec> 2
```

1. Specify the path to your Kubernetes configuration file.
2. Specify the path to the sample pod specification file

You can also add the **--verbose** option to output a detailed description of how many pods can be scheduled on each node in the cluster:

```
$ ./cluster-capacity --kubeconfig <path-to-kubeconfig> \
    --podspec <path-to-pod-spec> --verbose
```

2. View the output, which looks similar to the following:

```
small-pod pod requirements:
 - CPU: 150m
 - Memory: 100Mi

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the
following predicates:: Insufficient cpu (2).

Pod distribution among nodes:
small-pod
 - 192.168.124.214: 26 instance(s)
 - 192.168.124.120: 26 instance(s)
```

In the above example, the number of estimated pods that can be scheduled onto the cluster is 52.

6.2.3. Running the cluster capacity tool as a job inside a pod

Running the cluster capacity tool as a job inside of a pod has the advantage of being able to be run multiple times without needing user intervention. Running the cluster capacity tool as a job involves using a **ConfigMap**.

Prerequisites

Download and install the **cluster-capacity** tool.

Procedure

To run the cluster capacity tool:

1. Create the cluster role:

```
$ cat << EOF| oc create -f -
kind: ClusterRole
apiVersion: v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services"]
  verbs: ["get", "watch", "list"]
EOF
```

2. Create the service account:

```
$ oc create sa cluster-capacity-sa
```

3. Add the role to the service account:

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

4. Define and create the pod specification:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

5. The cluster capacity analysis is mounted in a volume using a **ConfigMap** named **cluster-capacity-configmap** to mount input pod spec file **pod.yaml** into a volume **test-volume** at the path **/test-pod**.
   If you haven't created a **ConfigMap**, create one before creating the job:

```
$ oc create configmap cluster-capacity-configmap \
    --from-file=pod.yaml=pod.yaml
```

6. Create the job using the below example of a job specification file:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
        containers:
        - name: cluster-capacity
          image: openshift/origin-cluster-capacity
          imagePullPolicy: "Always"
          volumeMounts:
          - mountPath: /test-pod
            name: test-volume
          env:
```

```
      - name: CC_INCLUSTER 1
        value: "true"
      command:
      - "/bin/sh"
      - "-ec"
      - |
        /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
    restartPolicy: "Never"
    serviceAccountName: cluster-capacity-sa
    volumes:
    - name: test-volume
      configMap:
        name: cluster-capacity-configmap
```

1 A required environment variable letting the cluster capacity tool know that it is running inside a cluster as a pod.
The **pod.yaml** key of the **ConfigMap** is the same as the pod specification file name, though it is not required. By doing this, the input pod spec file can be accessed inside the pod as **/test-pod/pod.yaml**.

7. Run the cluster capacity image as a job in a pod:

```
$ oc create -f cluster-capacity-job.yaml
```

8. Check the job logs to find the number of pods that can be scheduled in the cluster:

```
$ oc logs jobs/cluster-capacity-job
small-pod pod requirements:
        - CPU: 150m
        - Memory: 100Mi

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the
following predicates:: Insufficient cpu (2).

Pod distribution among nodes:
small-pod
        - 192.168.124.214: 26 instance(s)
        - 192.168.124.120: 26 instance(s)
```

6.3. CONFIGURING CLUSTER MEMORY TO MEET CONTAINER MEMORY AND RISK REQUIREMENTS

As a cluster administrator, you can help your clusters operate efficiently through managing application memory by:

- Determining the memory and risk requirements of a containerized application component and configuring the container memory parameters to suit those requirements.
- Configuring containerized application runtimes (for example, OpenJDK) to adhere optimally to the configured container memory parameters.

- Diagnosing and resolving memory-related error conditions associated with running in a container.

6.3.1. Understanding managing application memory

It is recommended to read fully the overview of how OpenShift Container Platform manages Compute Resources before proceeding.

For each kind of resource (memory, CPU, storage), OpenShift Container Platform allows optional **request** and **limit** values to be placed on each container in a pod.

Note the following about memory requests and memory limits:

- **Memory request**
  - The memory request value, if specified, influences the OpenShift Container Platform scheduler. The scheduler considers the memory request when scheduling a container to a node, then fences off the requested memory on the chosen node for the use of the container.
  - If a node's memory is exhausted, OpenShift Container Platform prioritizes evicting its containers whose memory usage most exceeds their memory request. In serious cases of memory exhaustion, the node OOM killer may select and kill a process in a container based on a similar metric.
  - The cluster administrator can assign quota or assign default values for the memory request value.
  - The cluster administrator may override the memory request values that a developer specifies, in order to manage cluster overcommit.
- **Memory limit**
  - The memory limit value, if specified, provides a hard limit on the memory that can be allocated across all the processes in a container.
  - If the memory allocated by all of the processes in a container exceeds the memory limit, the node OOM killer will immediately select and kill a process in the container.
  - If both memory request and limit are specified, the memory limit value must be greater than or equal to the memory request.
  - The cluster administrator can assign quota or assign default values for the memory limit value.

6.3.1.1. Managing application memory strategy

The steps for sizing application memory on OpenShift Container Platform are as follows:

1. **Determine expected container memory usage**
   Determine expected mean and peak container memory usage, empirically if necessary (for example, by separate load testing). Remember to consider all the processes that may potentially run in parallel in the container: for example, does the main application spawn any ancillary scripts?
2. **Determine risk appetite**
   Determine risk appetite for eviction. If the risk appetite is low, the container should request

memory according to the expected peak usage plus a percentage safety margin. If the risk appetite is higher, it may be more appropriate to request memory according to the expected mean usage.

3. **Set container memory request**
   Set container memory request based on the above. The more accurately the request represents the application memory usage, the better. If the request is too high, cluster and quota usage will be inefficient. If the request is too low, the chances of application eviction increase.

4. **Set container memory limit, if required**
   Set container memory limit, if required. Setting a limit has the effect of immediately killing a container process if the combined memory usage of all processes in the container exceeds the limit, and is therefore a mixed blessing. On the one hand, it may make unanticipated excess memory usage obvious early ("fail fast"); on the other hand it also terminates processes abruptly.

   Note that some OpenShift Container Platform clusters may require a limit value to be set; some may override the request based on the limit; and some application images rely on a limit value being set as this is easier to detect than a request value.

   If the memory limit is set, it should not be set to less than the expected peak container memory usage plus a percentage safety margin.

5. **Ensure application is tuned**
   Ensure application is tuned with respect to configured request and limit values, if appropriate. This step is particularly relevant to applications which pool memory, such as the JVM. The rest of this page discusses this.

6.3.2. Understanding OpenJDK settings for OpenShift Container Platform

The default OpenJDK settings do not work well with containerized environments. As a result, some additional Java memory settings must always be provided whenever running the OpenJDK in a container.

The JVM memory layout is complex, version dependent, and describing it in detail is beyond the scope of this documentation. However, as a starting point for running OpenJDK in a container, at least the following three memory-related tasks are key:

1. Overriding the JVM maximum heap size.

2. Encouraging the JVM to release unused memory to the operating system, if appropriate.

3. Ensuring all JVM processes within a container are appropriately configured.

Optimally tuning JVM workloads for running in a container is beyond the scope of this documentation, and may involve setting multiple additional JVM options.

6.3.2.1. Understanding how to override the JVM maximum heap size

For many Java workloads, the JVM heap is the largest single consumer of memory. Currently, the OpenJDK defaults to allowing up to 1/4 (1/**-XX:MaxRAMFraction**) of the compute node's memory to be used for the heap, regardless of whether the OpenJDK is running in a container or not. It is therefore **essential** to override this behavior, especially if a container memory limit is also set.

There are at least two ways the above can be achieved:

1. If the container memory limit is set and the experimental options are supported by the JVM, set **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap**.
   This sets **-XX:MaxRAM** to the container memory limit, and the maximum heap size (**-XX:MaxHeapSize** / **-Xmx**) to 1/ **-XX:MaxRAMFraction** (1/4 by default).

2. Directly override one of **-XX:MaxRAM**, **-XX:MaxHeapSize** or **-Xmx**.
   This option involves hard-coding a value, but has the advantage of allowing a safety margin to be calculated.

6.3.2.2. Understanding how to encourage the JVM to release unused memory to the operating system

By default, the OpenJDK does not aggressively return unused memory to the operating system. This may be appropriate for many containerized Java workloads, but notable exceptions include workloads where additional active processes co-exist with a JVM within a container, whether those additional processes are native, additional JVMs, or a combination of the two.

The OpenShift Container Platform Jenkins maven slave image uses the following JVM arguments to encourage the JVM to release unused memory to the operating system:

```
`-XX:+UseParallelGC
-XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4
-XX:AdaptiveSizePolicyWeight=90`.
```

These arguments are intended to return heap memory to the operating system whenever allocated memory exceeds 110% of in-use memory (**-XX:MaxHeapFreeRatio**), spending up to 20% of CPU time in the garbage collector (**-XX:GCTimeRatio**). At no time will the application heap allocation be less than the initial heap allocation (overridden by **-XX:InitialHeapSize** / **-Xms**). Detailed additional information is available Tuning Java's footprint in OpenShift (Part 1) , Tuning Java's footprint in OpenShift (Part 2) , and at OpenJDK and Containers.

6.3.2.3. Understanding how to ensure all JVM processes within a container are appropriately configured

In the case that multiple JVMs run in the same container, it is essential to ensure that they are all configured appropriately. For many workloads it will be necessary to grant each JVM a percentage memory budget, leaving a perhaps substantial additional safety margin.

Many Java tools use different environment variables (**JAVA_OPTS**, **GRADLE_OPTS**, **MAVEN_OPTS**, and so on) to configure their JVMs and it can be challenging to ensure that the right settings are being passed to the right JVM.

The **JAVA_TOOL_OPTIONS** environment variable is always respected by the OpenJDK, and values specified in **JAVA_TOOL_OPTIONS** will be overridden by other options specified on the JVM command line. By default, to ensure that these options are used by default for all JVM workloads run in the slave image, the OpenShift Container Platform Jenkins maven slave image sets:

```
`JAVA_TOOL_OPTIONS="-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true"`
```

This does not guarantee that additional options are not required, but is intended to be a helpful starting point.

6.3.3. Finding the memory request and limit from within a pod

An application wishing to dynamically discover its memory request and limit from within a pod should use the Downward API.

Procedure

1. Configure the pod to add the **MEMORY_REQUEST** and **MEMORY_LIMIT** stanzas:

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: test
    image: fedora:latest
    command:
    - sleep
    - "3600"
    env:
    - name: MEMORY_REQUEST (1)
      valueFrom:
        resourceFieldRef:
          containerName: test
          resource: requests.memory
    - name: MEMORY_LIMIT (2)
      valueFrom:
        resourceFieldRef:
          containerName: test
          resource: limits.memory
    resources:
      requests:
        memory: 384Mi
      limits:
        memory: 512Mi
```

(1) Add this stanza to discover the application memory request value.

(2) Add this stanza to discover the application memory limit value.

1. Create the pod:

```
$ oc create -f <file-name>.yaml
```

2. Access the pod using a remote shell:

```
$ oc rsh test
```

3. Check that the requested values were applied:

```
$ env | grep MEMORY | sort
MEMORY_LIMIT=536870912
MEMORY_REQUEST=402653184
```

NOTE

The memory limit value can also be read from inside the container by the **/sys/fs/cgroup/memory/memory.limit_in_bytes** file.

6.3.4. Understanding OOM kill policy

OpenShift Container Platform may kill a process in a container if the total memory usage of all the processes in the container exceeds the memory limit, or in serious cases of node memory exhaustion.

When a process is OOM killed, this may or may not result in the container exiting immediately. If the container PID 1 process receives the **SIGKILL**, the container will exit immediately. Otherwise, the container behavior is dependent on the behavior of the other processes.

For example, a container process exited with code 137, indicating it received a SIGKILL signal.

If the container does not exit immediately, an OOM kill is detectable as follows:

1. Access the pod using a remote shell:

```
# oc rsh test
```

2. The oom_kill counter in **/sys/fs/cgroup/memory/memory.oom_control** is incremented

```
$ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
oom_kill 0
$ sed -e '' </dev/zero  # provoke an OOM kill
Killed
$ echo $?
137
$ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
oom_kill 1
```

If one or more processes in a pod are OOM killed, when the pod subsequently exits, whether immediately or not, it will have phase **Failed** and reason **OOMKilled**. An OOM killed pod may be restarted depending on the value of **restartPolicy**. If not restarted, controllers such as the ReplicationController will notice the pod's failed status and create a new pod to replace the old one.

If not restarted, the pod status is as follows:

```
$ oc get pod test
NAME      READY     STATUS      RESTARTS   AGE
test      0/1       OOMKilled   0          1m

$ oc get pod test -o yaml
...
status:
  containerStatuses:
  - name: test
    ready: false
    restartCount: 0
    state:
      terminated:
```

```
      exitCode: 137
      reason: OOMKilled
  phase: Failed
```

If restarted, its status is as follows:

```
$ oc get pod test
NAME      READY     STATUS    RESTARTS   AGE
test      1/1       Running   1          1m

$ oc get pod test -o yaml
...
status:
  containerStatuses:
  - name: test
    ready: true
    restartCount: 1
    lastState:
      terminated:
        exitCode: 137
        reason: OOMKilled
    state:
      running:
  phase: Running
```

6.3.5. Understanding pod eviction

OpenShift Container Platform may evict a pod from its node when the node's memory is exhausted. Depending on the extent of memory exhaustion, the eviction may or may not be graceful. Graceful eviction implies the main process (PID 1) of each container receiving a SIGTERM signal, then some time later a SIGKILL signal if the process has not exited already. Non-graceful eviction implies the main process of each container immediately receiving a SIGKILL signal.

An evicted pod will have phase **Failed** and reason **Evicted**. It will not be restarted, regardless of the value of **restartPolicy**. However, controllers such as the ReplicationController will notice the pod's failed status and create a new pod to replace the old one.

```
$ oc get pod test
NAME      READY     STATUS    RESTARTS   AGE
test      0/1       Evicted   0          1m

$ oc get pod test -o yaml
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
  phase: Failed
  reason: Evicted
```

6.4. CONFIGURING YOUR CLUSTER TO PLACE PODS ON OVERCOMMITED NODES

In an *overcommited* state, the sum of the container compute resource requests and limits exceeds the resources available on the system. Overcommitment might be desirable in development environments where a tradeoff of guaranteed performance for capacity is acceptable.

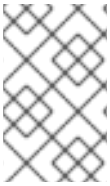
NOTE

In OpenShift Container Platform overcommittment is enabled by default. See Disabling overcommitment for a node.

6.4.1. Understanding overcommitment

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node. The scheduler uses requests for scheduling your container and providing a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

OpenShift Container Platform administrators can control the level of overcommit and manage container density on nodes by configuring masters 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.



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.

After these 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.

6.4.2. Understanding resource requests and overcommitment

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.

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.

6.4.2.1. Understanding Buffer Chunk Limiting for Fluentd

If the Fluentd logger is unable to keep up with a high number of logs, it will need to switch to file buffering to reduce memory usage and prevent data loss.

Fluentd file buffering stores records in *chunks*. Chunks are stored in *buffers*.

The Fluentd **buffer_chunk_limit** is determined by the environment variable **BUFFER_SIZE_LIMIT**,

which has the default value **8m**. The file buffer size per output is determined by the environment variable **FILE_BUFFER_LIMIT**, which has the default value **256Mi**. The permanent volume size must be larger than **FILE_BUFFER_LIMIT** multiplied by the output.

On the Fluentd pods, permanent volume **/var/lib/fluentd** should be prepared by the PVC or hostmount, for example. That area is then used for the file buffers.

The **buffer_type** and **buffer_path** are configured in the Fluentd configuration files as follows:

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-config`
output-es-ops-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
```

The Fluentd **buffer_queue_limit** is the value of the variable **BUFFER_QUEUE_LIMIT**. This value is **32** by default.

The environment variable **BUFFER_QUEUE_LIMIT** is calculated as **(FILE_BUFFER_LIMIT / (number_of_outputs * BUFFER_SIZE_LIMIT))**.

If the **BUFFER_QUEUE_LIMIT** variable has the default set of values:

- **FILE_BUFFER_LIMIT = 256Mi**
- **number_of_outputs = 1**
- **BUFFER_SIZE_LIMIT = 8Mi**

The value of **buffer_queue_limit** will be **32**. To change the **buffer_queue_limit**, you must change the value of **FILE_BUFFER_LIMIT**.

In this formula, **number_of_outputs** is **1** if all the logs are sent to a single resource, and it is incremented by **1** for each additional resource. For example, the value of **number_of_outputs** is:

- **1** - if all logs are sent to a single ElasticSearch pod
- **2** - if application logs are sent to an ElasticSearch pod and ops logs are sent to another ElasticSearch pod
- **4** - if application logs are sent to an ElasticSearch pod, ops logs are sent to another ElasticSearch pod, and both of them are forwarded to other Fluentd instances

6.4.3. Understanding compute resources and containers

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

6.4.3.1. Understanding container CPU requests

A container is guaranteed the amount of CPU it requests and is additionally able to consume excess CPU available on the node, up to any limit specified by the container. If multiple containers are attempting to use excess CPU, CPU time is distributed based on the amount of CPU requested by each container.

For example, if one container requested 500m of CPU time and another container requested 250m of CPU time, then any extra CPU time available on the node is distributed among the containers in a 2:1 ratio. If a container specified a limit, it will be throttled not to use more CPU than the specified limit. CPU requests are enforced using the CFS shares support in the Linux kernel. By default, CPU limits are enforced using the CFS quota support in the Linux kernel over a 100ms measuring interval, though this can be disabled.

6.4.3.2. Understanding container memory requests

A container is guaranteed the amount of memory it requests. A container can use more memory than requested, but once it exceeds its requested amount, it could be terminated in a low memory situation on the node. If a container uses less memory than requested, it will not be terminated 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 terminated if it exceeds the limit amount.

6.4.4. Understanding overcomitment and 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 6.19. Quality of Service Classes

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

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

- **Guaranteed** containers are considered top priority, and are guaranteed to only be terminated 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 terminated 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 terminated if the system runs out of memory.

6.4.4.1. Understanding how to reserve memory across quality of service tiers

You can use the **qos-reserved** parameter to specify a percentage of memory to be reserved by a pod in a particular QoS level. This feature attempts to reserve requested resources to exclude pods from lower OoS classes from using resources requested by pods in higher QoS classes.

OpenShift Container Platform uses the **qos-reserved** parameter as follows:

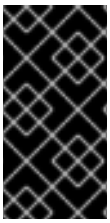
- A value of **qos-reserved=memory=100%** will prevent the **Burstable** and **BestEffort** QOS classes from consuming memory that was requested by a higher QoS class. This increases the risk of inducing OOM on **BestEffort** and **Burstable** workloads in favor of increasing memory resource guarantees for **Guaranteed** and **Burstable** workloads.
- A value of **qos-reserved=memory=50%** will allow the **Burstable** and **BestEffort** QOS classes to consume half of the memory requested by a higher QoS class.
- A value of **qos-reserved=memory=0%** will allow a **Burstable** and **BestEffort** QoS classes to consume up to the full node allocatable amount if available, but increases the risk that a **Guaranteed** workload will not have access to requested memory. This condition effectively disables this feature.

6.4.5. Understanding swap memory and QOS

You can disable swap by default on your nodes in order to preserve quality of service (QOS) 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.

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.

6.4.6. Understanding nodes overcommitment

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

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.

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

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, OpenShift Container Platform configures the kernel to always overcommit memory by setting the **vm.overcommit_memory** parameter to **1**, overriding the default operating system setting.

OpenShift Container Platform also configures the kernel not to panic when it runs out of memory by setting the **vm.panic_on_oom** parameter to **0**. A setting of 0 instructs the kernel to call oom_killer in an Out of Memory (OOM) condition, which kills processes based on priority

You can view the current setting by running the following commands on your nodes:

```
$ sysctl -a |grep commit

vm.overcommit_memory = 1
```

```
$ sysctl -a |grep panic
vm.panic_on_oom = 0
```

NOTE

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

You can also perform the following configurations for each node:

- Disable or enforce CPU limits using CPU CFS quotas
- Reserve resources for system processes
- Reserve memory across quality of service tiers

6.4.6.1. Disabling or enforcing CPU limits using CPU CFS quotas

Nodes by default enforce specified CPU limits using the Completely Fair Scheduler (CFS) quota support in the Linux kernel.

Prerequisites

1. Obtain the label associated with the static Machine Config Pool CRD for the type of node you want to configure. Perform one of the following steps:

   a. View the Machine Config Pool:

   ```
   $ oc describe machineconfigpool <name>
   ```

   For example:

   ```
   $ oc describe machineconfigpool worker

   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: small-pods 1
   ```

```
...
```

1 If a label has been added it appears under **labels**.

b. If the label is not present, add a key/value pair:

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procedure

1. Create a Custom Resource (CR) for your configuration change.

Sample configuration for a disabling CPU limits

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods 2
  kubeletConfig:
    cpu-cfs-quota: 3
      - "false"
```

1 Assign a name to CR.

2 Specify the label to apply the configuration change.

3 Set the **cpu-cfs-quota** parameter to **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.

6.4.6.2. Reserving resources for system processes

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by system daemons that are required to run on your node for your cluster to function (**sshd**, etc.). In particular, it is recommended that you reserve resources for incompressible resources such as memory.

Procedure

To explicitly reserve resources for non-pod processes, allocate node resources by specifying resources available for scheduling. For more details, see Allocating Resources for Nodes.

6.4.7. Disabling overcommitment for a node

When enabled, overcommitment can be disabled on each node.

Procedure

To disable overcommitment in a node run the following command on that node:

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

6.4.8. Disabling overcommitment for a project

When enabled, overcommitment can be disabled per-project. For example, you can allow infrastructure components to be configured independently of overcommitment.

Procedure

To disable overcommitment in a project:

1. Edit the project object file
2. Add the following annotation:

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

3. Create the project object:

```
$ oc create -f <file-name>.yaml
```

6.5. ENABLING OPENSHIFT CONTAINER PLATFORM FEATURES USING FEATUREGATES

As an administrator, you can turn on features that are Technology Preview features.

6.5.1. Understanding FeatureGates and Technology Preview features

You can use the FeatureGates Custom Resource to toggle on and off Technology Preview features throughout your cluster.

This allows you, for example, to ensure that Technology Preview features are off for production clusters while leaving the features on for test clusters where you can fully test them.

6.5.2. Features that are affected by FeatureGates

The following features are affected by FeatureGates:

FeatureGate	Description	Default
ExperimentalCriticalPodAnnotation	Enables annotating specific Pods as critical so that their scheduling is guaranteed.	True
RotateKubeletServerCertificate	Enables the rotation of the server TLS certificate on the cluster.	True
SupportPodPidsLimit	Enables support for limiting the number of processes (PIDs) running in a Pod.	True
MachineHealthCheck	Enables automatically repairing unhealthy machines in a machine pool.	True
LocalStorageCapacityIsolation	Enable the consumption of local ephemeral storage and also the **sizeLimit** property of an **emptyDir** volume.	False

You can enable these features by editing the Feature Gate Custom Resource. Turning on these features cannot be undone and prevents the ability to upgrade your cluster.

The **LocalStorageCapacityIsolation** cannot be enabled.

6.5.3. Enabling Technology Preview features using FeatureGates

You can turn Technology Preview features on and off for all nodes in the cluster by editing the FeatureGates Custom Resource, named **cluster**, in the **openshift-config** project.

The following Technology Preview features are enabled by feature gates:

- **ExperimentalCriticalPodAnnotation**
- **RotateKubeletServerCertificate**
- **SupportPodPidsLimit**
- **MachineHealthCheck**

IMPORTANT

Turning on Technology Preview features cannot be undone and prevents upgrades.

Procedure

To turn on the Technology Preview features for the entire cluster:

1. Create the FeatureGates instance:
   a. Switch to the the **Administration → Custom Resource Definitions** page.
   b. On the **Custom Resource Definitions** page, click **FeatureGate**.
   c. On the **Custom Resource Definitions** page, click the **Actions Menu** and select **View Instances**.
   d. On the **Feature Gates** page, click **Create Feature Gates**.
   e. Replace the code with following sample:

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
spec: {}
```

   f. Click **Create**.
2. To turn on the Technology Preview features, change the **spec** parameter to:

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster
spec:
  featureSet: TechPreviewNoUpgrade (1)
```

(1) Add **featureSet: TechPreviewNoUpgrade** to enable the Technology Preview features that are affected by FeatureGates.

NOTE

Turning on Technology Preview features cannot be undone and prevents upgrades.