OpenShift Dedicated 4

Nodes

OpenShift Dedicated Nodes

OpenShift Dedicated 4 Nodes

OpenShift Dedicated Nodes

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. OVERVIEW OF NODES

1.1. ABOUT NODES

A node is a virtual or bare-metal machine in a Kubernetes cluster. Worker nodes host your application containers, grouped as pods. The control plane nodes run services that are required to control the Kubernetes cluster. In OpenShift Dedicated, the control plane nodes contain more than just the Kubernetes services for managing the OpenShift Dedicated cluster.

Having stable and healthy nodes in a cluster is fundamental to the smooth functioning of your hosted application. In OpenShift Dedicated, you can access, manage, and monitor a node through the **Node** object representing the node. Using the OpenShift CLI (**oc**) or the web console, you can perform the following operations on a node.

The following components of a node are responsible for maintaining the running of pods and providing the Kubernetes runtime environment.

Container runtime

> The container runtime is responsible for running containers. OpenShift Dedicated deploys the CRI-O container runtime on each of the Red Hat Enterprise Linux CoreOS (RHCOS) nodes in your cluster. The Windows Machine Config Operator (WMCO) deploys the containerd runtime on its Windows nodes.

Kubelet

> Kubelet runs on nodes and reads the container manifests. It ensures that the defined containers have started and are running. The kubelet process maintains the state of work and the node server. Kubelet manages network rules and port forwarding. The kubelet manages containers that are created by Kubernetes only.

DNS

> Cluster DNS is a DNS server which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
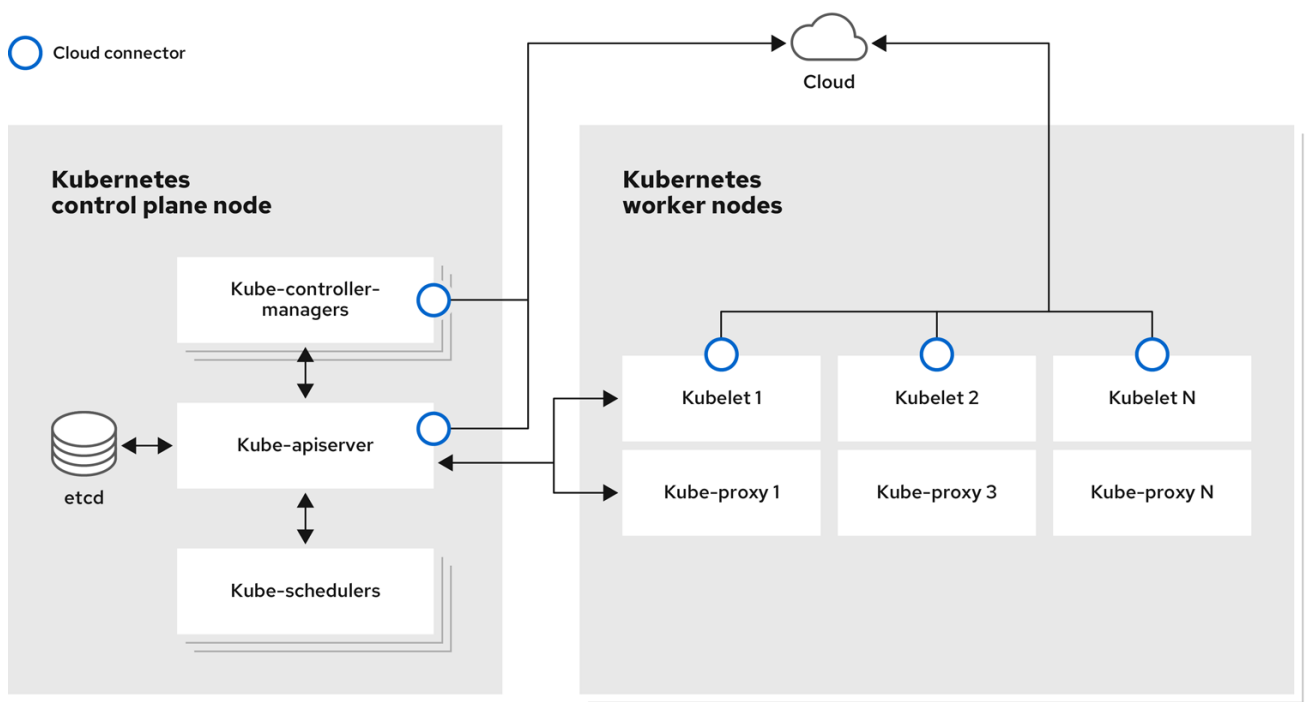
1.1.1. Read operations

The read operations allow an administrator or a developer to get information about nodes in an OpenShift Dedicated cluster.

- List all the nodes in a cluster .

- Get information about a node, such as memory and CPU usage, health, status, and age.

- List pods running on a node .

1.1.2. Enhancement operations

OpenShift Dedicated allows you to do more than just access and manage nodes; as an administrator, you can perform the following tasks on nodes to make the cluster more efficient, application-friendly, and to provide a better environment for your developers.

- Manage node-level tuning for high-performance applications that require some level of kernel tuning by using the Node Tuning Operator .

- Run background tasks on nodes automatically with daemon sets . You can create and use daemon sets to create shared storage, run a logging pod on every node, or deploy a monitoring agent on all nodes.

1.2. ABOUT PODS

A pod is one or more containers deployed together on a node. As a cluster administrator, you can define a pod, assign it to run on a healthy node that is ready for scheduling, and manage. A pod runs as long as the containers are running. You cannot change a pod once it is defined and is running. Some operations you can perform when working with pods are:

1.2.1. Read operations

As an administrator, you can get information about pods in a project through the following tasks:

- List pods associated with a project , including information such as the number of replicas and restarts, current status, and age.

- View pod usage statistics such as CPU, memory, and storage consumption.

1.2.2. Management operations

The following list of tasks provides an overview of how an administrator can manage pods in an OpenShift Dedicated cluster.

- Control scheduling of pods using the advanced scheduling features available in OpenShift Dedicated:

  - Node-to-pod binding rules such as pod affinity, node affinity, and anti-affinity.

  - Node labels and selectors .

  - Pod topology spread constraints .

- Configure how pods behave after a restart using pod controllers and restart policies .

- Limit both egress and ingress traffic on a pod .

- Add and remove volumes to and from any object that has a pod template . A volume is a mounted file system available to all the containers in a pod. Container storage is ephemeral; you can use volumes to persist container data.

1.2.3. Enhancement operations

You can work with pods more easily and efficiently with the help of various tools and features available in OpenShift Dedicated. The following operations involve using those tools and features to better manage pods.

- Secrets: Some applications need sensitive information, such as passwords and usernames. An administrator can use the **Secret** object to provide sensitive data to pods using the **Secret** object.

1.3. ABOUT CONTAINERS

A container is the basic unit of an OpenShift Dedicated application, which comprises the application code packaged along with its dependencies, libraries, and binaries. Containers provide consistency across environments and multiple deployment targets: physical servers, virtual machines (VMs), and private or public cloud.

Linux container technologies are lightweight mechanisms for isolating running processes and limiting access to only designated resources. As an administrator, You can perform various tasks on a Linux container, such as:

- Copy files to and from a container .

- Allow containers to consume API objects .

- Execute remote commands in a container .

- Use port forwarding to access applications in a container .

OpenShift Dedicated provides specialized containers called Init containers. Init containers run before application containers and can contain utilities or setup scripts not present in an application image. You can use an Init container to perform tasks before the rest of a pod is deployed.

Apart from performing specific tasks on nodes, pods, and containers, you can work with the overall OpenShift Dedicated cluster to keep the cluster efficient and the application pods highly available.

1.4. GLOSSARY OF COMMON TERMS FOR OPENSHIFT DEDICATED NODES

This glossary defines common terms that are used in the *node* content.

Container

It is a lightweight and executable image that comprises software and all its dependencies. Containers virtualize the operating system, as a result, you can run containers anywhere from a data center to a public or private cloud to even a developer's laptop.

Daemon set

Ensures that a replica of the pod runs on eligible nodes in an OpenShift Dedicated cluster.

egress

The process of data sharing externally through a network's outbound traffic from a pod.

garbage collection

The process of cleaning up cluster resources, such as terminated containers and images that are not referenced by any running pods.

Ingress

Incoming traffic to a pod.

Job

A process that runs to completion. A job creates one or more pod objects and ensures that the specified pods are successfully completed.

Labels

You can use labels, which are key-value pairs, to organise and select subsets of objects, such as a pod.

Node

A worker machine in the OpenShift Dedicated cluster. A node can be either be a virtual machine (VM) or a physical machine.

Node Tuning Operator

You can use the Node Tuning Operator to manage node-level tuning by using the TuneD daemon. It ensures custom tuning specifications are 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.

Self Node Remediation Operator

The Operator runs on the cluster nodes and identifies and reboots nodes that are unhealthy.

Pod

One or more containers with shared resources, such as volume and IP addresses, running in your OpenShift Dedicated cluster. A pod is the smallest compute unit defined, deployed, and managed.

Toleration

Indicates that the pod is allowed (but not required) to be scheduled on nodes or node groups with matching taints. You can use tolerations to enable the scheduler to schedule pods with matching taints.

Taint

A core object that comprises a key, value, and effect. Taints and tolerations work together to ensure that pods are not scheduled on irrelevant nodes.

CHAPTER 2. WORKING WITH PODS

2.1. 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.

2.1.1. 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 to enable access to the logs of their containers.

OpenShift Dedicated treats pods as largely immutable; changes cannot be made to a pod definition while it is running. OpenShift Dedicated 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.

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

2.1.2. Example pod configurations

OpenShift Dedicated 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. 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
  labels:
    environment: production
    app: abc 1
spec:
  restartPolicy: Always 2
  securityContext: 3
```

```
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers: 4
   - name: abc
     args:
     - sleep
     - "1000000"
     volumeMounts: 5
      - name: cache-volume
        mountPath: /cache 6
     image: registry.access.redhat.com/ubi7/ubi-init:latest 7
     securityContext:
       allowPrivilegeEscalation: false
       runAsNonRoot: true
       capabilities:
         drop: ["ALL"]
     resources:
       limits:
         memory: "100Mi"
         cpu: "1"
       requests:
         memory: "100Mi"
         cpu: "1"
  volumes: 8
  - name: cache-volume
    emptyDir:
      sizeLimit: 500Mi
```

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.

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

3 OpenShift Dedicated 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 one or more container definitions.

5 The container specifies where external storage volumes are mounted within the container.

6 Specify the volumes to provide for the pod. Volumes mount at the specified path. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host **/dev/pts** files. It is safe to mount the host by using **/host**.

7 Each container in the pod is instantiated from its own container image.

8 The pod defines storage volumes that are available to its container(s) to use.

If you attach persistent volumes that have high file counts to pods, those pods can fail or can take a long time to start. For more information, see When using Persistent Volumes with high file counts

in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state?.

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

2.1.3. Understanding resource requests and limits

You can specify CPU and memory requests and limits for pods by using a pod spec, as shown in "Example pod configurations", or the specification for the controlling object of the pod.

CPU and memory *requests* specify the minimum amount of a resource that a pod needs to run, helping OpenShift Dedicated to schedule pods on nodes with sufficient resources.

CPU and memory *limits* define the maximum amount of a resource that a pod can consume, preventing the pod from consuming excessive resources and potentially impacting other pods on the same node.

CPU and memory requests and limits are processed by using the following principles:

- CPU limits are enforced by using CPU throttling. When a container approaches its CPU limit, the kernel restricts access to the CPU specified as the container's limit. As such, a CPU limit is a hard limit that the kernel enforces. OpenShift Dedicated can allow a container to exceed its CPU limit for extended periods of time. However, container runtimes do not terminate pods or containers for excessive CPU usage.
  CPU limits and requests are measured in CPU units. One CPU unit is equivalent to 1 physical CPU core or 1 virtual core, depending on whether the node is a physical host or a virtual machine running inside a physical machine. Fractional requests are allowed. For example, when you define a container with a CPU request of **0.5**, you are requesting half as much CPU time than if you asked for **1.0** CPU. For CPU units, **0.1** is equivalent to the **100m**, which can be read as *one hundred millicpu* or *one hundred millicores*. A CPU resource is always an absolute amount of resource, and is never a relative amount.

  > **NOTE**
  >
  > By default, the smallest amount of CPU that can be allocated to a pod is 10 mCPU. You can request resource limits lower than 10 mCPU in a pod spec. However, the pod would still be allocated 10 mCPU.

- Memory limits are enforced by the kernel by using out of memory (OOM) kills. When a container uses more than its memory limit, the kernel can terminate that container. However, terminations happen only when the kernel detects memory pressure. As such, a container that over allocates memory might not be immediately killed. This means memory limits are enforced reactively. A container can use more memory than its memory limit. If it does, the container can get killed. You can express memory as a plain integer or as a fixed-point number by using one of these quantity suffixes: **E**, **P**, **T**, **G**, **M**, or **k**. You can also use the power-of-two equivalents: **Ei**, **Pi**, **Ti**, **Gi**, **Mi**, or **Ki**.

If the node where a pod is running has enough of a resource available, it is possible for a container to use more CPU or memory resources than it requested. However, the container cannot exceed the corresponding limit. For example, if you set a container memory request of **256 MiB**, and that container

is in a pod scheduled to a node with **8GiB** of memory and no other pods, the container can try to use more memory than the requested **256 MiB**.

This behavior does not apply to CPU and memory limits. These limits are applied by the kubelet and the container runtime, and are enforced by the kernel. On Linux nodes, the kernel enforces limits by using cgroups.

2.1.4. Additional resources

- For more information on pods and storage see Understanding persistent storage and Understanding ephemeral storage .

- Example pod configurations

2.2. VIEWING PODS

As an administrator, you can view cluster pods, check their health, and evaluate the overall health of the cluster. You can also view a list of pods associated with a specific project or view usage statistics about pods. Regularly viewing pods can help you detect problems early, track resource usage, and ensure cluster stability.

2.2.1. Viewing pods in a project

You can display pod usage statistics, such as CPU, memory, and storage consumption, to monitor container runtime environments and ensure efficient resource use.

Procedure

1. Change to the project by entering the following command:

   ```
   $ oc project <project_name>
   ```

2. Obtain a list of pods by entering the following command:

   ```
   $ oc get pods
   ```

   Example output

   ```
   NAME                    READY  STATUS   RESTARTS  AGE
   console-698d866b78-bnshf  1/1    Running  2         165m
   console-698d866b78-m87pm  1/1    Running  2         165m
   ```

3. Optional: Add the **-o wide** flags to view the pod IP address and the node where the pod is located. For example:

   ```
   $ oc get pods -o wide
   ```

   Example output

   ```
   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>
```

2.2.2. 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

1. View the usage statistics by entering the following command:

   ```
   $ oc adm top pods -n <namespace>
   ```

   Example output

   ```
   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. Optional: Add the **--selector=''** label to view usage statistics for pods with labels. Note that you must choose the label query to filter on, such as **=**, **==**, or **!=**. For example:

   ```
   $ oc adm top pod --selector='<pod_name>'
   ```

2.2.3. Viewing resource logs

You can view logs for resources in the OpenShift CLI (oc) or web console. Logs display from the end (or tail) by default. Viewing logs for resources can help you troubleshoot issues and monitor resource behavior.

2.2.3.1. Viewing resource logs by using the web console

Use the following procedure to view resource logs by using the OpenShift Dedicated web console.

Procedure

1. In the OpenShift Dedicated console, navigate to **Workloads → Pods** or navigate to the pod through the resource you want to investigate.

> **NOTE**
>
> Some resources, such as builds, do not have pods to query directly. In such instances, you can locate the **Logs** link on the **Details** page for the resource.

2. Select a project from the drop-down menu.

3. Click the name of the pod you want to investigate.

4. Click **Logs**.

2.2.3.2. Viewing resource logs by using the CLI

Use the following procedure to view resource logs by using the command-line interface (CLI).

Prerequisites

- Access to the OpenShift CLI (**oc**).

Procedure

- View the log for a specific pod by entering the following command:

  ```
  $ oc logs -f <pod_name> -c <container_name>
  ```

  where:

  -f

  Optional: Specifies that the output follows what is being written into the logs.

  <pod_name>

  Specifies the name of the pod.

  <container_name>

  Optional: Specifies the name of a container. When a pod has more than one container, you must specify the container name.

  For example:

  ```
  $ oc logs -f ruby-57f7f4855b-znl92 -c ruby
  ```

- View the log for a specific resource by entering the following command:

  ```
  $ oc logs <object_type>/<resource_name>
  ```

  For example:

  ```
  $ oc logs deployment/ruby
  ```

2.3. CONFIGURING AN OPENSHIFT DEDICATED 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.

2.3.1. Configuring how pods behave after restart

A pod restart policy determines how OpenShift Dedicated 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) capped at 5 minutes. 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	Daemon set	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 Dedicated 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 Dedicated 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 Dedicated uses restart policy with failed Containers, see the Example States in the Kubernetes documentation.

2.3.2. 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>
   ```

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

A *pod disruption budget* allows 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**
>
> **Available** refers to the number of pods that has condition **Ready=True**. **Ready=True** refers to the pod that is able to serve requests and should be added to the load balancing pools of all matching services.
>
> 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.

> **WARNING**
>
> The default setting for **maxUnavailable** is **1** for all the machine config pools in OpenShift Dedicated. It is recommended to not change this value and update one control plane node at a time. Do not change this value to **3** for the control plane pool.

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

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

> **NOTE**
>
> The following example contains some values that are specific to OpenShift Dedicated on AWS.

Example output

```
NAMESPACE                       NAME                              MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED DISRUPTIONS   AGE
openshift-apiserver             openshift-apiserver-pdb           N/A             1                 1                     121m
openshift-cloud-controller-manager   aws-cloud-controller-manager   1          N/A               1                     125m
openshift-cloud-credential-operator  pod-identity-webhook           1          N/A               1                     117m
openshift-cluster-csi-drivers   aws-ebs-csi-driver-controller-pdb  N/A         1                 1                     121m
openshift-cluster-storage-operator   csi-snapshot-controller-pdb   N/A          1                 1
```

```
122m
openshift-cluster-storage-operator     csi-snapshot-webhook-pdb          N/A          1          1
122m
openshift-console                 console               N/A          1          1
116m
#...
```

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.

2.3.3.1. 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/v1 ❶
   kind: PodDisruptionBudget
   metadata:
     name: my-pdb
   spec:
     minAvailable: 2 ❷
     selector: ❸
       matchLabels:
         name: my-pod
   ```

   ❶ **PodDisruptionBudget** is part of the **policy/v1** API group.

   ❷ 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%**.

   ❸ A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined. Leave this parameter blank, for example **selector {}**, to select all pods in the project.

   Or:

   ```
   apiVersion: policy/v1 ❶
   kind: PodDisruptionBudget
   metadata:
     name: my-pdb
   spec:
     maxUnavailable: 25% ❷
   ```

```
    selector: 3
     matchLabels:
       name: my-pod
```

1 **PodDisruptionBudget** is part of the **policy/v1** 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. Leave this parameter blank, for example **selector {}**, to select all pods in the project.

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

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

2.4. PROVIDING SENSITIVE DATA TO PODS BY USING SECRETS

Additional resources

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.

2.4.1. Understanding secrets

The **Secret** object type provides a mechanism to hold sensitive information such as passwords, OpenShift Dedicated 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 plugin 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: <username> 3
```

```
    password: <password>
  stringData: 4
    hostname: myapp.mydomain.com 5
```

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).

2.4.1.1. 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/basic-auth**: Use with Basic authentication

- **kubernetes.io/dockercfg**: Use as an image pull secret

- **kubernetes.io/dockerconfigjson**: Use as an image pull secret

- **kubernetes.io/service-account-token**: Use to obtain a legacy service account API token

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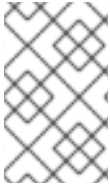
> **NOTE**
>
> You can specify other arbitrary types, such as **example.com/my-secret-type**. These types are not enforced server-side, but indicate that the creator of the secret intended to conform to the key/value requirements of that type.

For examples of creating different types of secrets, see *Understanding how to create secrets*.

2.4.1.2. Secret data keys

Secret keys must be in a DNS subdomain.

2.4.1.3. Automatically generated image pull secrets

By default, OpenShift Dedicated creates an image pull secret for each service account.

> **NOTE**
>
> Prior to OpenShift Dedicated 4.16, a long-lived service account API token secret was also generated for each service account that was created. Starting with OpenShift Dedicated 4.16, this service account API token secret is no longer created.
>
> After upgrading to 4, any existing long-lived service account API token secrets are not deleted and will continue to function. For information about detecting long-lived API tokens that are in use in your cluster or deleting them if they are not needed, see the Red Hat Knowledgebase article Long-lived service account API tokens in OpenShift Container Platform.

This image pull secret is necessary to integrate the OpenShift image registry into the cluster's user authentication and authorization system.

However, if you do not enable the **ImageRegistry** capability or if you disable the integrated OpenShift image registry in the Cluster Image Registry Operator's configuration, an image pull secret is not generated for each service account.

When the integrated OpenShift image registry is disabled on a cluster that previously had it enabled, the previously generated image pull secrets are deleted automatically.

2.4.2. 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:

1. Create a secret object that contains the data you want to keep secret. The specific data required for each secret type is descibed in the following sections.

   Example YAML object that creates an opaque secret

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: test-secret
   ```

```
type: Opaque ❶
data: ❷
  username: <username>
  password: <password>
stringData: ❸
  hostname: myapp.mydomain.com
  secret.properties: |
    property1=valueA
    property2=valueB
```

❶   Specifies the type of secret.

❷   Specifies encoded string and data.

❸   Specifies decoded string and data.

Use either the **data** or **stringdata** fields, not both.

2. Update the pod's service account to reference the secret:

YAML of a service account that uses a secret

```
apiVersion: v1
kind: ServiceAccount
 ...
secrets:
- name: test-secret
```

3. Create a pod, which consumes the secret as an environment variable or as a file (using a **secret** volume):

YAML of a pod populating files in a volume with secret data

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
   - name: secret-test-container
     image: busybox
     command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
     volumeMounts: ❶
       - name: secret-volume
         mountPath: /etc/secret-volume ❷
         readOnly: true ❸
     securityContext:
       allowPrivilegeEscalation: false
       capabilities:
         drop: [ALL]
```

```
    volumes:
     - name: secret-volume
       secret:
         secretName: test-secret 4
    restartPolicy: Never
```

1   Add a **volumeMounts** field to each container that needs the secret.

2   Specifies an unused directory name where you would like the secret to appear. Each key in the secret data map becomes the filename under **mountPath**.

3   Set to **true**. If true, this instructs the driver to provide a read-only volume.

4   Specifies the name of the secret.

YAML of a pod populating environment variables with secret data

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
   - name: secret-test-container
     image: busybox
     command: [ "/bin/sh", "-c", "export" ]
     env:
       - name: TEST_SECRET_USERNAME_ENV_VAR
         valueFrom:
           secretKeyRef: 1
             name: test-secret
             key: username
     securityContext:
       allowPrivilegeEscalation: false
       capabilities:
         drop: [ALL]
  restartPolicy: Never
```

1   Specifies the environment variable that consumes the secret key.

YAML of a build config populating environment variables with secret data

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
```

```
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef: 1
            name: test-secret
            key: username
      from:
        kind: ImageStreamTag
        namespace: openshift
        name: 'cli:latest'
```

1   Specifies the environment variable that consumes the secret key.

2.4.2.1. 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 namespace.

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 a **Secret** object. 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.

2.4.2.2. Creating an opaque secret

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

Procedure

1. Create a **Secret** object in a YAML file.
   For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: mysecret
   type: Opaque 1
   ```

```
data:
  username: <username>
  password: <password>
```

1   Specifies an opaque secret.

2. Use the following command to create a **Secret** object:

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

3. To use the secret in a pod:

   a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.

   b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- Understanding how to create secrets

2.4.2.3. Creating a legacy service account token secret

As an administrator, you can create a legacy service account token secret, which allows you to distribute a service account token to applications that must authenticate to the API.

> **WARNING**
>
> It is recommended to obtain bound service account tokens using the TokenRequest API instead of using legacy service account token secrets. You should create a service account token secret only if you cannot use the TokenRequest API and if the security exposure of a nonexpiring token in a readable API object is acceptable to you.
>
> Bound service account tokens are more secure than service account token secrets for the following reasons:
>
> - Bound service account tokens have a bounded lifetime.
>
> - Bound service account tokens contain audiences.
>
> - Bound service account tokens can be bound to pods or secrets and the bound tokens are invalidated when the bound object is removed.
>
> Workloads are automatically injected with a projected volume to obtain a bound service account token. If your workload needs an additional service account token, add an additional projected volume in your workload manifest.
>
> For more information, see "Configuring bound service account tokens using volume projection".

Procedure

1. Create a **Secret** object in a YAML file:

   Example Secret object

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: secret-sa-sample
     annotations:
       kubernetes.io/service-account.name: "sa-name" 1
   type: kubernetes.io/service-account-token 2
   ```

   1 Specifies an existing service account name. If you are creating both the **ServiceAccount** and the **Secret** objects, create the **ServiceAccount** object first.

   2 Specifies a service account token secret.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:

    a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.

    b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)

2.4.2.4. Creating a basic authentication secret

As an administrator, you can create a basic authentication secret, which allows you to store the credentials needed for basic authentication. When using this secret type, the **data** parameter of the **Secret** object must contain the following keys encoded in the base64 format:

- **username**: the user name for authentication

- **password**: the password or token for authentication

> **NOTE**
>
> You can use the **stringData** parameter to use clear text content.

Procedure

1. Create a **Secret** object in a YAML file:

   Example secret object

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: secret-basic-auth
   type: kubernetes.io/basic-auth ❶
   data:
   stringData: ❷
     username: admin
     password: <password>
   ```

   ❶ Specifies a basic authentication secret.

   ❷ Specifies the basic authentication values to use.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:

   a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.

b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- Understanding how to create secrets

2.4.2.5. Creating an SSH authentication secret

As an administrator, you can create an SSH authentication secret, which allows you to store data used for SSH authentication. When using this secret type, the **data** parameter of the **Secret** object must contain the SSH credential to use.

Procedure

1. Create a **Secret** object in a YAML file on a control plane node:

   Example secret object

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: secret-ssh-auth
   type: kubernetes.io/ssh-auth ❶
   data:
     ssh-privatekey: | ❷
           MIIEpQIBAAKCAQEAulqb/Y ...
   ```

   ❶ Specifies an SSH authentication secret.

   ❷ Specifies the SSH key/value pair as the SSH credentials to use.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:

   a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.

   b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- Understanding how to create secrets

2.4.2.6. Creating a Docker configuration secret

As an administrator, you can create a Docker configuration secret, which allows you to store the credentials for accessing a container image registry.

- **kubernetes.io/dockercfg**. Use this secret type to store your local Docker configuration file. The **data** parameter of the **secret** object must contain the contents of a **.dockercfg** file encoded in the base64 format.

- **kubernetes.io/dockerconfigjson**. Use this secret type to store your local Docker configuration JSON file. The **data** parameter of the **secret** object must contain the contents of a **.docker/config.json** file encoded in the base64 format.

Procedure

1. Create a **Secret** object in a YAML file.

   Example Docker configuration secret object

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: secret-docker-cfg
     namespace: my-project
   type: kubernetes.io/dockerconfig ❶
   data:

   .dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cgYXV
   0aCBrZXlzCg== ❷
   ```

   ❶ Specifies that the secret is using a Docker configuration file.

   ❷ The output of a base64-encoded Docker configuration file

   Example Docker configuration JSON secret object

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: secret-docker-json
     namespace: my-project
   type: kubernetes.io/dockerconfig ❶
   data:

   .dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cg
   YXV0aCBrZXlzCg== ❷
   ```

   ❶ Specifies that the secret is using a Docker configuration JSONfile.

   ❷ The output of a base64-encoded Docker configuration JSON file

2. Use the following command to create the **Secret** object

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

3. To use the secret in a pod:

a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.

b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- Understanding how to create secrets

2.4.2.7. Creating a secret using the web console

You can create secrets using the web console.

Procedure

1. Navigate to **Workloads → Secrets**.

2. Click **Create → From YAML**.

    a. Edit the YAML manually to your specifications, or drag and drop a file into the YAML editor. For example:

    ```
    apiVersion: v1
    kind: Secret
    metadata:
      name: example
      namespace: <namespace>
    type: Opaque 1
    data:
      username: <base64 encoded username>
      password: <base64 encoded password>
    stringData: 2
      hostname: myapp.mydomain.com
    ```

    1   This example specifies an opaque secret; however, you may see other secret types such as service account token secret, basic authentication secret, SSH authentication secret, or a secret that uses Docker configuration.

    2   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.

3. Click **Create**.

4. Click **Add Secret to workload**

    a. From the drop-down menu, select the workload to add.

    b. Click **Save**.

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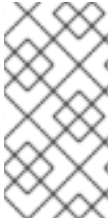
> **NOTE**
>
> Currently, it is not possible to check the resource version of a secret object that was used when a pod was created. It is planned that pods will report this information, so that a controller could restart ones using an old **resourceVersion**. In the interim, do not update the data of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

2.4.4. Creating and using secrets

As an administrator, you can create a service account token secret. This allows you to distribute a service account token to applications that must authenticate to the API.

Procedure

1. Create a service account in your namespace by running the following command:

   ```
   $ oc create sa <service_account_name> -n <your_namespace>
   ```

2. Save the following YAML example to a file named **service-account-token-secret.yaml**. The example includes a **Secret** object configuration that you can use to generate a service account token:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: <secret_name>          1
     annotations:
       kubernetes.io/service-account.name: "sa-name"   2
   type: kubernetes.io/service-account-token            3
   ```

   1 Replace **<secret_name>** with the name of your service token secret.

   2 Specifies an existing service account name. If you are creating both the **ServiceAccount** and the **Secret** objects, create the **ServiceAccount** object first.

   3 Specifies a service account token secret type.

3. Generate the service account token by applying the file:

   ```
   $ oc apply -f service-account-token-secret.yaml
   ```

4. Get the service account token from the secret by running the following command:

```
$ oc get secret <sa_token_secret> -o jsonpath='{.data.token}' | base64 --decode ❶
```

Example output

```
ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpvNnh5YnZhSE9QMkk3YnRZMVZoclFf
QTZfRFp1YlUifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5
pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZX
J2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImJ1aWxkZXItdG9rZW4tdHZrbnIiLCJrdWJlcm5l
dGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiYnVpbGRlciIsImt1
YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjZmZGU
2MGZmLTA1NGYtNDkyZi04YzhjLTNlZjE0NDk3MmFmNyIsInN1YiI6InN5c3RlbTpzZXJ2aWNl
YWNjb3VudDpkZWZhdWx0OmJ1aWxkZXIifQ.OmqFTDuMHC_lYvvEUrjr1x453hlEEHYcxS9VK
SzmRkP1SiVZWPNPkTWlfNRp6bIUZD3U6aN3N7dMSN0el5hu36xPgpKTdvuckKLTCnelMx6c
xOdAbrcw1mCmOClNscwjS1KO1kzMtYnnq8rXHiMJELsNlhnRyyIXRTtNBsy4t64T3283s3SLsa
ncyx0gy0ujx-Ch3uKAKdZi5iT-I8jnnQ-ds5THDs2h65RJhgglQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnvi3DGn2XI-
hZxl1yD2yGH1RBpYUHA
```

❶ Replace <sa_token_secret> with the name of your service token secret.

5. Use your service account token to authenticate with the API of your cluster:

```
$ curl -X GET <openshift_cluster_api> --header "Authorization: Bearer <token>" ❶ ❷
```

❶ Replace **<openshift_cluster_api>** with the OpenShift cluster API.

❷ Replace **<token>** with the service account token that is output in the preceding command.

2.4.5. About using signed certificates with secrets

To secure communication to your service, you can configure OpenShift Dedicated 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 spec configured for a service serving certificates secret.

```
apiVersion: v1
kind: Service
metadata:
  name: registry
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: registry-cert ❶
# ...
```

❶ 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.

2.4.5.1. 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.beta.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** spec for your service.

2. Add the **service.beta.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.beta.openshift.io/serving-cert-secret-name: my-cert ①
   spec:
     selector:
       app: MyApp
     ports:
     - protocol: TCP
       port: 80
       targetPort: 9376
   ```

   ① 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:

   a. View a list of all secrets:

      ```
      $ oc get secrets
      ```

      Example output

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

   b. View details on your secret:

CHAPTER 2. WORKING WITH PODS

```
$ oc describe secret my-cert
```

Example output

```
Name:         my-cert
Namespace:    openshift-console
Labels:       <none>
Annotations:  service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
              service.beta.openshift.io/originating-service-name: my-service
              service.beta.openshift.io/originating-service-uid: 640f0ec3-afc2-4380-bf31-
a8c784846a11
              service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z

Type:  kubernetes.io/tls

Data
====
tls.key:  1679 bytes
tls.crt:  2595 bytes
```

5. Edit your **Pod** spec with that secret.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: my-container
      mountPath: "/etc/my-path"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  volumes:
  - name: my-volume
    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 nears expiration. View the

The certificate/key pair is automatically replaced when it gets close to expiration. View the expiration date in the **service.beta.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.

2.4.6. Troubleshooting secrets

If a service certificate generation fails with (service's **service.beta.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.beta.openshift.io/serving-cert-generation-error**, **service.beta.openshift.io/serving-cert-generation-error-num**:

1. Delete the secret:

   ```
   $ oc delete secret <secret_name>
   ```

2. Clear the annotations:

   ```
   $ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
   ```

   ```
   $ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
   ```

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

2.5. CREATING AND USING CONFIG MAPS

The following sections define config maps and how to create and use them.

2.5.1. Understanding config maps

Many applications require configuration by using some combination of configuration files, command-line arguments, and environment variables. In OpenShift Dedicated, these configuration artifacts are decoupled from image content to keep containerized applications portable.

The **ConfigMap** object provides mechanisms to inject containers with configuration data while keeping containers agnostic of OpenShift Dedicated. A config map can be used to store fine-grained

information like individual properties or coarse-grained information like entire configuration files or JSON blobs.

The **ConfigMap** object holds key-value pairs of configuration data that can be consumed in pods or used to store configuration data for system components such as controllers. For example:

ConfigMap Object Definition

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: my-namespace
data: 1
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw 2
```

1   Contains the configuration data.

2   Points to a file that contains non-UTF8 data, for example, a binary Java keystore file. Enter the file data in Base 64.

> **NOTE**
>
> You can use the **binaryData** field when you create a config map from a binary file, such as an image.

Configuration data can be consumed in pods in a variety of ways. A config map can be used to:

- Populate environment variable values in containers

- Set command-line arguments in a container

- Populate configuration files in a volume

Users and system components can store configuration data in a config map.

A config map is similar to a secret, but designed to more conveniently support working with strings that do not contain sensitive information.

2.5.1.1. Config map restrictions

A config map must be created before its contents can be consumed in pods.

Controllers can be written to tolerate missing configuration data. Consult individual components configured by using config maps on a case-by-case basis.

ConfigMap objects reside in a project.

They can only be referenced by pods in the same project.

The Kubelet only supports the use of a config map for pods it gets from the API server.

This includes any pods created by using the CLI, or indirectly from a replication controller. It does not include pods created by using the OpenShift Dedicated node's **--manifest-url** flag, its **--config** flag, or its REST API because these are not common ways to create pods.

2.5.2. Creating a config map in the OpenShift Dedicated web console

You can create a config map in the OpenShift Dedicated web console.

Procedure

- To create a config map as a cluster administrator:

    1. In the Administrator perspective, select **Workloads → Config Maps**.

    2. At the top right side of the page, select **Create Config Map**.

    3. Enter the contents of your config map.

    4. Select **Create**.

- To create a config map as a developer:

    1. In the Developer perspective, select **Config Maps**.

    2. At the top right side of the page, select **Create Config Map**.

    3. Enter the contents of your config map.

    4. Select **Create**.

2.5.3. Creating a config map by using the CLI

You can use the following command to create a config map from directories, specific files, or literal values.

Procedure

- Create a config map:

    ```
    $ oc create configmap <configmap_name> [options]
    ```

2.5.3.1. Creating a config map from a directory

You can create a config map from a directory by using the **--from-file** flag. This method allows you to use multiple files within a directory to create a config map.

Each file in the directory is used to populate a key in the config map, where the name of the key is the file name, and the value of the key is the content of the file.

For example, the following command creates a config map with the contents of the **example-files** directory:

```
$ oc create configmap game-config --from-file=example-files/
```

View the keys in the config map:

```
$ oc describe configmaps game-config
```

Example output

```
Name:         game-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data

game.properties:     158 bytes
ui.properties:        83 bytes
```

You can see that the two keys in the map are created from the file names in the directory specified in the command. The content of those keys might be large, so the output of **oc describe** only shows the names of the keys and their sizes.

Prerequisite

- You must have a directory with files that contain the data you want to populate a config map with.
  The following procedure uses these example files: **game.properties** and **ui.properties**:

  ```
  $ cat example-files/game.properties
  ```

  Example output

  ```
  enemies=aliens
  lives=3
  enemies.cheat=true
  enemies.cheat.level=noGoodRotten
  secret.code.passphrase=UUDDLRLRBABAS
  secret.code.allowed=true
  secret.code.lives=30
  ```

  ```
  $ cat example-files/ui.properties
  ```

  Example output

  ```
  color.good=purple
  color.bad=yellow
  allow.textmode=true
  how.nice.to.look=fairlyNice
  ```

Procedure

- Create a config map holding the content of each file in this directory by entering the following command:

```
$ oc create configmap game-config \
    --from-file=example-files/
```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys:

```
$ oc get configmaps game-config -o yaml
```

Example output

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"
  selflink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

2.5.3.2. Creating a config map from a file

You can create a config map from a file by using the **--from-file** flag. You can pass the **--from-file** option multiple times to the CLI.

You can also specify the key to set in a config map for content imported from a file by passing a **key=value** expression to the **--from-file** option. For example:

```
$ oc create configmap game-config-3 --from-file=game-special-key=example-files/game.properties
```

> **NOTE**
>
> If you create a config map from a file, you can include files containing non-UTF8 data that are placed in this field without corrupting the non-UTF8 data. OpenShift Dedicated detects binary files and transparently encodes the file as **MIME**. On the server, the **MIME** payload is decoded and stored without corrupting the data.

Prerequisite

- You must have a directory with files that contain the data you want to populate a config map with.
  The following procedure uses these example files: **game.properties** and **ui.properties**:

  ```
  $ cat example-files/game.properties
  ```

Example output

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

Example output

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

Procedure

- Create a config map by specifying a specific file:

  ```
  $ oc create configmap game-config-2 \
      --from-file=example-files/game.properties \
      --from-file=example-files/ui.properties
  ```

- Create a config map by specifying a key-value pair:

  ```
  $ oc create configmap game-config-3 \
      --from-file=game-special-key=example-files/game.properties
  ```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys from the file:

```
$ oc get configmaps game-config-2 -o yaml
```

Example output

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selflink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys from the key–value pair:

```
$ oc get configmaps game-config-3 -o yaml
```

Example output

```
apiVersion: v1
data:
  game-special-key: |-    1
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  selflink: /api/v1/namespaces/default/configmaps/game-config-3
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985
```

1 This is the key that you set in the preceding step.

2.5.3.3. Creating a config map from literal values

You can supply literal values for a config map.

The **--from-literal** option takes a **key=value** syntax, which allows literal values to be supplied directly on the command line.

Procedure

- Create a config map by specifying a literal value:

  ```
  $ oc create configmap special-config \
      --from-literal=special.how=very \
      --from-literal=special.type=charm
  ```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys:

  ```
  $ oc get configmaps special-config -o yaml
  ```

  Example output

  ```
  apiVersion: v1
  data:
    special.how: very
    special.type: charm
  kind: ConfigMap
  metadata:
    creationTimestamp: 2016-02-18T19:14:38Z
    name: special-config
    namespace: default
    resourceVersion: "651"
    selflink: /api/v1/namespaces/default/configmaps/special-config
    uid: dadce046-d673-11e5-8cd0-68f728db1985
  ```

2.5.4. Use cases: Consuming config maps in pods

The following sections describe some uses cases when consuming **ConfigMap** objects in pods.

2.5.4.1. Populating environment variables in containers by using config maps

You can use config maps to populate individual environment variables in containers or to populate environment variables in containers from all keys that form valid environment variable names.

As an example, consider the following config map:

ConfigMap with two environment variables

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: special-config ❶
  namespace: default ❷
data:
  special.how: very ❸
  special.type: charm ❹
```

❶   Name of the config map.

❷   The project in which the config map resides. Config maps can only be referenced by pods in the same project.

❸ ❹ Environment variables to inject.

ConfigMap with one environment variable

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config ❶
  namespace: default
data:
  log_level: INFO ❷
```

❶   Name of the config map.

❷   Environment variable to inject.

Procedure

- You can consume the keys of this **ConfigMap** in a pod using **configMapKeyRef** sections.

  Sample Pod specification configured to inject specific environment variables

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: dapi-test-pod
  spec:
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    containers:
      - name: test-container
        image: gcr.io/google_containers/busybox
        command: [ "/bin/sh", "-c", "env" ]
        env: ❶
          - name: SPECIAL_LEVEL_KEY ❷
            valueFrom:
              configMapKeyRef:
  ```

```
          name: special-config 3
          key: special.how 4
       - name: SPECIAL_TYPE_KEY
         valueFrom:
           configMapKeyRef:
             name: special-config 5
             key: special.type 6
             optional: true 7
       envFrom: 8
        - configMapRef:
            name: env-config 9
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           drop: [ALL]
  restartPolicy: Never
```

1 Stanza to pull the specified environment variables from a **ConfigMap**.

2 Name of a pod environment variable that you are injecting a key's value into.

3 5 Name of the **ConfigMap** to pull specific environment variables from.

4 6 Environment variable to pull from the **ConfigMap**.

7 Makes the environment variable optional. As optional, the pod will be started even if the specified **ConfigMap** and keys do not exist.

8 Stanza to pull all environment variables from a **ConfigMap**.

9 Name of the **ConfigMap** to pull all environment variables from.

When this pod is run, the pod logs will include the following output:

```
SPECIAL_LEVEL_KEY=very
log_level=INFO
```

NOTE

SPECIAL_TYPE_KEY=charm is not listed in the example output because **optional: true** is set.

2.5.4.2. Setting command-line arguments for container commands with config maps

You can use a config map to set the value of the commands or arguments in a container by using the Kubernetes substitution syntax **$(VAR_NAME)**.

As an example, consider the following config map:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
```

```
  namespace: default
data:
  special.how: very
  special.type: charm
```

Procedure

- To inject values into a command in a container, you must consume the keys you want to use as environment variables. Then you can refer to them in a container's command using the **$(VAR_NAME)** syntax.

  Sample pod specification configured to inject specific environment variables

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: dapi-test-pod
  spec:
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    containers:
      - name: test-container
        image: gcr.io/google_containers/busybox
        command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
  ①
        env:
          - name: SPECIAL_LEVEL_KEY
            valueFrom:
              configMapKeyRef:
                name: special-config
                key: special.how
          - name: SPECIAL_TYPE_KEY
            valueFrom:
              configMapKeyRef:
                name: special-config
                key: special.type
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: [ALL]
    restartPolicy: Never
  ```

  ① Inject the values into a command in a container using the keys you want to use as environment variables.

  When this pod is run, the output from the echo command run in the test-container container is as follows:

  ```
  very charm
  ```

2.5.4.3. Injecting content into a volume by using config maps

You can inject content into a volume by using config maps.

Example ConfigMap custom resource (CR)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

Procedure

You have a couple different options for injecting content into a volume by using config maps.

- The most basic way to inject content into a volume by using a config map is to populate the volume with files where the key is the file name and the content of the file is the value of the key:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/special.how" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: config-volume
      configMap:
        name: special-config  1
  restartPolicy: Never
```

  1 File containing key.

  When this pod is run, the output of the cat command will be:

```
very
```

- You can also control the paths within the volume where config map keys are projected:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/path/to/special-key" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
        - key: special.how
          path: path/to/special-key  ❶
  restartPolicy: Never
```

❶   Path to config map key.

When this pod is run, the output of the cat command will be:

```
very
```

2.6. 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, reference a priority class in the pod specification to apply that weight for scheduling.

2.6.1. 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.

2.6.1.1. 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 or equal to one billion for critical pods that must not be preempted or evicted. By default, OpenShift Dedicated has two reserved priority classes for critical system pods to have guaranteed scheduling.

```
$ oc get priorityclasses
```

Example output

```
NAME                    VALUE        GLOBAL-DEFAULT   AGE
system-node-critical    2000001000   false            72m
system-cluster-critical 2000000000   false            72m
openshift-user-critical 1000000000   false            3d13h
cluster-logging         1000000      false            29s
```

- **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 **ovnkube-node**, 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

  - ovn-kubernetes

  - 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

- **openshift-user-critical** – You can use the **priorityClassName** field with important pods that cannot bind their resource consumption and do not have predictable resource consumption behavior. Prometheus pods under the **openshift-monitoring** and **openshift-user-workload-monitoring** namespaces use the **openshift-user-critical priorityClassName**. Monitoring workloads use **system-critical** as their first **priorityClass**, but this causes problems when

monitoring uses excessive memory and the nodes cannot evict them. As a result, monitoring drops priority to give the scheduler flexibility, moving heavy workloads around to keep critical nodes operating.

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

2.6.1.2. Pod priority names

After you have one or more priority classes, you can create pods that specify a priority class name in a **Pod** spec. 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.

2.6.2. 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** spec 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** spec 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.

2.6.2.1. Non-preempting priority classes

Pods with the preemption policy set to **Never** are placed in the scheduling queue ahead of lower-priority pods, but they cannot preempt other pods. A non-preempting pod waiting to be scheduled stays in the scheduling queue until sufficient resources are free and it can be scheduled. Non-preempting pods, like other pods, are subject to scheduler back-off. This means that if the scheduler tries unsuccessfully to schedule these pods, they are retried with lower frequency, allowing other pods with lower priority to be scheduled before them.

Non-preempting pods can still be preempted by other, high-priority pods.

2.6.2.2. 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 Dedicated 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.

2.6.2.3. 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.

2.6.3. Configuring priority and preemption

You apply pod priority and preemption by creating a priority class object and associating pods to the priority by using the **priorityClassName** in your pod specs.
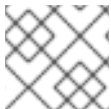
> **NOTE**
>
> You cannot add a priority class directly to an existing scheduled pod.

Procedure

To configure your cluster to use priority and preemption:

1. Define a pod spec to include the name of a priority class by creating a YAML file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: nginx
     labels:
       env: test
   spec:
     containers:
     - name: nginx
       image: nginx
       imagePullPolicy: IfNotPresent
     priorityClassName: system-cluster-critical ❶
   ```

1 Specify the priority class to use with this pod.

2. 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.

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.

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 selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Dedicated schedules the pods on nodes that contain matching labels.

You add labels to a node, a compute machine set, or a machine config. Adding the label to the compute machine set ensures that if the node or machine goes down, new nodes have the label. Labels added to a node or machine config do not persist if the node or machine goes down.

To add node selectors to an existing pod, add a node selector to the controlling object for that pod, such as a **ReplicaSet** object, **DaemonSet** object, **StatefulSet** object, **Deployment** object, or **DeploymentConfig** object. 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. If the pod does not have a controlling object, you must delete the pod, edit the pod spec, and recreate the pod.

> **NOTE**
>
> You cannot add a node selector directly to an existing scheduled pod.

Prerequisites

To add a node selector to existing pods, determine the controlling object for that pod. For example, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** replica set:

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

Example output

```
kind: Pod
apiVersion: v1
metadata:
# ...
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:

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: router-default-66d5cf9464
      uid: d81dd094-da26-11e9-a48a-128e7edf0312
      controller: true
      blockOwnerDeletion: true
# ...
```

Procedure

- Add the matching node selector to a pod:

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

    Example ReplicaSet object with labels

    ```
    kind: ReplicaSet
    apiVersion: apps/v1
    metadata:
      name: hello-node-6fbccf8d9
    # ...
    spec:
    # ...
      template:
        metadata:
          creationTimestamp: null
          labels:
            ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
            pod-template-hash: 66d5cf9464
        spec:
          nodeSelector:
            kubernetes.io/os: linux
            node-role.kubernetes.io/worker: ''
            type: user-node  ❶
    # ...
    ```

    ❶ Add the node selector.

  - To add a node selector to a specific, new pod, add the selector to the **Pod** object directly:

Example **Pod** object with a node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
  nodeSelector:
    region: east
    type: user-node
# ...
```

> **NOTE**
>
> You cannot add a node selector directly to an existing scheduled pod.

CHAPTER 3. AUTOMATICALLY SCALING PODS WITH THE CUSTOM METRICS AUTOSCALER OPERATOR

3.1. RELEASE NOTES

3.1.1. Custom Metrics Autoscaler Operator release notes

The release notes for the Custom Metrics Autoscaler Operator for Red Hat OpenShift describe new features and enhancements, deprecated features, and known issues.

The Custom Metrics Autoscaler Operator uses the Kubernetes-based Event Driven Autoscaler (KEDA) and is built on top of the OpenShift Dedicated horizontal pod autoscaler (HPA).

> **NOTE**
>
> The Custom Metrics Autoscaler Operator for Red Hat OpenShift is provided as an installable component, with a distinct release cycle from the core OpenShift Dedicated. The Red Hat OpenShift Container Platform Life Cycle Policy outlines release compatibility.

3.1.1.1. Supported versions

The following table defines the Custom Metrics Autoscaler Operator versions for each OpenShift Dedicated version.

Version	OpenShift Dedicated version	General availability
2.17.2-2	4.20	General availability
2.17.2-2	4.19	General availability
2.17.2-2	4.18	General availability
2.17.2-2	4.17	General availability
2.17.2-2	4.16	General availability
2.17.2-2	4.15	General availability
2.17.2-2	4.14	General availability
2.17.2-2	4.13	General availability
2.17.2-2	4.12	General availability

3.1.1.2. Custom Metrics Autoscaler Operator 2.17.2-2 release notes

Issued: 21 October 2025

This release of the Custom Metrics Autoscaler Operator 2.17.2-2 is a rebuild of the 2.17.2 version of the Custom Metrics Autoscaler Operator using a newer base image and Go compiler. There are no code changes to the Custom Metrics Autoscaler Operator. The following advisory is available for the Custom Metrics Autoscaler Operator:

- RHBA-2025:18914
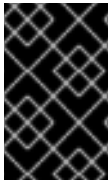
> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2. Release notes for past releases of the Custom Metrics Autoscaler Operator

The following release notes are for previous versions of the Custom Metrics Autoscaler Operator.

For the current version, see Custom Metrics Autoscaler Operator release notes .

3.1.2.1. Custom Metrics Autoscaler Operator 2.17.2 release notes

Issued: 25 September 2025

This release of the Custom Metrics Autoscaler Operator 2.17.2 addresses Common Vulnerabilities and Exposures (CVEs). The following advisory is available for the Custom Metrics Autoscaler Operator:

- RHSA-2025:16124
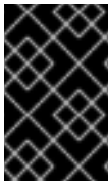
> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.1.1. New features and enhancements

3.1.2.1.1.1. The KEDA controller is automatically created during installation

The KEDA controller is now automatically created when you install the Custom Metrics Autoscaler Operator. Previously, you needed to manually create the KEDA controller. You can edit the automatically-created KEDA controller, as needed.

3.1.2.1.1.2. Support for the Kubernetes workload trigger

The Cluster Metrics Autoscaler Operator now supports using the Kubernetes workload trigger to scale pods based on the number of pods matching a specific label selector.

3.1.2.1.1.3. Support for bound service account tokens

The Cluster Metrics Autoscaler Operator now supports bound service account tokens. Previously, the Operator supported only legacy service account tokens, which are being phased out in favor of bound service account tokens for security reasons.

3.1.2.1.2. Bug fixes

- Previously, the KEDA controller did not support volume mounts. As a result, you could not use Kerberos with the Kafka scaler. With this fix, the KEDA controller now supports volume mounts. (OCPBUGS-42559)

- Previously, the KEDA version in the **keda-operator** deployment object log reported that the Custom Metrics Autoscaler Operator was based on an incorrect KEDA version. With this fix, the correct KEDA version is reported in the log. (OCPBUGS-58129)

Additional resources

- Editing the Keda Controller CR

- Understanding the Kubernetes workload trigger

- Understanding custom metrics autoscaler trigger authentications

3.1.2.2. Custom Metrics Autoscaler Operator 2.15.1-4 release notes

Issued: 31 March 2025

This release of the Custom Metrics Autoscaler Operator 2.15.1-4 addresses Common Vulnerabilities and Exposures (CVEs). The following advisory is available for the Custom Metrics Autoscaler Operator:

- RHSA-2025:3501

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.2.1. New features and enhancements

3.1.2.2.1.1. CMA multi-arch builds

With this version of the Custom Metrics Autoscaler Operator, you can now install and run the Operator on an ARM64 OpenShift Dedicated cluster.

3.1.2.3. Custom Metrics Autoscaler Operator 2.14.1-467 release notes

This release of the Custom Metrics Autoscaler Operator 2.14.1-467 provides a CVE and a bug fix for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHSA-2024:7348.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.3.1. Bug fixes

- Previously, the root file system of the Custom Metrics Autoscaler Operator pod was writable, which is unnecessary and could present security issues. This update makes the pod root file system read-only, which addresses the potential security issue. (OCPBUGS-37989)

3.1.2.4. Custom Metrics Autoscaler Operator 2.14.1-454 release notes

This release of the Custom Metrics Autoscaler Operator 2.14.1-454 provides a CVE, a new feature, and bug fixes for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHBA-2024:5865.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.4.1. New features and enhancements

3.1.2.4.1.1. Support for the Cron trigger with the Custom Metrics Autoscaler Operator

The Custom Metrics Autoscaler Operator can now use the Cron trigger to scale pods based on an hourly schedule. When your specified time frame starts, the Custom Metrics Autoscaler Operator scales pods to your desired amount. When the time frame ends, the Operator scales back down to the previous level.

For more information, see Understanding the Cron trigger.

3.1.2.4.2. Bug fixes

- Previously, if you made changes to audit configuration parameters in the **KedaController** custom resource, the **keda-metrics-server-audit-policy** config map would not get updated. As a consequence, you could not change the audit configuration parameters after the initial deployment of the Custom Metrics Autoscaler. With this fix, changes to the audit configuration now render properly in the config map, allowing you to change the audit configuration any time after installation. (OCPBUGS-32521)

3.1.2.5. Custom Metrics Autoscaler Operator 2.13.1 release notes

This release of the Custom Metrics Autoscaler Operator 2.13.1-421 provides a new feature and a bug fix for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHBA-2024:4837.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.5.1. New features and enhancements

3.1.2.5.1.1. Support for custom certificates with the Custom Metrics Autoscaler Operator

The Custom Metrics Autoscaler Operator can now use custom service CA certificates to connect

securely to TLS-enabled metrics sources, such as an external Kafka cluster or an external Prometheus service. By default, the Operator uses automatically-generated service certificates to connect to on-cluster services only. There is a new field in the **KedaController** object that allows you to load custom server CA certificates for connecting to external services by using config maps.

For more information, see Custom CA certificates for the Custom Metrics Autoscaler .

3.1.2.5.2. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images were missing time zone information. As a consequence, scaled objects with **cron** triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds are updated to include time zone information. As a result, scaled objects containing **cron** triggers now function properly. Scaled objects containing **cron** triggers are currently not supported for the custom metrics autoscaler. (OCPBUGS-34018)

3.1.2.6. Custom Metrics Autoscaler Operator 2.12.1–394 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1–394 provides a bug fix for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHSA-2024:2901.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.6.1. Bug fixes

- Previously, the **protojson.Unmarshal** function entered into an infinite loop when unmarshaling certain forms of invalid JSON. This condition could occur when unmarshaling into a message that contains a **google.protobuf.Any** value or when the **UnmarshalOptions.DiscardUnknown** option is set. This release fixes this issue. (OCPBUGS-30305)

- Previously, when parsing a multipart form, either explicitly with the **Request.ParseMultipartForm** method or implicitly with the **Request.FormValue**, **Request.PostFormValue**, or **Request.FormFile** method, the limits on the total size of the parsed form were not applied to the memory consumed. This could cause memory exhaustion. With this fix, the parsing process now correctly limits the maximum size of form lines while reading a single form line. (OCPBUGS-30360)

- Previously, when following an HTTP redirect to a domain that is not on a matching subdomain or on an exact match of the initial domain, an HTTP client would not forward sensitive headers, such as **Authorization** or **Cookie**. For example, a redirect from **example.com** to **www.example.com** would forward the **Authorization** header, but a redirect to **www.example.org** would not forward the header. This release fixes this issue. (OCPBUGS-30365)

- Previously, verifying a certificate chain that contains a certificate with an unknown public key algorithm caused the certificate verification process to panic. This condition affected all crypto and Transport Layer Security (TLS) clients and servers that set the **Config.ClientAuth**

parameter to the **VerifyClientCertIfGiven** or **RequireAndVerifyClientCert** value. The default behavior is for TLS servers to not verify client certificates. This release fixes this issue. (OCPBUGS-30370)

- Previously, if errors returned from the **MarshalJSON** method contained user-controlled data, an attacker could have used the data to break the contextual auto-escaping behavior of the HTML template package. This condition would allow for subsequent actions to inject unexpected content into the templates. This release fixes this issue. (OCPBUGS-30397)

- Previously, the **net/http** and **golang.org/x/net/http2** Go packages did not limit the number of **CONTINUATION** frames for an HTTP/2 request. This condition could result in excessive CPU consumption. This release fixes this issue. (OCPBUGS-30894)

3.1.2.7. Custom Metrics Autoscaler Operator 2.12.1–384 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1–384 provides a bug fix for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHBA-2024:2043.
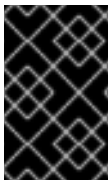
IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.7.1. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images were missing time zone information. As a consequence, scaled objects with **cron** triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds are updated to include time zone information. As a result, scaled objects containing **cron** triggers now function properly. (OCPBUGS-32395)

3.1.2.8. Custom Metrics Autoscaler Operator 2.12.1–376 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1–376 provides security updates and bug fixes for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHSA-2024:1812.
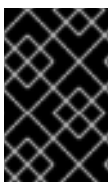
IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.8.1. Bug fixes

- Previously, if invalid values such as nonexistent namespaces were specified in scaled object metadata, the underlying scaler clients would not free, or close, their client descriptors, resulting in a slow memory leak. This fix properly closes the underlying client descriptors when there are errors, preventing memory from leaking. (OCPBUGS-30145)

- Previously the **ServiceMonitor** custom resource (CR) for the **keda-metrics-apiserver** pod was not functioning, because the CR referenced an incorrect metrics port name of **http**. This fix

corrects the **ServiceMonitor** CR to reference the proper port name of **metrics**. As a result, the Service Monitor functions properly. (OCPBUGS-25806)

3.1.2.9. Custom Metrics Autoscaler Operator 2.11.2-322 release notes

This release of the Custom Metrics Autoscaler Operator 2.11.2-322 provides security updates and bug fixes for running the Operator in an OpenShift Dedicated cluster. The following advisory is available for the RHSA-2023:6144.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.9.1. Bug fixes

- Because the Custom Metrics Autoscaler Operator version 3.11.2-311 was released without a required volume mount in the Operator deployment, the Custom Metrics Autoscaler Operator pod would restart every 15 minutes. This fix adds the required volume mount to the Operator deployment. As a result, the Operator no longer restarts every 15 minutes. (OCPBUGS-22361)

3.1.2.10. Custom Metrics Autoscaler Operator 2.11.2-311 release notes

This release of the Custom Metrics Autoscaler Operator 2.11.2-311 provides new features and bug fixes for running the Operator in an OpenShift Dedicated cluster. The components of the Custom Metrics Autoscaler Operator 2.11.2-311 were released in RHBA-2023:5981.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.10.1. New features and enhancements

3.1.2.10.1.1. Red Hat OpenShift Service on AWS and OpenShift Dedicated are now supported

The Custom Metrics Autoscaler Operator 2.11.2-311 can be installed on Red Hat OpenShift Service on AWS and OpenShift Dedicated managed clusters. Previous versions of the Custom Metrics Autoscaler Operator could be installed only in the **openshift-keda** namespace. This prevented the Operator from being installed on Red Hat OpenShift Service on AWS and OpenShift Dedicated clusters. This version of Custom Metrics Autoscaler allows installation to other namespaces such as **openshift-operators** or **keda**, enabling installation into Red Hat OpenShift Service on AWS and OpenShift Dedicated clusters.

3.1.2.10.2. Bug fixes

- Previously, if the Custom Metrics Autoscaler Operator was installed and configured, but not in use, the OpenShift CLI reported the **couldn't get resource list for external.metrics.k8s.io/v1beta1: Got empty response for: external.metrics.k8s.io/v1beta1** error after any **oc** command was entered. The message, although harmless, could have caused confusion. With this fix, the **Got empty response for: external.metrics…** error no longer appears inappropriately. (OCPBUGS-15779)

- Previously, any annotation or label change to objects managed by the Custom Metrics Autoscaler were reverted by Custom Metrics Autoscaler Operator any time the Keda Controller was modified, for example after a configuration change. This caused continuous changing of labels in your objects. The Custom Metrics Autoscaler now uses its own annotation to manage labels and annotations, and annotation or label are no longer inappropriately reverted. (OCPBUGS-15590)

3.1.2.11. Custom Metrics Autoscaler Operator 2.10.1–267 release notes

This release of the Custom Metrics Autoscaler Operator 2.10.1-267 provides new features and bug fixes for running the Operator in an OpenShift Dedicated cluster. The components of the Custom Metrics Autoscaler Operator 2.10.1-267 were released in RHBA-2023:4089.
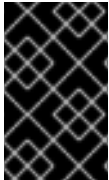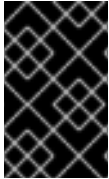
> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.11.1. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images did not contain time zone information. Because of this, scaled objects with cron triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds now include time zone information. As a result, scaled objects containing cron triggers now function properly. (OCPBUGS-15264)

- Previously, the Custom Metrics Autoscaler Operator would attempt to take ownership of all managed objects, including objects in other namespaces and cluster-scoped objects. Because of this, the Custom Metrics Autoscaler Operator was unable to create the role binding for reading the credentials necessary to be an API server. This caused errors in the **kube-system** namespace. With this fix, the Custom Metrics Autoscaler Operator skips adding the **ownerReference** field to any object in another namespace or any cluster-scoped object. As a result, the role binding is now created without any errors. (OCPBUGS-15038)

- Previously, the Custom Metrics Autoscaler Operator added an **ownerReferences** field to the **openshift-keda** namespace. While this did not cause functionality problems, the presence of this field could have caused confusion for cluster administrators. With this fix, the Custom Metrics Autoscaler Operator does not add the **ownerReference** field to the **openshift-keda** namespace. As a result, the **openshift-keda** namespace no longer has a superfluous **ownerReference** field. (OCPBUGS-15293)

- Previously, if you used a Prometheus trigger configured with authentication method other than pod identity, and the **podIdentity** parameter was set to **none**, the trigger would fail to scale. With this fix, the Custom Metrics Autoscaler for OpenShift now properly handles the **none** pod identity provider type. As a result, a Prometheus trigger configured with authentication method other than pod identity, and the **podIdentity** parameter sset to **none** now properly scales. (OCPBUGS-15274)

3.1.2.12. Custom Metrics Autoscaler Operator 2.10.1 release notes

This release of the Custom Metrics Autoscaler Operator 2.10.1 provides new features and bug fixes for running the Operator in an OpenShift Dedicated cluster. The components of the Custom Metrics Autoscaler Operator 2.10.1 were released in RHEA-2023:3199.

> **IMPORTANT**
>
> Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.12.1. New features and enhancements

3.1.2.12.1.1. Custom Metrics Autoscaler Operator general availability

The Custom Metrics Autoscaler Operator is now generally available as of Custom Metrics Autoscaler Operator version 2.10.1.

> **IMPORTANT**
>
> Scaling by using a scaled job is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in 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 about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

3.1.2.12.1.2. Performance metrics

You can now use the Prometheus Query Language (PromQL) to query metrics on the Custom Metrics Autoscaler Operator.

3.1.2.12.1.3. Pausing the custom metrics autoscaling for scaled objects

You can now pause the autoscaling of a scaled object, as needed, and resume autoscaling when ready.

3.1.2.12.1.4. Replica fall back for scaled objects

You can now specify the number of replicas to fall back to if a scaled object fails to get metrics from the source.

3.1.2.12.1.5. Customizable HPA naming for scaled objects

You can now specify a custom name for the horizontal pod autoscaler in scaled objects.

3.1.2.12.1.6. Activation and scaling thresholds

Because the horizontal pod autoscaler (HPA) cannot scale to or from 0 replicas, the Custom Metrics Autoscaler Operator does that scaling, after which the HPA performs the scaling. You can now specify when the HPA takes over autoscaling, based on the number of replicas. This allows for more flexibility with your scaling policies.

3.1.2.13. Custom Metrics Autoscaler Operator 2.8.2-174 release notes

This release of the Custom Metrics Autoscaler Operator 2.8.2-174 provides new features and bug fixes for running the Operator in an OpenShift Dedicated cluster. The components of the Custom Metrics Autoscaler Operator 2.8.2-174 were released in RHEA-2023:1683.
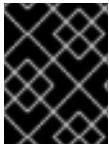
> **IMPORTANT**
>
> The Custom Metrics Autoscaler Operator version 2.8.2-174 is a Technology Preview feature.

3.1.2.13.1. New features and enhancements

3.1.2.13.1.1. Operator upgrade support

You can now upgrade from a prior version of the Custom Metrics Autoscaler Operator. See "Changing the update channel for an Operator" in the "Additional resources" for information on upgrading an Operator.

3.1.2.13.1.2. must-gather support

You can now collect data about the Custom Metrics Autoscaler Operator and its components by using the OpenShift Dedicated **must-gather** tool. Currently, the process for using the **must-gather** tool with the Custom Metrics Autoscaler is different than for other operators. See "Gathering debugging data in the "Additional resources" for more information.

3.1.2.14. Custom Metrics Autoscaler Operator 2.8.2 release notes

This release of the Custom Metrics Autoscaler Operator 2.8.2 provides new features and bug fixes for running the Operator in an OpenShift Dedicated cluster. The components of the Custom Metrics Autoscaler Operator 2.8.2 were released in RHSA-2023:1042.
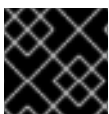
> **IMPORTANT**
>
> The Custom Metrics Autoscaler Operator version 2.8.2 is a Technology Preview feature.

3.1.2.14.1. New features and enhancements

3.1.2.14.1.1. Audit Logging

You can now gather and view audit logs for the Custom Metrics Autoscaler Operator and its associated components. Audit logs are security-relevant chronological sets of records that document the sequence of activities that have affected the system by individual users, administrators, or other components of the system.

3.1.2.14.1.2. Scale applications based on Apache Kafka metrics

You can now use the KEDA Apache kafka trigger/scaler to scale deployments based on an Apache Kafka topic.

3.1.2.14.1.3. Scale applications based on CPU metrics

You can now use the KEDA CPU trigger/scaler to scale deployments based on CPU metrics.

3.1.2.14.1.4. Scale applications based on memory metrics

You can now use the KEDA memory trigger/scaler to scale deployments based on memory metrics.

3.2. CUSTOM METRICS AUTOSCALER OPERATOR OVERVIEW

As a developer, you can use Custom Metrics Autoscaler Operator for Red Hat OpenShift to specify how OpenShift Dedicated should automatically increase or decrease the number of pods for a deployment, stateful set, custom resource, or job based on custom metrics that are not based only on CPU or memory.

The Custom Metrics Autoscaler Operator is an optional Operator, based on the Kubernetes Event Driven Autoscaler (KEDA), that allows workloads to be scaled using additional metrics sources other than pod metrics.

The custom metrics autoscaler currently supports only the Prometheus, CPU, memory, and Apache Kafka metrics.

The Custom Metrics Autoscaler Operator scales your pods up and down based on custom, external metrics from specific applications. Your other applications continue to use other scaling methods. You configure *triggers*, also known as scalers, which are the source of events and metrics that the custom metrics autoscaler uses to determine how to scale. The custom metrics autoscaler uses a metrics API to convert the external metrics to a form that OpenShift Dedicated can use. The custom metrics autoscaler creates a horizontal pod autoscaler (HPA) that performs the actual scaling.

To use the custom metrics autoscaler, you create a **ScaledObject** or **ScaledJob** object for a workload, which is a custom resource (CR) that defines the scaling metadata. You specify the deployment or job to scale, the source of the metrics to scale on (trigger), and other parameters such as the minimum and maximum replica counts allowed.

> **NOTE**
>
> You can create only one scaled object or scaled job for each workload that you want to scale. Also, you cannot use a scaled object or scaled job and the horizontal pod autoscaler (HPA) on the same workload.

The custom metrics autoscaler, unlike the HPA, can scale to zero. If you set the **minReplicaCount** value in the custom metrics autoscaler CR to **0**, the custom metrics autoscaler scales the workload down from 1 to 0 replicas to or up from 0 replicas to 1. This is known as the *activation phase*. After scaling up to 1 replica, the HPA takes control of the scaling. This is known as the *scaling phase*.

Some triggers allow you to change the number of replicas that are scaled by the cluster metrics autoscaler. In all cases, the parameter to configure the activation phase always uses the same phrase, prefixed with *activation*. For example, if the **threshold** parameter configures scaling, **activationThreshold** would configure activation. Configuring the activation and scaling phases allows you more flexibility with your scaling policies. For example, you can configure a higher activation phase to prevent scaling up or down if the metric is particularly low.

The activation value has more priority than the scaling value in case of different decisions for each. For example, if the **threshold** is set to **10**, and the **activationThreshold** is **50**, if the metric reports **40**, the scaler is not active and the pods are scaled to zero even if the HPA requires 4 instances.

Figure 3.1. Custom metrics autoscaler workflow
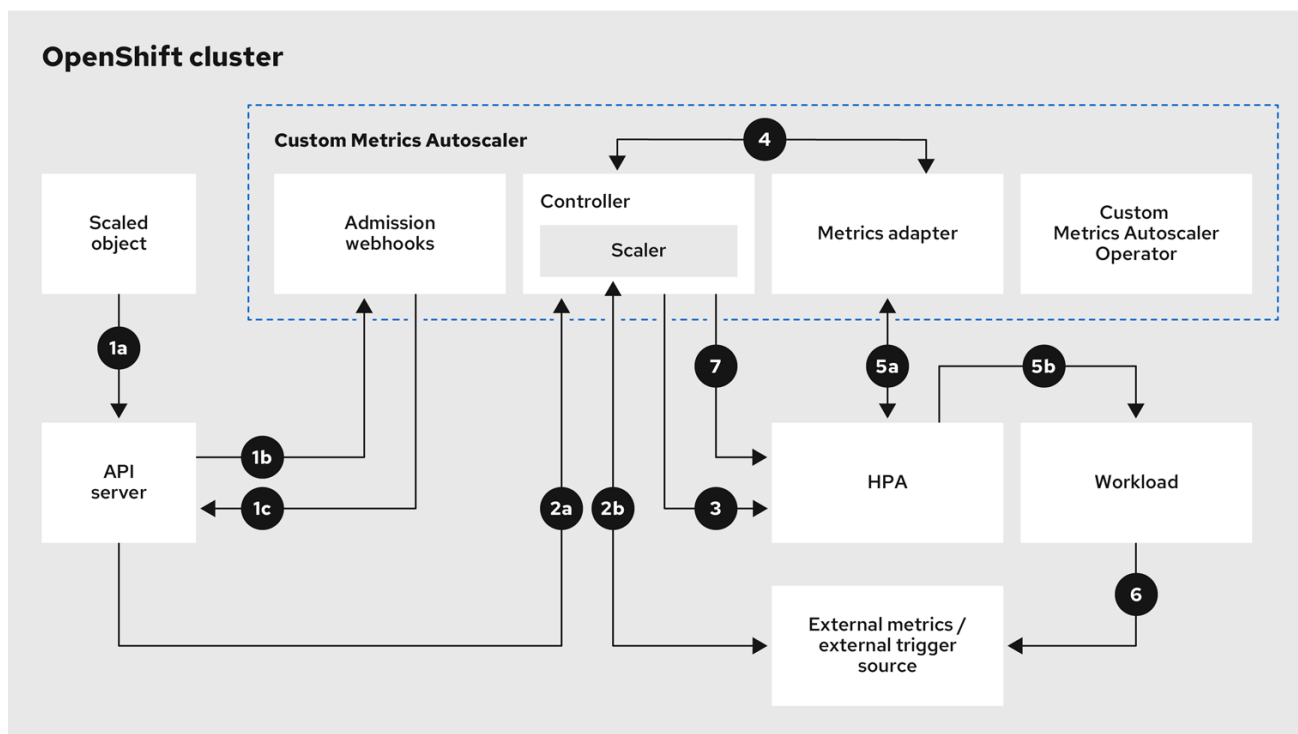
1. You create or modify a scaled object custom resource for a workload on a cluster. The object contains the scaling configuration for that workload. Prior to accepting the new object, the OpenShift API server sends it to the custom metrics autoscaler admission webhooks process to ensure that the object is valid. If validation succeeds, the API server persists the object.

2. The custom metrics autoscaler controller watches for new or modified scaled objects. When the OpenShift API server notifies the controller of a change, the controller monitors any external trigger sources, also known as data sources, that are specified in the object for changes to the metrics data. One or more scalers request scaling data from the external trigger source. For example, for a Kafka trigger type, the controller uses the Kafka scaler to communicate with a Kafka instance to obtain the data requested by the trigger.

3. The controller creates a horizontal pod autoscaler object for the scaled object. As a result, the Horizontal Pod Autoscaler (HPA) Operator starts monitoring the scaling data associated with the trigger. The HPA requests scaling data from the cluster OpenShift API server endpoint.

4. The OpenShift API server endpoint is served by the custom metrics autoscaler metrics adapter. When the metrics adapter receives a request for custom metrics, it uses a GRPC connection to the controller to request it for the most recent trigger data received from the scaler.

5. The HPA makes scaling decisions based upon the data received from the metrics adapter and scales the workload up or down by increasing or decreasing the replicas.

6. As a it operates, a workload can affect the scaling metrics. For example, if a workload is scaled up to handle work in a Kafka queue, the queue size decreases after the workload processes all the work. As a result, the workload is scaled down.

7. If the metrics are in a range specified by the **minReplicaCount** value, the custom metrics autoscaler controller disables all scaling, and leaves the replica count at a fixed level. If the metrics exceed that range, the custom metrics autoscaler controller enables scaling and allows the HPA to scale the workload. While scaling is disabled, the HPA does not take any action.

3.2.1. Custom CA certificates for the Custom Metrics Autoscaler

By default, the Custom Metrics Autoscaler Operator uses automatically-generated service CA certificates to connect to on-cluster services.

If you want to use off-cluster services that require custom CA certificates, you can add the required certificates to a config map. Then, add the config map to the **KedaController** custom resource as described in Installing the custom metrics autoscaler. The Operator loads those certificates on start-up and registers them as trusted by the Operator.

The config maps can contain one or more certificate files that contain one or more PEM-encoded CA certificates. Or, you can use separate config maps for each certificate file.

> **NOTE**
>
> If you later update the config map to add additional certificates, you must restart the **keda-operator-*** pod for the changes to take effect.

3.3. INSTALLING THE CUSTOM METRICS AUTOSCALER

You can use the OpenShift Dedicated web console to install the Custom Metrics Autoscaler Operator.

The installation creates the following five CRDs:

- **ClusterTriggerAuthentication**

- **KedaController**

- **ScaledJob**

- **ScaledObject**

- **TriggerAuthentication**

The installation process also creates the **KedaController** custom resource (CR). You can modify the default **KedaController** CR, if needed. For more information, see "Editing the Keda Controller CR".

> **NOTE**
>
> If you are installing a Custom Metrics Autoscaler Operator version lower than 2.17.2, you must manually create the Keda Controller CR. You can use the procedure described in "Editing the Keda Controller CR" to create the CR.

3.3.1. Installing the custom metrics autoscaler

You can use the following procedure to install the Custom Metrics Autoscaler Operator.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
  If your OpenShift Dedicated cluster is in a cloud account that is owned by Red Hat (non-CCS), you must request **cluster-admin** privileges.

- Remove any previously-installed Technology Preview versions of the Cluster Metrics Autoscaler Operator.

- Remove any versions of the community-based KEDA.
  Also, remove the KEDA 1.x custom resource definitions by running the following commands:

  ```
  $ oc delete crd scaledobjects.keda.k8s.io
  ```

  ```
  $ oc delete crd triggerauthentications.keda.k8s.io
  ```

- Ensure that the **keda** namespace exists. If not, you must manaully create the **keda** namespace.

- Optional: If you need the Custom Metrics Autoscaler Operator to connect to off-cluster services, such as an external Kafka cluster or an external Prometheus service, put any required service CA certificates into a config map. The config map must exist in the same namespace where the Operator is installed. For example:

  ```
  $ oc create configmap -n openshift-keda thanos-cert  --from-file=ca-cert.pem
  ```

Procedure

1. In the OpenShift Dedicated web console, click **Ecosystem → Software Catalog**.

2. Choose **Custom Metrics Autoscaler** from the list of available Operators, and click **Install**.

3. On the **Install Operator** page, ensure that the **A specific namespace on the cluster**option is selected for **Installation Mode**.

4. For **Installed Namespace**, click **Select a namespace**.

5. Click **Select Project**:

   - If the **keda** namespace exists, select **keda** from the list.

   - If the **keda** namespace does not exist:

     a. Select **Create Project** to open the **Create Project** window.

     b. In the **Name** field, enter **keda**.

     c. In the **Display Name** field, enter a descriptive name, such as **keda**.

     d. Optional: In the **Display Name** field, add a description for the namespace.

     e. Click **Create**.

6. Click **Install**.

7. Verify the installation by listing the Custom Metrics Autoscaler Operator components:

   a. Navigate to **Workloads → Pods**.

   b. Select the **keda** project from the drop-down menu and verify that the **custom-metrics-autoscaler-operator-*** pod is running.

   c. Navigate to **Workloads → Deployments** to verify that the **custom-metrics-autoscaler-operator** deployment is running.

8. Optional: Verify the installation in the OpenShift CLI using the following command:

```
$ oc get all -n keda
```

The output appears similar to the following:

Example output

```
NAME                                                    READY   STATUS    RESTARTS   AGE
pod/custom-metrics-autoscaler-operator-5fd8d9ffd8-xt4xp   1/1   Running   0          18m

NAME                                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/custom-metrics-autoscaler-operator   1/1   1            1           18m

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/custom-metrics-autoscaler-operator-5fd8d9ffd8   1     1         1       18m
```

3.3.2. Editing the Keda Controller CR

You can use the following procedure to modify the **KedaController** custom resource (CR), which is automatically installed during the installation of the Custom Metrics Autoscaler Operator.

Procedure

1. In the OpenShift Dedicated web console, click **Ecosystem → Installed Operators**.

2. Click **Custom Metrics Autoscaler**.

3. On the **Operator Details** page, click the **KedaController** tab.

4. On the **KedaController** tab, click **Create KedaController** and edit the file.

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  watchNamespace: ''         1
  operator:
    logLevel: info           2
    logEncoder: console       3
    caConfigMaps:            4
    - thanos-cert
    - kafka-cert
    volumeMounts:            5
    - mountPath: /<path_to_directory>
      name: <name>
    volumes:                6
    - name: <volume_name>
      emptyDir:
```

```
        medium: Memory
    metricsServer:
      logLevel: '0'  7
      auditConfig:  8
        logFormat: "json"
        logOutputVolumeClaim: "persistentVolumeClaimName"
        policy:
          rules:
          - level: Metadata
          omitStages: ["RequestReceived"]
          omitManagedFields: false
        lifetime:
          maxAge: "2"
          maxBackup: "1"
          maxSize: "50"
    serviceAccount: {}
```

1 Specifies a single namespace in which the Custom Metrics Autoscaler Operator scales applications. Leave it blank or leave it empty to scale applications in all namespaces. This field should have a namespace or be empty. The default value is empty.

2 Specifies the level of verbosity for the Custom Metrics Autoscaler Operator log messages. The allowed values are **debug**, **info**, **error**. The default is **info**.

3 Specifies the logging format for the Custom Metrics Autoscaler Operator log messages. The allowed values are **console** or **json**. The default is **console**.

4 Optional: Specifies one or more config maps with CA certificates, which the Custom Metrics Autoscaler Operator can use to connect securely to TLS-enabled metrics sources.

5 Optional: Add the container mount path.

6 Optional: Add a **volumes** block to list each projected volume source.

7 Specifies the logging level for the Custom Metrics Autoscaler Metrics Server. The allowed values are **0** for **info** and **4** for **debug**. The default is **0**.

8 Activates audit logging for the Custom Metrics Autoscaler Operator and specifies the audit policy to use, as described in the "Configuring audit logging" section.

5. Click **Save** to save the changes.

3.4. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGERS

Triggers, also known as scalers, provide the metrics that the Custom Metrics Autoscaler Operator uses to scale your pods.

The custom metrics autoscaler currently supports the Prometheus, CPU, memory, Apache Kafka, and cron triggers.

You use a **ScaledObject** or **ScaledJob** custom resource to configure triggers for specific objects, as described in the sections that follow.

You can configure a certificate authority to use with your scaled objects or for all scalers in the cluster .

3.4.1. Understanding the Prometheus trigger

You can scale pods based on Prometheus metrics, which can use the installed OpenShift Dedicated monitoring or an external Prometheus server as the metrics source. See "Configuring the custom metrics autoscaler to use OpenShift Dedicated monitoring" for information on the configurations required to use the OpenShift Dedicated monitoring as a source for metrics.

> **NOTE**
>
> If Prometheus is collecting metrics from the application that the custom metrics autoscaler is scaling, do not set the minimum replicas to **0** in the custom resource. If there are no application pods, the custom metrics autoscaler does not have any metrics to scale on.

Example scaled object with a Prometheus target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prom-scaledobject
  namespace: my-namespace
spec:
# ...
  triggers:
  - type: prometheus 1
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092 2
      namespace: kedatest 3
      metricName: http_requests_total 4
      threshold: '5' 5
      query: sum(rate(http_requests_total{job="test-app"}[1m])) 6
      authModes: basic 7
      cortexOrgID: my-org 8
      ignoreNullValues: "false" 9
      unsafeSsl: "false" 10
      timeout: 1000 11
```

1 Specifies Prometheus as the trigger type.

2 Specifies the address of the Prometheus server. This example uses OpenShift Dedicated monitoring.

3 Optional: Specifies the namespace of the object you want to scale. This parameter is mandatory if using OpenShift Dedicated monitoring as a source for the metrics.

4 Specifies the name to identify the metric in the **external.metrics.k8s.io** API. If you are using more than one trigger, all metric names must be unique.

5 Specifies the value that triggers scaling. Must be specified as a quoted string value.

6 Specifies the Prometheus query to use.

7 Specifies the authentication method to use. Prometheus scalers support bearer authentication (**bearer**), basic authentication (**basic**), or TLS authentication (**tls**). You configure the specific

authentication parameters in a trigger authentication, as discussed in a following section. As needed, you can also use a secret.

8 Optional: Passes the **X-Scope-OrgID** header to multi-tenant Cortex or Mimir storage for Prometheus. This parameter is required only with multi-tenant Prometheus storage, to indicate which data Prometheus should return.

9 Optional: Specifies how the trigger should proceed if the Prometheus target is lost.

- If **true**, the trigger continues to operate if the Prometheus target is lost. This is the default behavior.

- If **false**, the trigger returns an error if the Prometheus target is lost.

10 Optional: Specifies whether the certificate check should be skipped. For example, you might skip the check if you are running in a test environment and using self-signed certificates at the Prometheus endpoint.

- If **false**, the certificate check is performed. This is the default behavior.

- If **true**, the certificate check is not performed.

> **IMPORTANT**
>
> Skipping the check is not recommended.

11 Optional: Specifies an HTTP request timeout in milliseconds for the HTTP client used by this Prometheus trigger. This value overrides any global timeout setting.

3.4.1.1. Configuring GPU-based autoscaling with Prometheus and DCGM metrics

You can use the Custom Metrics Autoscaler with NVIDIA Data Center GPU Manager (DCGM) metrics to scale workloads based on GPU utilization. This is particularly useful for AI and machine learning workloads that require GPU resources.

Example scaled object with a Prometheus target for GPU-based autoscaling

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: gpu-scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    kind: Deployment
    name: gpu-deployment
  minReplicaCount: 1 1
  maxReplicaCount: 5 2
  triggers:
  - type: prometheus
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: my-namespace
      metricName: gpu_utilization
```

```
    threshold: '90' 3
    query: SUM(DCGM_FI_DEV_GPU_UTIL{instance=~".+", gpu=~".+"}) 4
    authModes: bearer
  authenticationRef:
    name: keda-trigger-auth-prometheus
```

1 Specifies the minimum number of replicas to maintain. For GPU workloads, this should not be set to **0** to ensure that metrics continue to be collected.

2 Specifies the maximum number of replicas allowed during scale-up operations.

3 Specifies the GPU utilization percentage threshold that triggers scaling. When the average GPU utilization exceeds 90%, the autoscaler scales up the deployment.

4 Specifies a Prometheus query using NVIDIA DCGM metrics to monitor GPU utilization across all GPU devices. The **DCGM_FI_DEV_GPU_UTIL** metric provides GPU utilization percentages.

3.4.1.2. Configuring the custom metrics autoscaler to use OpenShift Dedicated monitoring

You can use the installed OpenShift Dedicated Prometheus monitoring as a source for the metrics used by the custom metrics autoscaler. However, there are some additional configurations you must perform.

For your scaled objects to be able to read the OpenShift Dedicated Prometheus metrics, you must use a trigger authentication or a cluster trigger authentication in order to provide the authentication information required. The following procedure differs depending on which trigger authentication method you use. For more information on trigger authentications, see "Understanding custom metrics autoscaler trigger authentications".

> **NOTE**
>
> These steps are not required for an external Prometheus source.

You must perform the following tasks, as described in this section:

- Create a service account.

- Create the trigger authentication.

- Create a role.

- Add that role to the service account.

- Reference the token in the trigger authentication object used by Prometheus.

Prerequisites

- OpenShift Dedicated monitoring must be installed.

- Monitoring of user-defined workloads must be enabled in OpenShift Dedicated monitoring, as described in the **Creating a user-defined workload monitoring config map** section.

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. Change to the appropriate project:

```
$ oc project <project_name> 1
```

1 Specifies one of the following projects:

- If you are using a trigger authentication, specify the project with the object you want to scale.

- If you are using a cluster trigger authentication, specify the **openshift-keda** project.

2. Create a service account if your cluster does not have one:

   a. Create a **service account** object by using the following command:

   ```
   $ oc create serviceaccount thanos 1
   ```

   1 Specifies the name of the service account.

3. Create a trigger authentication with the service account token:

   a. Create a YAML file similar to the following:

   ```
   apiVersion: keda.sh/v1alpha1
   kind: <authentication_method> 1
   metadata:
     name: keda-trigger-auth-prometheus
   spec:
     boundServiceAccountToken: 2
       - parameter: bearerToken 3
         serviceAccountName: thanos 4
   ```

   1 Specifies one of the following trigger authentication methods:

   - If you are using a trigger authentication, specify **TriggerAuthentication**. This example configures a trigger authentication.

   - If you are using a cluster trigger authentication, specify **ClusterTriggerAuthentication**.

   2 Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.

   3 Specifies the authentication parameter to supply by using the token. Here, the example uses bearer authentication.

   4 Specifies the name of the service account to use.

   b. Create the CR object:

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

4. Create a role for reading Thanos metrics:

    a. Create a YAML file with the following parameters:

    ```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: Role
    metadata:
      name: thanos-metrics-reader
    rules:
    - apiGroups:
      - ""
      resources:
      - pods
      verbs:
      - get
    - apiGroups:
      - metrics.k8s.io
      resources:
      - pods
      - nodes
      verbs:
      - get
      - list
      - watch
    ```

    b. Create the CR object:

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

5. Create a role binding for reading Thanos metrics:

    a. Create a YAML file similar to the following:

    ```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: <binding_type>          1
    metadata:
      name: thanos-metrics-reader 2
      namespace: my-project       3
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: Role
      name: thanos-metrics-reader
    subjects:
    - kind: ServiceAccount
      name: thanos               4
      namespace: <namespace_name> 5
    ```

    1   Specifies one of the following object types:

    - If you are using a trigger authentication, specify **RoleBinding**.

    - If you are using a cluster trigger authentication, specify **ClusterRoleBinding**.

    2   Specifies the name of the role you created.

3 Specifies one of the following projects:

- If you are using a trigger authentication, specify the project with the object you want to scale.

- If you are using a cluster trigger authentication, specify the **openshift-keda** project.

4 Specifies the name of the service account to bind to the role.

5 Specifies the project where you previously created the service account.

b. Create the CR object:

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

You can now deploy a scaled object or scaled job to enable autoscaling for your application, as described in "Understanding how to add custom metrics autoscalers". To use OpenShift Dedicated monitoring as the source, in the trigger, or scaler, you must include the following parameters:

- **triggers.type** must be **prometheus**

- **triggers.metadata.serverAddress** must be **https://thanos-querier.openshift-monitoring.svc.cluster.local:9092**

- **triggers.metadata.authModes** must be **bearer**

- **triggers.metadata.namespace** must be set to the namespace of the object to scale

- **triggers.authenticationRef** must point to the trigger authentication resource specified in the previous step

Additional resources

- [Understanding custom metrics autoscaler trigger authentications](#)

3.4.2. Understanding the CPU trigger

You can scale pods based on CPU metrics. This trigger uses cluster metrics as the source for metrics.

The custom metrics autoscaler scales the pods associated with an object to maintain the CPU usage that you specify. The autoscaler increases or decreases the number of replicas between the minimum and maximum numbers to maintain the specified CPU utilization across all pods. The memory trigger considers the memory utilization of the entire pod. If the pod has multiple containers, the memory trigger considers the total memory utilization of all containers in the pod.

> **NOTE**
>
> - This trigger cannot be used with the **ScaledJob** custom resource.
>
> - When using a memory trigger to scale an object, the object does not scale to **0**, even if you are using multiple triggers.

Example scaled object with a CPU target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: my-namespace
spec:
# ...
  triggers:
  - type: cpu 1
    metricType: Utilization 2
    metadata:
      value: '60' 3
  minReplicaCount: 1 4
```

1   Specifies CPU as the trigger type.

2   Specifies the type of metric to use, either **Utilization** or **AverageValue**.

3   Specifies the value that triggers scaling. Must be specified as a quoted string value.

- When using **Utilization**, the target value is the average of the resource metrics across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- When using **AverageValue**, the target value is the average of the metrics across all relevant pods.

4   Specifies the minimum number of replicas when scaling down. For a CPU trigger, enter a value of **1** or greater, because the HPA cannot scale to zero if you are using only CPU metrics.

3.4.3. Understanding the memory trigger

You can scale pods based on memory metrics. This trigger uses cluster metrics as the source for metrics.

The custom metrics autoscaler scales the pods associated with an object to maintain the average memory usage that you specify. The autoscaler increases and decreases the number of replicas between the minimum and maximum numbers to maintain the specified memory utilization across all pods. The memory trigger considers the memory utilization of entire pod. If the pod has multiple containers, the memory utilization is the sum of all of the containers.

> **NOTE**
>
> - This trigger cannot be used with the **ScaledJob** custom resource.
>
> - When using a memory trigger to scale an object, the object does not scale to **0**, even if you are using multiple triggers.

Example scaled object with a memory target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
```

```
    name: memory-scaledobject
    namespace: my-namespace
spec:
# ...
  triggers:
  - type: memory 1
    metricType: Utilization 2
    metadata:
      value: '60' 3
      containerName: api 4
```

1   Specifies memory as the trigger type.

2   Specifies the type of metric to use, either **Utilization** or **AverageValue**.

3   Specifies the value that triggers scaling. Must be specified as a quoted string value.

- When using **Utilization**, the target value is the average of the resource metrics across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- When using **AverageValue**, the target value is the average of the metrics across all relevant pods.

4   Optional: Specifies an individual container to scale, based on the memory utilization of only that container, rather than the entire pod. In this example, only the container named **api** is to be scaled.

3.4.4. Understanding the Kafka trigger

You can scale pods based on an Apache Kafka topic or other services that support the Kafka protocol. The custom metrics autoscaler does not scale higher than the number of Kafka partitions, unless you set the **allowIdleConsumers** parameter to **true** in the scaled object or scaled job.

> **NOTE**
>
> If the number of consumer groups exceeds the number of partitions in a topic, the extra consumer groups remain idle. To avoid this, by default the number of replicas does not exceed:
>
> - The number of partitions on a topic, if a topic is specified
>
> - The number of partitions of all topics in the consumer group, if no topic is specified
>
> - The **maxReplicaCount** specified in scaled object or scaled job CR
>
> You can use the **allowIdleConsumers** parameter to disable these default behaviors.

Example scaled object with a Kafka target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
```

```
    name: kafka-scaledobject
    namespace: my-namespace
  spec:
  # ...
    triggers:
    - type: kafka 1
      metadata:
        topic: my-topic 2
        bootstrapServers: my-cluster-kafka-bootstrap.openshift-operators.svc:9092 3
        consumerGroup: my-group 4
        lagThreshold: '10' 5
        activationLagThreshold: '5' 6
        offsetResetPolicy: latest 7
        allowIdleConsumers: true 8
        scaleToZeroOnInvalidOffset: false 9
        excludePersistentLag: false 10
        version: '1.0.0' 11
        partitionLimitation: '1,2,10-20,31' 12
        tls: enable 13
```

1   Specifies Kafka as the trigger type.

2   Specifies the name of the Kafka topic on which Kafka is processing the offset lag.

3   Specifies a comma-separated list of Kafka brokers to connect to.

4   Specifies the name of the Kafka consumer group used for checking the offset on the topic and processing the related lag.

5   Optional: Specifies the average target value that triggers scaling. Must be specified as a quoted string value. The default is **5**.

6   Optional: Specifies the target value for the activation phase. Must be specified as a quoted string value.

7   Optional: Specifies the Kafka offset reset policy for the Kafka consumer. The available values are: **latest** and **earliest**. The default is **latest**.

8   Optional: Specifies whether the number of Kafka replicas can exceed the number of partitions on a topic.

- If **true**, the number of Kafka replicas can exceed the number of partitions on a topic. This allows for idle Kafka consumers.

- If **false**, the number of Kafka replicas cannot exceed the number of partitions on a topic. This is the default.

9   Specifies how the trigger behaves when a Kafka partition does not have a valid offset.

- If **true**, the consumers are scaled to zero for that partition.

- If **false**, the scaler keeps a single consumer for that partition. This is the default.

10   Optional: Specifies whether the trigger includes or excludes partition lag for partitions whose current offset is the same as the current offset of the previous polling cycle.

- If **true**, the scaler excludes partition lag in these partitions.

- If **false**, the trigger includes all consumer lag in all partitions. This is the default.

11 Optional: Specifies the version of your Kafka brokers. Must be specified as a quoted string value. The default is **1.0.0**.

12 Optional: Specifies a comma-separated list of partition IDs to scope the scaling on. If set, only the listed IDs are considered when calculating lag. Must be specified as a quoted string value. The default is to consider all partitions.

13 Optional: Specifies whether to use TSL client authentication for Kafka. The default is **disable**. For information on configuring TLS, see "Understanding custom metrics autoscaler trigger authentications".

3.4.5. Understanding the Cron trigger

You can scale pods based on a time range.

When the time range starts, the custom metrics autoscaler scales the pods associated with an object from the configured minimum number of pods to the specified number of desired pods. At the end of the time range, the pods are scaled back to the configured minimum. The time period must be configured in cron format.

The following example scales the pods associated with this scaled object from **0** to **100** from 6:00 AM to 6:30 PM India Standard Time.

Example scaled object with a Cron trigger

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cron-scaledobject
  namespace: default
spec:
  scaleTargetRef:
    name: my-deployment
  minReplicaCount: 0     1
  maxReplicaCount: 100   2
  cooldownPeriod: 300
  triggers:
  - type: cron           3
    metadata:
      timezone: Asia/Kolkata   4
      start: "0 6 * * *"       5
      end: "30 18 * * *"       6
      desiredReplicas: "100"   7
```

1 Specifies the minimum number of pods to scale down to at the end of the time frame.

2 Specifies the maximum number of replicas when scaling up. This value should be the same as **desiredReplicas**. The default is **100**.

3 Specifies a Cron trigger.

4 Specifies the timezone for the time frame. This value must be from the [IANA Time Zone Database](#).

5 Specifies the start of the time frame.

6 Specifies the end of the time frame.

7 Specifies the number of pods to scale to between the start and end of the time frame. This value should be the same as **maxReplicaCount**.

3.4.6. Understanding the Kubernetes workload trigger

You can scale pods based on the number of pods matching a specific label selector.

The Custom Metrics Autoscaler Operator tracks the number of pods with a specific label that are in the same namespace, then calculates a *relation* based on the number of labeled pods to the pods for the scaled object. Using this relation, the Custom Metrics Autoscaler Operator scales the object according to the scaling policy in the **ScaledObject** or **ScaledJob** specification.

The pod counts includes pods with a **Succeeded** or **Failed** phase.

For example, if you have a **frontend** deployment and a **backend** deployment. You can use a **kubernetes-workload** trigger to scale the **backend** deployment based on the number of **frontend** pods. If number of **frontend** pods goes up, the Operator would scale the **backend** pods to maintain the specified ratio. In this example, if there are 10 pods with the **app=frontend** pod selector, the Operator scales the backend pods to 5 in order to maintain the **0.5** ratio set in the scaled object.

Example scaled object with a Kubernetes workload trigger

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: workload-scaledobject
  namespace: my-namespace
spec:
  triggers:
  - type: kubernetes-workload 1
    metadata:
      podSelector: 'app=frontend' 2
      value: '0.5' 3
      activationValue: '3.1' 4
```

1 Specifies a Kubernetes workload trigger.

2 Specifies one or more pod selectors and/or set-based selectors, separated with commas, to use to get the pod count.

3 Specifies the target relation between the scaled workload and the number of pods that match the selector. The relation is calculated following the following formula:

```
relation = (pods that match the selector) / (scaled workload pods)
```

4   Optional: Specifies the target value for scaler activation phase. The default is **0**.

3.5. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGER AUTHENTICATIONS

A trigger authentication allows you to include authentication information in a scaled object or a scaled job that can be used by the associated containers. You can use trigger authentications to pass OpenShift Dedicated secrets, platform-native pod authentication mechanisms, environment variables, and so on.

You define a **TriggerAuthentication** object in the same namespace as the object that you want to scale. That trigger authentication can be used only by objects in that namespace.

Alternatively, to share credentials between objects in multiple namespaces, you can create a **ClusterTriggerAuthentication** object that can be used across all namespaces.

Trigger authentications and cluster trigger authentication use the same configuration. However, a cluster trigger authentication requires an additional **kind** parameter in the authentication reference of the scaled object.

Example trigger authentication that uses a bound service account token

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace 1
spec:
  boundServiceAccountToken: 2
  - parameter: bearerToken
    serviceAccountName: thanos 3
```

1   Specifies the namespace of the object you want to scale.

2   Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.

3   Specifies the name of the service account to use.

Example cluster trigger authentication that uses a bound service account token

```
kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: bound-service-account-token-triggerauthentication 1
spec:
  boundServiceAccountToken: 2
  - parameter: bearerToken
    serviceAccountName: thanos 3
```

1   Specifies the namespace of the object you want to scale.

2   Specifies that this cluster trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.

3   Specifies the name of the service account to use.

Example trigger authentication that uses a secret for Basic authentication

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: username 3
    name: my-basic-secret 4
    key: username 5
  - parameter: password
    name: my-basic-secret
    key: password
```

1   Specifies the namespace of the object you want to scale.

2   Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.

3   Specifies the authentication parameter to supply by using the secret.

4   Specifies the name of the secret to use. See the following example secret for Basic authentication.

5   Specifies the key in the secret to use with the specified parameter.

Example secret for Basic authentication

```
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-secret
  namespace: default
data:
  username: "dXNlcm5hbWU=" 1
  password: "cGFzc3dvcmQ="
```

1   User name and password to supply to the trigger authentication. The values in the **data** stanza must be base-64 encoded.

Example trigger authentication that uses a secret for CA details

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
```

```
    name: secret-triggerauthentication
    namespace: my-namespace 1
spec:
  secretTargetRef: 2
   - parameter: key 3
     name: my-secret 4
     key: client-key.pem 5
   - parameter: ca 6
     name: my-secret 7
     key: ca-cert.pem 8
```

[1] Specifies the namespace of the object you want to scale.

[2] Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.

[3] Specifies the type of authentication to use.

[4] Specifies the name of the secret to use.

[5] Specifies the key in the secret to use with the specified parameter.

[6] Specifies the authentication parameter for a custom CA when connecting to the metrics endpoint.

[7] Specifies the name of the secret to use. See the following example secret with certificate authority (CA) details.

[8] Specifies the key in the secret to use with the specified parameter.

Example secret with certificate authority (CA) details

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: my-namespace
data:
  ca-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0... 1
  client-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0... 2
  client-key.pem: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0t...
```

[1] Specifies the TLS CA Certificate for authentication of the metrics endpoint. The value must be base-64 encoded.

[2] Specifies the TLS certificates and key for TLS client authentication. The values must be base-64 encoded.

Example trigger authentication that uses a bearer token

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: token-triggerauthentication
```

```
    namespace: my-namespace 1
  spec:
    secretTargetRef: 2
    - parameter: bearerToken 3
      name: my-secret 4
      key: bearerToken 5
```

1   Specifies the namespace of the object you want to scale.

2   Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.

3   Specifies the type of authentication to use.

4   Specifies the name of the secret to use. See the following example secret for a bearer token.

5   Specifies the key in the token to use with the specified parameter.

Example secret for a bearer token

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: my-namespace
data:
  bearerToken: "<bearer_token>" 1
```

1   Specifies a bearer token to use with bearer authentication. The value must be base-64 encoded.

Example trigger authentication that uses an environment variable

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: env-var-triggerauthentication
  namespace: my-namespace 1
spec:
  env: 2
  - parameter: access_key 3
    name: ACCESS_KEY 4
    containerName: my-container 5
```

1   Specifies the namespace of the object you want to scale.

2   Specifies that this trigger authentication uses environment variables for authorization when connecting to the metrics endpoint.

3   Specify the parameter to set with this variable.

4   Specify the name of the environment variable.

5 Optional: Specify a container that requires authentication. The container must be in the same resource as referenced by **scaleTargetRef** in the scaled object.

Example trigger authentication that uses pod authentication providers

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: pod-id-triggerauthentication
  namespace: my-namespace 1
spec:
  podIdentity: 2
    provider: aws-eks 3
```

1 Specifies the namespace of the object you want to scale.

2 Specifies that this trigger authentication uses a platform-native pod authentication when connecting to the metrics endpoint.

3 Specifies a pod identity. Supported values are **none**, **azure**, **gcp**, **aws-eks**, or **aws-kiam**. The default is **none**.

Additional resources

- [Understanding and creating service accounts](#)

- [Providing sensitive data to pods](#) .

3.5.1. Using trigger authentications

You use trigger authentications and cluster trigger authentications by using a custom resource to create the authentication, then add a reference to a scaled object or scaled job.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

- If you are using a bound service account token, the service account must exist.

- If you are using a bound service account token, a role-based access control (RBAC) object that enables the Custom Metrics Autoscaler Operator to request service account tokens from the service account must exist.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: keda-operator-token-creator
  namespace: <namespace_name> 1
rules:
- apiGroups:
  - ""
  resources:
```

```
  - serviceaccounts/token
  verbs:
  - create
  resourceNames:
  - thanos 2
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: keda-operator-token-creator-binding
  namespace: <namespace_name> 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: keda-operator-token-creator
subjects:
- kind: ServiceAccount
  name: keda-operator
  namespace: openshift-keda
```

1 Specifies the namespace of the service account.

2 Specifies the name of the service account.

3 Specifies the namespace of the service account.

- If you are using a secret, the **Secret** object must exist.

Procedure

1. Create the **TriggerAuthentication** or **ClusterTriggerAuthentication** object.

    a. Create a YAML file that defines the object:

    Example trigger authentication with a bound service account token

    ```
    kind: TriggerAuthentication
    apiVersion: keda.sh/v1alpha1
    metadata:
      name: prom-triggerauthentication
      namespace: my-namespace 1
      spec:
      boundServiceAccountToken: 2
        - parameter: token
          serviceAccountName: thanos 3
    ```

    1 Specifies the namespace of the object you want to scale.

    2 Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.

    3 Specifies the name of the service account to use.

b. Create the **TriggerAuthentication** object:

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

2. Create or edit a **ScaledObject** YAML file that uses the trigger authentication:

a. Create a YAML file that defines the object by running the following command:

Example scaled object with a trigger authentication

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: kedatest # replace <NAMESPACE>
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "basic"
    authenticationRef:
      name: prom-triggerauthentication ❶
      kind: TriggerAuthentication ❷
```

❶ Specify the name of your trigger authentication object.

❷ Specify **TriggerAuthentication**. **TriggerAuthentication** is the default.

Example scaled object with a cluster trigger authentication

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
  - type: prometheus
    metadata:
```

```
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: kedatest # replace <NAMESPACE>
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "basic"
   authenticationRef:
     name: prom-cluster-triggerauthentication  ❶
     kind: ClusterTriggerAuthentication  ❷
```

❶   Specify the name of your trigger authentication object.

❷   Specify **ClusterTriggerAuthentication**.

b. Create the scaled object by running the following command:

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

3.6. UNDERSTANDING HOW TO ADD CUSTOM METRICS AUTOSCALERS

To add a custom metrics autoscaler, create a **ScaledObject** custom resource for a deployment, stateful set, or custom resource. Create a **ScaledJob** custom resource for a job.

You can create only one scaled object for each workload that you want to scale. Also, you cannot use a scaled object and the horizontal pod autoscaler (HPA) on the same workload.

3.6.1. Adding a custom metrics autoscaler to a workload

You can create a custom metrics autoscaler for a workload that is created by a **Deployment**, **StatefulSet**, or **custom resource** object.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

- If you use a custom metrics autoscaler for scaling based on CPU or memory:

  - 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
    ```

    Example output

    ```
    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>
```

- The pods associated with the object you want to scale must include specified memory and CPU limits. For example:

Example pod spec

```
apiVersion: v1
kind: Pod
# ...
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
# ...
```

Procedure

1. Create a YAML file similar to the following. Only the name **<2>**, object name **<4>**, and object kind **<5>** are required:

Example scaled object

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "0"   1
  name: scaledobject   2
  namespace: my-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1   3
    name: example-deployment   4
    kind: Deployment   5
```

```
      envSourceContainerName: .spec.template.spec.containers[0] 6
   cooldownPeriod:  200 7
   maxReplicaCount: 100 8
   minReplicaCount: 0 9
   metricsServer: 10
    auditConfig:
      logFormat: "json"
      logOutputVolumeClaim: "persistentVolumeClaimName"
      policy:
        rules:
        - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
      lifetime:
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
   fallback: 11
    failureThreshold: 3
    replicas: 6
    behavior: static 12
   pollingInterval: 30 13
   advanced:
    restoreToOriginalReplicaCount: false 14
    horizontalPodAutoscalerConfig:
      name: keda-hpa-scale-down 15
      behavior: 16
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
          - type: Percent
            value: 100
            periodSeconds: 15
   triggers:
   - type: prometheus 17
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: kedatest
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: basic
    authenticationRef: 18
      name: prom-triggerauthentication
      kind: TriggerAuthentication
```

1. Optional: Specifies that the Custom Metrics Autoscaler Operator is to scale the replicas to the specified value and stop autoscaling, as described in the "Pausing the custom metrics autoscaler for a workload" section.

2. Specifies a name for this custom metrics autoscaler.

3. Optional: Specifies the API version of the target resource. The default is **apps/v1**.

4 Specifies the name of the object that you want to scale.

5 Specifies the **kind** as **Deployment**, **StatefulSet** or **CustomResource**.

6 Optional: Specifies the name of the container in the target resource, from which the custom metrics autoscaler gets environment variables holding secrets and so forth. The default is **.spec.template.spec.containers[0]**.

7 Optional. Specifies the period in seconds to wait after the last trigger is reported before scaling the deployment back to **0** if the **minReplicaCount** is set to **0**. The default is **300**.

8 Optional: Specifies the maximum number of replicas when scaling up. The default is **100**.

9 Optional: Specifies the minimum number of replicas when scaling down.

10 Optional: Specifies the parameters for audit logs. as described in the "Configuring audit logging" section.

11 Optional: Specifies the number of replicas to fall back to if a scaler fails to get metrics from the source for the number of times defined by the **failureThreshold** parameter. For more information on fallback behavior, see the KEDA documentation.

12 Optional: Specifies the replica count to be used if a fallback occurs. Enter one of the following options or omit the parameter:

- Enter **static** to use the number of replicas specified by the **fallback.replicas** parameter. This is the default.

- Enter **currentReplicas** to maintain the current number of replicas.

- Enter **currentReplicasIfHigher** to maintain the current number of replicas, if that number is higher than the **fallback.replicas** parameter. If the current number of replicas is lower than the **fallback.replicas** parameter, use the **fallback.replicas** value.

- Enter **currentReplicasIfLower** to maintain the current number of replicas, if that number is lower than the **fallback.replicas** parameter. If the current number of replicas is higher than the **fallback.replicas** parameter, use the **fallback.replicas** value.

13 Optional: Specifies the interval in seconds to check each trigger on. The default is **30**.

14 Optional: Specifies whether to scale back the target resource to the original replica count after the scaled object is deleted. The default is **false**, which keeps the replica count as it is when the scaled object is deleted.

15 Optional: Specifies a name for the horizontal pod autoscaler. The default is **keda-hpa-{scaled-object-name}**.

16 Optional: Specifies a scaling policy to use to control the rate to scale pods up or down, as described in the "Scaling policies" section.

17 Specifies the trigger to use as the basis for scaling, as described in the "Understanding the custom metrics autoscaler triggers" section. This example uses OpenShift Dedicated monitoring.

18

Optional: Specifies a trigger authentication or a cluster trigger authentication. For more information, see *Understanding the custom metrics autoscaler trigger authentication* in the

- Enter **TriggerAuthentication** to use a trigger authentication. This is the default.

- Enter **ClusterTriggerAuthentication** to use a cluster trigger authentication.

2. Create the custom metrics autoscaler by running the following command:

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

Verification

- View the command output to verify that the custom metrics autoscaler was created:

```
$ oc get scaledobject <scaled_object_name>
```

Example output

```
NAME          SCALETARGETKIND    SCALETARGETNAME    MIN  MAX  TRIGGERS
AUTHENTICATION               READY  ACTIVE  FALLBACK  AGE
scaledobject   apps/v1.Deployment  example-deployment   0    50   prometheus  prom-
triggerauthentication True   True    True      17s
```

Note the following fields in the output:

- **TRIGGERS**: Indicates the trigger, or scaler, that is being used.

- **AUTHENTICATION**: Indicates the name of any trigger authentication being used.

- **READY**: Indicates whether the scaled object is ready to start scaling:

  - If **True**, the scaled object is ready.

  - If **False**, the scaled object is not ready because of a problem in one or more of the objects you created.

- **ACTIVE**: Indicates whether scaling is taking place:

  - If **True**, scaling is taking place.

  - If **False**, scaling is not taking place because there are no metrics or there is a problem in one or more of the objects you created.

- **FALLBACK**: Indicates whether the custom metrics autoscaler is able to get metrics from the source

  - If **False**, the custom metrics autoscaler is getting metrics.

  - If **True**, the custom metrics autoscaler is getting metrics because there are no metrics or there is a problem in one or more of the objects you created.

3.6.2. Additional resources

- Understanding custom metrics autoscaler trigger authentications

3.7. PAUSING THE CUSTOM METRICS AUTOSCALER FOR A SCALED OBJECT

You can pause and restart the autoscaling of a workload, as needed.

For example, you might want to pause autoscaling before performing cluster maintenance or to avoid resource starvation by removing non-mission-critical workloads.

3.7.1. Pausing a custom metrics autoscaler

You can pause the autoscaling of a scaled object by adding the **autoscaling.keda.sh/paused-replicas** annotation to the custom metrics autoscaler for that scaled object. The custom metrics autoscaler scales the replicas for that workload to the specified value and pauses autoscaling until the annotation is removed.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```

Procedure

1. Use the following command to edit the **ScaledObject** CR for your workload:

   ```
   $ oc edit ScaledObject scaledobject
   ```

2. Add the **autoscaling.keda.sh/paused-replicas** annotation with any value:

   ```
   apiVersion: keda.sh/v1alpha1
   kind: ScaledObject
   metadata:
     annotations:
       autoscaling.keda.sh/paused-replicas: "4"  ❶
     creationTimestamp: "2023-02-08T14:41:01Z"
     generation: 1
     name: scaledobject
     namespace: my-project
     resourceVersion: '65729'
     uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
   ```

   ❶ Specifies that the Custom Metrics Autoscaler Operator is to scale the replicas to the specified value and stop autoscaling.

3.7.2. Restarting the custom metrics autoscaler for a scaled object

You can restart a paused custom metrics autoscaler by removing the **autoscaling.keda.sh/paused-replicas** annotation for that **ScaledObject**.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```

Procedure

1. Use the following command to edit the **ScaledObject** CR for your workload:

   ```
   $ oc edit ScaledObject scaledobject
   ```

2. Remove the **autoscaling.keda.sh/paused-replicas** annotation.

   ```
   apiVersion: keda.sh/v1alpha1
   kind: ScaledObject
   metadata:
     annotations:
       autoscaling.keda.sh/paused-replicas: "4" 1
     creationTimestamp: "2023-02-08T14:41:01Z"
     generation: 1
     name: scaledobject
     namespace: my-project
     resourceVersion: '65729'
     uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
   ```

   1   Remove this annotation to restart a paused custom metrics autoscaler.

3.8. GATHERING AUDIT LOGS

You can gather audit logs, which are a security–relevant chronological set of records documenting the sequence of activities that have affected the system by individual users, administrators, or other components of the system.

For example, audit logs can help you understand where an autoscaling request is coming from. This is key information when backends are getting overloaded by autoscaling requests made by user applications and you need to determine which is the troublesome application.

3.8.1. Configuring audit logging

You can configure auditing for the Custom Metrics Autoscaler Operator by editing the **KedaController** custom resource. The logs are sent to an audit log file on a volume that is secured by using a persistent volume claim in the **KedaController** CR.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. Edit the **KedaController** custom resource to add the **auditConfig** stanza:

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: keda
spec:
# ...
  metricsServer:
# ...
    auditConfig:
      logFormat: "json" ❶
      logOutputVolumeClaim: "pvc-audit-log" ❷
      policy:
        rules: ❸
        - level: Metadata
        omitStages: "RequestReceived" ❹
        omitManagedFields: false ❺
      lifetime: ❻
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
```

❶ Specifies the output format of the audit log, either **legacy** or **json**.

❷ Specifies an existing persistent volume claim for storing the log data. All requests coming to the API server are logged to this persistent volume claim. If you leave this field empty, the log data is sent to stdout.

❸ Specifies which events should be recorded and what data they should include:

- **None**: Do not log events.

- **Metadata**: Log only the metadata for the request, such as user, timestamp, and so forth. Do not log the request text and the response text. This is the default.

- **Request**: Log only the metadata and the request text but not the response text. This option does not apply for non-resource requests.

- **RequestResponse**: Log event metadata, request text, and response text. This option does not apply for non-resource requests.

❹ Specifies stages for which no event is created.

❺ Specifies whether to omit the managed fields of the request and response bodies from being written to the API audit log, either **true** to omit the fields or **false** to include the fields.

❻ Specifies the size and lifespan of the audit logs.

- **maxAge**: The maximum number of days to retain audit log files, based on the timestamp encoded in their filename.

- **maxBackup**: The maximum number of audit log files to retain. Set to **0** to retain all audit log files.

- **maxSize**: The maximum size in megabytes of an audit log file before it gets rotated.

Verification

1. View the audit log file directly:

   a. Obtain the name of the **keda-metrics-apiserver-*** pod:

   ```
   oc get pod -n keda
   ```

   ### Example output

   ```
   NAME                                          READY   STATUS    RESTARTS   AGE
   custom-metrics-autoscaler-operator-5cb44cd75d-9v4lv   1/1   Running   0       8m20s
   keda-metrics-apiserver-65c7cc44fd-rrl4r               1/1   Running   0       2m55s
   keda-operator-776cbb6768-zpj5b                        1/1   Running   0       2m55s
   ```

   b. View the log data by using a command similar to the following:

   ```
   $ oc logs keda-metrics-apiserver-<hash>|grep -i metadata   1
   ```

   1 Optional: You can use the **grep** command to specify the log level to display: **Metadata**, **Request**, **RequestResponse**.

   For example:

   ```
   $ oc logs keda-metrics-apiserver-65c7cc44fd-rrl4r|grep -i metadata
   ```

   ### Example output

   ```
    ...
   {"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"4c81d41b-
   3dab-4675-90ce-
   20b87ce24013","stage":"ResponseComplete","requestURI":"/healthz","verb":"get","user":
   {"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":
   ["10.131.0.1"],"userAgent":"kube-probe/1.28","responseStatus":{"metadata":
   {},"code":200},"requestReceivedTimestamp":"2023-02-
   16T13:00:03.554567Z","stageTimestamp":"2023-02-
   16T13:00:03.555032Z","annotations":
   {"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
    ...
   ```

2. Alternatively, you can view a specific log:

   a. Use a command similar to the following to log into the **keda-metrics-apiserver-*** pod:

   ```
   $ oc rsh pod/keda-metrics-apiserver-<hash> -n keda
   ```

   For example:

   ```
   $ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n keda
   ```

b. Change to the **/var/audit-policy/** directory:

```
sh-4.4$ cd /var/audit-policy/
```

c. List the available logs:

```
sh-4.4$ ls
```

Example output

```
log-2023.02.17-14:50  policy.yaml
```

d. View the log, as needed:

```
sh-4.4$ cat <log_name>/<pvc_name>|grep -i <log_level> ❶
```

❶ Optional: You can use the **grep** command to specify the log level to display: **Metadata**, **Request**, **RequestResponse**.

For example:

```
sh-4.4$ cat log-2023.02.17-14:50/pvc-audit-log|grep -i Request
```

Example output

```
 ...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"63e7f68c-04ec-
4f4d-8749-
bf1656572a41","stage":"ResponseComplete","requestURI":"/openapi/v2","verb":"get","user
":{"username":"system:aggregator","groups":["system:authenticated"]},"sourceIPs":
["10.128.0.1"],"responseStatus":{"metadata":
{},"code":304},"requestReceivedTimestamp":"2023-02-
17T13:12:55.035478Z","stageTimestamp":"2023-02-
17T13:12:55.038346Z","annotations":
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by
ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group
\"system:authenticated\""}}
 ...
```

3.9. GATHERING DEBUGGING DATA

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

To help troubleshoot your issue, provide the following information:

- Data gathered using the **must-gather** tool.

- The unique cluster ID.

You can use the **must-gather** tool to collect data about the Custom Metrics Autoscaler Operator and its components, including the following items:

- The **keda** namespace and its child objects.

- The Custom Metric Autoscaler Operator installation objects.

- The Custom Metric Autoscaler Operator CRD objects.

3.9.1. Gathering debugging data

The following command runs the **must-gather** tool for the Custom Metrics Autoscaler Operator:

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?
(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

> **NOTE**
>
> The standard OpenShift Dedicated **must-gather** command, **oc adm must-gather**, does not collect Custom Metrics Autoscaler Operator data.

Prerequisites

- You are logged in to OpenShift Dedicated as a user with the **dedicated-admin** role.

- The OpenShift Dedicated CLI (**oc**) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.

2. Perform one of the following:

   - To get only the Custom Metrics Autoscaler Operator **must-gather** data, use the following command:

     ```
     $ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
     -n openshift-marketplace \
     -o jsonpath='{.status.channels[?
     (@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
     ```

     The custom image for the **must-gather** command is pulled directly from the Operator package manifests, so that it works on any cluster where the Custom Metric Autoscaler Operator is available.

   - To gather the default **must-gather** data in addition to the Custom Metric Autoscaler Operator information:

     a. Use the following command to obtain the Custom Metrics Autoscaler Operator image and set it as an environment variable:

        ```
        $ IMAGE="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
        ```

```
      -n openshift-marketplace \
      -o jsonpath='{.status.channels[?
    (@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

b. Use the **oc adm must-gather** with the Custom Metrics Autoscaler Operator image:

```
$ oc adm must-gather --image-stream=openshift/must-gather --image=${IMAGE}
```

Example 3.1. Example must-gather output for the Custom Metric Autoscaler

```
└── keda
    ├── apps
    │   ├── daemonsets.yaml
    │   ├── deployments.yaml
    │   ├── replicasets.yaml
    │   └── statefulsets.yaml
    ├── apps.openshift.io
    │   └── deploymentconfigs.yaml
    ├── autoscaling
    │   └── horizontalpodautoscalers.yaml
    ├── batch
    │   ├── cronjobs.yaml
    │   └── jobs.yaml
    ├── build.openshift.io
    │   ├── buildconfigs.yaml
    │   └── builds.yaml
    ├── core
    │   ├── configmaps.yaml
    │   ├── endpoints.yaml
    │   ├── events.yaml
    │   ├── persistentvolumeclaims.yaml
    │   ├── pods.yaml
    │   ├── replicationcontrollers.yaml
    │   ├── secrets.yaml
    │   └── services.yaml
    ├── discovery.k8s.io
    │   └── endpointslices.yaml
    ├── image.openshift.io
    │   └── imagestreams.yaml
    ├── k8s.ovn.org
    │   ├── egressfirewalls.yaml
    │   └── egressqoses.yaml
    ├── keda.sh
    │   ├── kedacontrollers
    │   │   └── keda.yaml
    │   ├── scaledobjects
    │   │   └── example-scaledobject.yaml
    │   └── triggerauthentications
    │       └── example-triggerauthentication.yaml
    ├── monitoring.coreos.com
    │   └── servicemonitors.yaml
    ├── networking.k8s.io
    │   └── networkpolicies.yaml
    ├── keda.yaml
    ├── pods
```

```
│     │       ├──── custom-metrics-autoscaler-operator-58bd9f458-ptgwx
│     │       ├──── custom-metrics-autoscaler-operator
│     │       │     └──── custom-metrics-autoscaler-operator
│     │       │           └──── logs
│     │       │                 ├──── current.log
│     │       │                 ├──── previous.insecure.log
│     │       │                 └──── previous.log
│     │       └──── custom-metrics-autoscaler-operator-58bd9f458-ptgwx.yaml
│     ├──── custom-metrics-autoscaler-operator-58bd9f458-thbsh
│     │     └──── custom-metrics-autoscaler-operator
│     │           └──── custom-metrics-autoscaler-operator
│     │                 └──── logs
│     ├──── keda-metrics-apiserver-65c7cc44fd-6wq4g
│     │     ├──── keda-metrics-apiserver
│     │     │     └──── keda-metrics-apiserver
│     │     │           └──── logs
│     │     │                 ├──── current.log
│     │     │                 ├──── previous.insecure.log
│     │     │                 └──── previous.log
│     │     └──── keda-metrics-apiserver-65c7cc44fd-6wq4g.yaml
│     └──── keda-operator-776cbb6768-fb6m5
│           ├──── keda-operator
│           │     └──── keda-operator
│           │           └──── logs
│           │                 ├──── current.log
│           │                 ├──── previous.insecure.log
│           │                 └──── previous.log
│           └──── keda-operator-776cbb6768-fb6m5.yaml
├──── policy
│     └──── poddisruptionbudgets.yaml
└──── route.openshift.io
      └──── routes.yaml
```

3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/    ❶
```

❶   Replace **must-gather-local.5421342344627712289/** with the actual directory name.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#) .

3.10. VIEWING OPERATOR METRICS

The Custom Metrics Autoscaler Operator exposes ready-to-use metrics that it pulls from the on-cluster monitoring component. You can query the metrics by using the Prometheus Query Language (PromQL) to analyze and diagnose issues. All metrics are reset when the controller pod restarts.

3.10.1. Accessing performance metrics

You can access the metrics and run queries by using the OpenShift Dedicated web console.

Procedure

1. Select the **Administrator** perspective in the OpenShift Dedicated web console.

2. Select **Observe → Metrics**.

3. To create a custom query, add your PromQL query to the **Expression** field.

4. To add multiple queries, select **Add Query**.

3.10.1.1. Provided Operator metrics

The Custom Metrics Autoscaler Operator exposes the following metrics, which you can view by using the OpenShift Dedicated web console.

Table 3.1. Custom Metric Autoscaler Operator metrics

Metric name	Description
keda_scaler_activity	Whether the particular scaler is active or inactive. A value of **1** indicates the scaler is active; a value of **0** indicates the scaler is inactive.
keda_scaler_metrics_value	The current value for each scaler's metric, which is used by the Horizontal Pod Autoscaler (HPA) in computing the target average.
keda_scaler_metrics_latency	The latency of retrieving the current metric from each scaler.
keda_scaler_errors	The number of errors that have occurred for each scaler.
keda_scaler_errors_total	The total number of errors encountered for all scalers.
keda_scaled_object_errors	The number of errors that have occurred for each scaled obejct.
keda_resource_totals	The total number of Custom Metrics Autoscaler custom resources in each namespace for each custom resource type.
keda_trigger_totals	The total number of triggers by trigger type.

Custom Metrics Autoscaler Admission webhook metrics

The Custom Metrics Autoscaler Admission webhook also exposes the following Prometheus metrics.

Metric name	Description
keda_scaled_object_validation_total	The number of scaled object validations.

Metric name	Description
keda_scaled_object_valid ation_errors	The number of validation errors.

3.11. REMOVING THE CUSTOM METRICS AUTOSCALER OPERATOR

You can remove the custom metrics autoscaler from your OpenShift Dedicated cluster. After removing the Custom Metrics Autoscaler Operator, remove other components associated with the Operator to avoid potential issues.

> **NOTE**
>
> Delete the **KedaController** custom resource (CR) first. If you do not delete the **KedaController** CR, OpenShift Dedicated can hang when you delete the **keda** project. If you delete the Custom Metrics Autoscaler Operator before deleting the CR, you are not able to delete the CR.

3.11.1. Uninstalling the Custom Metrics Autoscaler Operator

Use the following procedure to remove the custom metrics autoscaler from your OpenShift Dedicated cluster.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. In the OpenShift Dedicated web console, click **Ecosystem → Installed Operators**.

2. Switch to the **keda** project.

3. Remove the **KedaController** custom resource.

   a. Find the **CustomMetricsAutoscaler** Operator and click the **KedaController** tab.

   b. Find the custom resource, and then click **Delete KedaController**.

   c. Click **Uninstall**.

4. Remove the Custom Metrics Autoscaler Operator:

   a. Click **Ecosystem → Installed Operators**.

   b. Find the **CustomMetricsAutoscaler** Operator and click the Options menu ⋮ and select **Uninstall Operator**.

   c. Click **Uninstall**.

5. Optional: Use the OpenShift CLI to remove the custom metrics autoscaler components:

    a. Delete the custom metrics autoscaler CRDs:

        - **clustertriggerauthentications.keda.sh**

        - **kedacontrollers.keda.sh**

        - **scaledjobs.keda.sh**

        - **scaledobjects.keda.sh**

        - **triggerauthentications.keda.sh**

        ```
        $ oc delete crd clustertriggerauthentications.keda.sh kedacontrollers.keda.sh
        scaledjobs.keda.sh scaledobjects.keda.sh triggerauthentications.keda.sh
        ```

        Deleting the CRDs removes the associated roles, cluster roles, and role bindings. However, there might be a few cluster roles that must be manually deleted.

    b. List any custom metrics autoscaler cluster roles:

        ```
        $ oc get clusterrole | grep keda.sh
        ```

    c. Delete the listed custom metrics autoscaler cluster roles. For example:

        ```
        $ oc delete clusterrole.keda.sh-v1alpha1-admin
        ```

    d. List any custom metrics autoscaler cluster role bindings:

        ```
        $ oc get clusterrolebinding | grep keda.sh
        ```

    e. Delete the listed custom metrics autoscaler cluster role bindings. For example:

        ```
        $ oc delete clusterrolebinding.keda.sh-v1alpha1-admin
        ```

6. Delete the custom metrics autoscaler project:

    ```
    $ oc delete project keda
    ```

7. Delete the Cluster Metric Autoscaler Operator:

    ```
    $ oc delete operator/openshift-custom-metrics-autoscaler-operator.keda
    ```

CHAPTER 4. CONTROLLING POD PLACEMENT ONTO NODES (SCHEDULING)

4.1. 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 Dedicated 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 Dedicated 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.
You can control pod placement by using the following scheduling features:

- Pod affinity and anti-affinity rules

- Node affinity

- Node selectors

- Node overcommitment

4.1.1. About the default scheduler

The default OpenShift Dedicated pod scheduler is responsible for determining the placement of new pods onto nodes within the cluster. It reads data from the pod and finds a node that is a good fit based on configured profiles. It is completely independent and exists as a standalone solution. It does not modify the pod; it creates a binding for the pod that ties the pod to the particular node.

4.1.1.1. 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*, or *filters*.

Prioritizes the filtered list of nodes

This is achieved by passing each node through a series of *priority*, or *scoring*, 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 scoring function. The node score provided by each scoring function is multiplied by the weight (default

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

Selects 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.

4.1.2. Scheduler use cases

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

4.1.2.1. 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 Controlling pod placement on nodes using node affinity rules and Placing pods relative to other pods using affinity and anti-affinity rules .

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.

4.1.2.2. 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 Controlling pod placement on nodes using node affinity rules and Placing pods relative to other pods using affinity and anti-affinity rules .

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.

4.2. 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 Dedicated, *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.

4.2.1. 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, availability zones, or availability sets to reduce correlated failures.

> **NOTE**
>
> A label selector might match pods with multiple pod deployments. Use unique combinations of labels when configuring anti-affinity rules to avoid matching pods.

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** spec 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** spec 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAffinity: 1
```

```
        requiredDuringSchedulingIgnoredDuringExecution: 2
      - labelSelector:
          matchExpressions:
          - key: security 3
            operator: In 4
            values:
            - S1 5
        topologyKey: topology.kubernetes.io/zone
    containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

[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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
```

```
allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
```

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.

4.2.2. 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.

> **NOTE**
>
> You cannot add an affinity directly to a scheduled pod.

Procedure

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

   a. Create a YAML file with the following content:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: security-s1
     labels:
       security: S1
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     containers:
     - name: security-s1
       image: docker.io/ocpqe/hello-pod
       securityContext:
   ```

```
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
```

b. Create the pod.

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

2. When creating other pods, configure the following parameters to add the affinity:

a. Create a YAML file with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1-east
# ...
spec:
  affinity: 1
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: 2
      - labelSelector:
          matchExpressions:
          - key: security 3
            values:
            - S1
            operator: In 4
        topologyKey: topology.kubernetes.io/zone 5
# ...
```

1   Adds a pod affinity.

2   Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter or the **preferredDuringSchedulingIgnoredDuringExecution** parameter.

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

4   Specifies 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.

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

b. Create the pod.

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

4.2.3. 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.
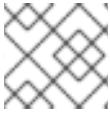
> **NOTE**
>
> You cannot add an affinity directly to a scheduled pod.

Procedure

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

   a. Create a YAML file with the following content:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: security-s1
     labels:
       security: S1
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     containers:
     - name: security-s1
       image: docker.io/ocpqe/hello-pod
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           drop: [ALL]
   ```

   b. Create the pod.

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

2. When creating other pods, configure the following parameters:

   a. Create a YAML file with the following content:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: security-s2-east
   # ...
   spec:
   # ...
     affinity: 1
       podAntiAffinity:
         preferredDuringSchedulingIgnoredDuringExecution: 2
         - weight: 100 3
           podAffinityTerm:
             labelSelector:
               matchExpressions:
               - key: security 4
                 values:
                 - S1
   ```

```
            operator: In 5
          topologyKey: kubernetes.io/hostname 6
   # ...
```

1 Adds a pod anti-affinity.

2 Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter or the **preferredDuringSchedulingIgnoredDuringExecution** parameter.

3 For a preferred rule, specifies a weight for the node, 1-100. The node that with highest weight is preferred.

4 Specifies 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 **values** parameters as the label on the first pod.

5 Specifies 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.

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

b. Create the pod.

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

4.2.4. Sample pod affinity and anti-affinity rules

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

4.2.4.1. Pod Affinity

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
```

```
      drop: [ALL]
  # ...
```

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

```
apiVersion: v1
kind: Pod
metadata:
  name: team4a
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

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

4.2.4.2. 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**.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
```

```
    - name: ocp
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    # ...
```

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

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

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

4.2.4.3. 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**.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
```

```
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    containers:
    - name: ocp
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  # ...
```

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

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- 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:

Example output

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

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

4.3.1. 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** spec 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** spec 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**:

Example pod configuration file with a node affinity required rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

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** spec. 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:

Example pod configuration file with a node affinity preferred rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

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** spec. 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.

4.3.2. 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. Create a pod with a specific label in the pod spec:

   a. Create a YAML file with the following content:

   > **NOTE**
   >
   > You cannot add an affinity directly to a scheduled pod.

   Example output

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: s1
   spec:
     affinity: 1
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution: 2
   ```

```
      nodeSelectorTerms:
      - matchExpressions:
        - key: e2e-az-name 3
          values:
          - e2e-az1
          - e2e-az2
          operator: In 4
    #...
```

1   Adds a pod affinity.

2   Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter.

3   Specifies 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 **values** parameters as the label in the node.

4   Specifies 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.

    b.   Create the pod:

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

4.3.3. 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. Create a pod with a specific label:

    a.   Create a YAML file with the following content:

> **NOTE**
>
> You cannot add an affinity directly to a scheduled pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
spec:
  affinity: 1
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 3
        preference:
          matchExpressions:
```

```
        - key: e2e-az-name  4
          values:
          - e2e-az3
          operator: In  5
    #...
```

1 Adds a pod affinity.

2 Configures the **preferredDuringSchedulingIgnoredDuringExecution** parameter.

3 Specifies a weight for the node, as a number 1–100. The node with highest weight is preferred.

4 Specifies 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 **values** parameters as the label in the node.

5 Specifies 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.

b. Create the pod.

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

4.3.4. Sample node affinity rules

The following examples demonstrate node affinity.

4.3.4.1. 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
```

TIP

You can alternatively apply the following YAML to add the label:

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: us
#...
```

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

```
$ cat pod-s1.yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
#...
```

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

  ```
  $ oc get pod -o wide
  ```

Example output

```
NAME    READY   STATUS    RESTARTS  AGE   IP    NODE
pod-s1  1/1     Running   0         4m    IP1   node1
```

4.3.4.2. 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
  ```

TIP

You can alternatively apply the following YAML to add the label:

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: emea
#...
```

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

```
$ cat pod-s1.yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
#...
```

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

```
$ oc describe pod pod-s1
```

Example output

```
...
```

```
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).
```

4.4. 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.

4.4.1. 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 Dedicated 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** object 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, to ensure that the overrides apply.

After these overrides, the container limits and requests must still be validated by any **LimitRange** object 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 **LimitRange** objects with caution.

4.4.2. Understanding nodes overcommitment

To maintain optimal system performance and stability in an overcommitted environment in OpenShift Dedicated, configure your nodes to manage resource contention effectively.

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

OpenShift Dedicated also configures the kernel to not 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 the 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
```

Example output

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

Example output

```
#...
vm.panic_on_oom = 0
#...
```

> **NOTE**
>
> The previous commands should already be set on nodes, so 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

4.5. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS

A *node selector* specifies a map of key/value pairs that 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 same key/value node selector as the label on the node.

4.5.1. About node selectors

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Dedicated schedules the pods on nodes that contain matching labels.

You can use a node selector to place specific pods on specific nodes, cluster-wide node selectors to place new pods on specific nodes anywhere in the cluster, and project node selectors to place new pods in a project on specific nodes.

For example, as a cluster administrator, you can create an infrastructure where application developers can deploy pods only onto the nodes closest to their geographical location by including a node selector in every pod they create. In this example, the cluster consists of five data centers spread across two

regions. In the U.S., label the nodes as **us-east**, **us-central**, or **us-west**. In the Asia–Pacific region (APAC), label the nodes as **apac-east** or **apac-west**. The developers can add a node selector to the pods they create to ensure the pods get scheduled on those nodes.

A pod is not scheduled if the **Pod** object contains a node selector, but no node has a matching label.

> **IMPORTANT**
>
> If you are using node selectors and node affinity in the same pod configuration, the following rules control pod placement onto nodes:
>
> - 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.

Node selectors on specific pods and nodes

You can control which node a specific pod is scheduled on by using node selectors and labels. To use node selectors and labels, first label the node to avoid pods being descheduled, then add the node selector to the pod.

> **NOTE**
>
> You cannot add a node selector directly to an existing scheduled pod. You must label the object that controls the pod, such as deployment config.

For example, the following **Node** object has the **region: east** label:

Sample Node object with a label

```
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:
    kubernetes.io/os: linux
    topology.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: ''
    topology.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    node.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
```

```
      kubernetes.io/arch: amd64
      region: east 1
      type: user-node
#...
```

1 Labels to match the pod node selector.

A pod has the **type: user-node,region: east** node selector:

Sample **Pod** object with node selectors

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector: 1
    region: east
    type: user-node
#...
```

1 Node selectors to match the node label. The node must have a label for each node selector.

When you create the pod using the example pod spec, it can be scheduled on the example node.

Default cluster-wide node selectors

With default cluster-wide node selectors, when you create a pod in that cluster, OpenShift Dedicated adds the default node selectors to the pod and schedules the pod on nodes with matching labels.

For example, the following **Scheduler** object has the default cluster-wide **region=east** and **type=user-node** node selectors:

Example Scheduler Operator Custom Resource

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
#...
spec:
  defaultNodeSelector: type=user-node,region=east
#...
```

A node in that cluster has the **type=user-node,region=east** labels:

Example **Node** object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
```

```
#...
  labels:
    region: east
    type: user-node
#...
```

Example **Pod** object with a node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector:
    region: east
#...
```

When you create the pod using the example pod spec in the example cluster, the pod is created with the cluster-wide node selector and is scheduled on the labeled node:

Example pod list with the pod on the labeled node

```
NAME    READY STATUS   RESTARTS AGE IP         NODE
NOMINATED NODE   READINESS GATES
pod-s1  1/1   Running  0        20s 10.131.2.6  ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>          <none>
```

> **NOTE**
>
> If the project where you create the pod has a project node selector, that selector takes preference over a cluster-wide node selector. Your pod is not created or scheduled if the pod does not have the project node selector.

Project node selectors

With project node selectors, when you create a pod in this project, OpenShift Dedicated adds the node selectors to the pod and schedules the pods on a node with matching labels. If there is a cluster-wide default node selector, a project node selector takes preference.

For example, the following project has the **region=east** node selector:

Example **Namespace** object

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
  annotations:
    openshift.io/node-selector: "region=east"
#...
```

The following node has the **type=user-node,region=east** labels:

Example **Node** object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
  labels:
    region: east
    type: user-node
#...
```

When you create the pod using the example pod spec in this example project, the pod is created with the project node selectors and is scheduled on the labeled node:

Example **Pod** object

```
apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...
```

Example pod list with the pod on the labeled node

```
NAME    READY  STATUS   RESTARTS  AGE  IP         NODE
NOMINATED NODE   READINESS GATES
pod-s1  1/1    Running  0         20s  10.131.2.6  ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>          <none>
```

A pod in the project is not created or scheduled if the pod contains different node selectors. For example, if you deploy the following pod into the example project, it is not created:

Example **Pod** object with an invalid node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: west-region
#...
spec:
  nodeSelector:
    region: west
#...
```

4.5.2. Using node selectors to control pod placement

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Dedicated schedules the pods on nodes that contain matching labels.

You add labels to a node, a compute machine set, or a machine config. Adding the label to the compute machine set ensures that if the node or machine goes down, new nodes have the label. Labels added to a node or machine config do not persist if the node or machine goes down.

To add node selectors to an existing pod, add a node selector to the controlling object for that pod, such as a **ReplicaSet** object, **DaemonSet** object, **StatefulSet** object, **Deployment** object, or **DeploymentConfig** object. 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. If the pod does not have a controlling object, you must delete the pod, edit the pod spec, and recreate the pod.

> **NOTE**
>
> You cannot add a node selector directly to an existing scheduled pod.

Prerequisites

To add a node selector to existing pods, determine the controlling object for that pod. For example, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** replica set:

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

Example output

```
kind: Pod
apiVersion: v1
metadata:
# ...
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:

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: router-default-66d5cf9464
      uid: d81dd094-da26-11e9-a48a-128e7edf0312
      controller: true
      blockOwnerDeletion: true
# ...
```

Procedure

- Add the matching node selector to a pod:

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

    Example ReplicaSet object with labels

    ```
    kind: ReplicaSet
    apiVersion: apps/v1
    metadata:
      name: hello-node-6fbccf8d9
    # ...
    spec:
    # ...
      template:
        metadata:
          creationTimestamp: null
          labels:
            ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
            pod-template-hash: 66d5cf9464
        spec:
          nodeSelector:
            kubernetes.io/os: linux
            node-role.kubernetes.io/worker: ''
            type: user-node  ❶
    # ...
    ```

    ❶ Add the node selector.

  - To add a node selector to a specific, new pod, add the selector to the **Pod** object directly:

    Example Pod object with a node selector

    ```
    apiVersion: v1
    kind: Pod
    metadata:
      name: hello-node-6fbccf8d9
    # ...
    spec:
      nodeSelector:
        region: east
        type: user-node
    # ...
    ```

    > **NOTE**
    >
    > You cannot add a node selector directly to an existing scheduled pod.

4.6. CONTROLLING POD PLACEMENT BY USING POD TOPOLOGY SPREAD CONSTRAINTS

You can use pod topology spread constraints to provide fine-grained control over the placement of your pods across nodes, zones, regions, or other user-defined topology domains. Distributing pods across failure domains can help to achieve high availability and more efficient resource utilization.

4.6.1. Example use cases

- As an administrator, I want my workload to automatically scale between two to fifteen pods. I want to ensure that when there are only two pods, they are not placed on the same node, to avoid a single point of failure.

- As an administrator, I want to distribute my pods evenly across multiple infrastructure zones to reduce latency and network costs. I want to ensure that my cluster can self-heal if issues arise.

4.6.2. Important considerations

- Pods in an OpenShift Dedicated cluster are managed by *workload controllers* such as deployments, stateful sets, or daemon sets. These controllers define the desired state for a group of pods, including how they are distributed and scaled across the nodes in the cluster. You should set the same pod topology spread constraints on all pods in a group to avoid confusion. When using a workload controller, such as a deployment, the pod template typically handles this for you.

- Mixing different pod topology spread constraints can make OpenShift Dedicated behavior confusing and troubleshooting more difficult. You can avoid this by ensuring that all nodes in a topology domain are consistently labeled. OpenShift Dedicated automatically populates well-known labels, such as **kubernetes.io/hostname**. This helps avoid the need for manual labeling of nodes. These labels provide essential topology information, ensuring consistent node labeling across the cluster.

- Only pods within the same namespace are matched and grouped together when spreading due to a constraint.

- You can specify multiple pod topology spread constraints, but you must ensure that they do not conflict with each other. All pod topology spread constraints must be satisfied for a pod to be placed.

4.6.3. Understanding skew and maxSkew

Skew refers to the difference in the number of pods that match a specified label selector across different topology domains, such as zones or nodes.

The skew is calculated for each domain by taking the absolute difference between the number of pods in that domain and the number of pods in the domain with the lowest amount of pods scheduled. Setting a **maxSkew** value guides the scheduler to maintain a balanced pod distribution.

4.6.3.1. Example skew calculation

You have three zones (A, B, and C), and you want to distribute your pods evenly across these zones. If zone A has 5 pods, zone B has 3 pods, and zone C has 2 pods, to find the skew, you can subtract the number of pods in the domain with the lowest amount of pods scheduled from the number of pods currently in each zone. This means that the skew for zone A is 3, the skew for zone B is 1, and the skew for zone C is 0.

4.6.3.2. The maxSkew parameter

The **maxSkew** parameter defines the maximum allowable difference, or skew, in the number of pods between any two topology domains. If **maxSkew** is set to **1**, the number of pods in any topology domain should not differ by more than 1 from any other domain. If the skew exceeds **maxSkew**, the scheduler attempts to place new pods in a way that reduces the skew, adhering to the constraints.

Using the previous example skew calculation, the skew values exceed the default **maxSkew** value of **1**. The scheduler places new pods in zone B and zone C to reduce the skew and achieve a more balanced distribution, ensuring that no topology domain exceeds the skew of 1.

4.6.4. Example configurations for pod topology spread constraints

You can specify which pods to group together, which topology domains they are spread among, and the acceptable skew.

The following examples demonstrate pod topology spread constraint configurations.

Example to distribute pods that match the specified labels based on their zone

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1 1
    topologyKey: topology.kubernetes.io/zone 2
    whenUnsatisfiable: DoNotSchedule 3
    labelSelector: 4
      matchLabels:
        region: us-east 5
    matchLabelKeys:
      - my-pod-label 6
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1 The maximum difference in number of pods between any two topology domains. The default is **1**, and you cannot specify a value of **0**.

2 The key of a node label. Nodes with this key and identical value are considered to be in the same topology.

3 How to handle a pod if it does not satisfy the spread constraint. The default is **DoNotSchedule**, which tells the scheduler not to schedule the pod. Set to **ScheduleAnyway** to still schedule the pod, but the scheduler prioritizes honoring the skew to not make the cluster more imbalanced.

pod, but the scheduler prioritizes honoring the skew to not make the cluster more imbalanced.

4 Pods that match this label selector are counted and recognized as a group when spreading to satisfy the constraint. Be sure to specify a label selector, otherwise no pods can be matched.

5 Be sure that this **Pod** spec also sets its labels to match this label selector if you want it to be counted properly in the future.

6 A list of pod label keys to select which pods to calculate spreading over.

Example demonstrating a single pod topology spread constraint

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        region: us-east
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

The previous example defines a **Pod** spec with a one pod topology spread constraint. It matches on pods labeled **region: us-east**, distributes among zones, specifies a skew of **1**, and does not schedule the pod if it does not meet these requirements.

Example demonstrating multiple pod topology spread constraints

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod-2
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

```
    topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: node
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
        matchLabels:
          region: us-east
    - maxSkew: 1
      topologyKey: rack
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
        matchLabels:
          region: us-east
    containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

The previous example defines a **Pod** spec with two pod topology spread constraints. Both match on pods labeled **region: us-east**, specify a skew of **1**, and do not schedule the pod if it does not meet these requirements.

The first constraint distributes pods based on a user-defined label **node**, and the second constraint distributes pods based on a user-defined label **rack**. Both constraints must be met for the pod to be scheduled.

CHAPTER 5. USING JOBS AND DAEMON SETS

5.1. RUNNING BACKGROUND TASKS ON NODES AUTOMATICALLY WITH DAEMON SETS

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

A daemon set 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 daemon set will clean up the pods it created.

You can use daemon sets 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, the cluster administrators and the project administrators can create daemon sets.

For more information on daemon sets, see the Kubernetes documentation.
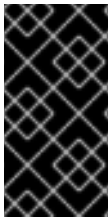
> **IMPORTANT**
>
> Daemon set scheduling is incompatible with project's default node selector. If you fail to disable it, the daemon set gets restricted by merging with the default node selector. This results in frequent pod recreates on the nodes that got unselected by the merged node selector, which in turn puts unwanted load on the cluster.

5.1.1. Scheduled by default scheduler

A daemon set 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, daemon set pods are created and scheduled by the daemon set controller. That introduces the following issues:

- Inconsistent pod behavior: Normal pods waiting to be scheduled are created and in Pending state, but daemon set 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 daemon set controller will make scheduling decisions without considering pod priority and preemption.

The **ScheduleDaemonSetPods** feature, enabled by default in OpenShift Dedicated, lets you schedule daemon sets using the default scheduler instead of the daemon set controller, by adding the **NodeAffinity** term to the daemon set 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 daemon set pod already exists, it is replaced. The daemon set controller only performs these operations when creating or modifying daemon set pods, and no changes are made to the **spec.template** of the daemon set.

```
kind: Pod
apiVersion: v1
metadata:
  name: hello-node-6fbccf8d9-9tmzr
#...
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```
      - matchFields:
        - key: metadata.name
          operator: In
          values:
          - target-host-name
  #...
```

In addition, a **node.kubernetes.io/unschedulable:NoSchedule** toleration is added automatically to daemon set pods. The default scheduler ignores unschedulable Nodes when scheduling daemon set pods.

5.1.2. Creating daemonsets

When creating daemon sets, the **nodeSelector** field is used to indicate the nodes on which the daemon set should deploy replicas.

Prerequisites

- Before you start using daemon sets, 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": ""}}}'
```

TIP

You can alternatively apply the following YAML to disable the default project-wide node selector for a namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/node-selector: ''
#...
```

Procedure

To create a daemon set:

1. Define the daemon set yaml file:

```
apiVersion: apps/v1
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 daemon set.

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 daemon set 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
   ```

   Example output

   ```
   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
   ```

   Example output

   ```
   Node:        openshift-node01.hostname.com/10.14.20.134
   ```

   ```
   $ oc describe pod/hello-daemonset-e3md9|grep Node
   ```

   Example output

> Node: openshift-node02.hostname.com/10.14.20.137
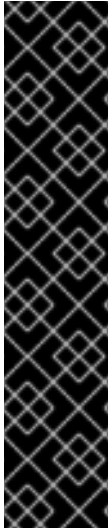
IMPORTANT

- If you update a daemon set pod template, the existing pod replicas are not affected.

- If you delete a daemon set and then create a new daemon set 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 daemon set adds pods to nodes that match the new labels and deletes pods from nodes that do not match the new labels.

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

5.2. RUNNING TASKS IN PODS USING JOBS

A *job* executes a task in your OpenShift Dedicated 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.

Sample Job specification

```
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 The pod replicas a job should run in parallel.

2 Successful pod completions are needed to mark a job completed.

3 The maximum duration the job can run.

4   The number of retries for a job.

5   The template for the pod the controller creates.

6   The restart policy of the pod.

Additional resources

- Jobs (Kubernetes documentation)

5.2.1. Understanding jobs and cron jobs

A job tracks the overall progress of a task and updates its status with information about active, succeeded, and failed pods. Deleting a job cleans 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 Dedicated:

Job

A regular job is a run-once object that creates a task and ensures the job finishes.
There are three main types of task suitable to run as a job:

- Non-parallel jobs:

  - A job that starts only one pod, unless the pod fails.

  - The job is complete as soon as its pod terminates successfully.

- Parallel jobs with a fixed completion count:

  - a job that starts multiple pods.

  - The job represents the overall task and is complete when there is one successful pod for each value in the range **1** to the **completions** value.

- Parallel jobs with a work queue:

  - A job with multiple parallel worker processes in a given pod.

  - OpenShift Dedicated coordinates pods to determine what each should work on or use an external queue service.

  - Each pod is independently capable of determining whether or not all peer pods are complete and that the entire job is done.

  - When any pod from the job terminates with success, no new pods are created.

  - When at least one pod has terminated with success and all pods are terminated, the job is successfully completed.

  - When any pod has exited with success, no other pod should be doing any work for this task or writing any output. Pods should all be in the process of exiting.
    For more information about how to make use of the different types of job, see Job Patterns in the Kubernetes documentation.

Cron job

A job can be scheduled to run multiple times, using a cron job.

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

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period. A cron job creates a **Job** object based on the timezone configured on the control plane node that runs the cronjob controller.

> ⚠️ **WARNING**
>
> A cron job 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.

5.2.1.1. 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 Dedicated creates.

- The **parallelism** parameter, which specifies how many pods running in parallel at any point in time should execute a job.

  - For non-parallel jobs, leave unset. When unset, defaults to **1**.

- The **completions** parameter, specifying how many successful pod completions are needed to finish a job.

  - For non-parallel jobs, leave unset. When unset, defaults to **1**.

  - For parallel jobs with a fixed completion count, specify a value.

  - For parallel jobs with a work queue, leave unset. When unset defaults to the **parallelism** value.

5.2.1.2. 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 Dedicated.

5.2.1.3. 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.

5.2.1.4. Understanding how to configure a cron job to remove artifacts

Cron jobs 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 cron job'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).

5.2.1.5. 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.

5.2.2. Creating jobs

You create a job in OpenShift Dedicated 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: Specify how many pod replicas a job should run in parallel; defaults to **1**.

- For non-parallel jobs, leave unset. When unset, defaults to **1**.

2 Optional: Specify how many successful pod completions are needed to mark a job completed.

- For non-parallel jobs, leave unset. When unset, defaults to **1**.

- For parallel jobs with a fixed completion count, specify the number of completions.

- For parallel jobs with a work queue, leave unset. When unset defaults to the **parallelism** value.

3 Optional: Specify the maximum duration the job can run.

4 Optional: Specify the number of retries for a job. This field defaults to six.

5 Specify the template for the pod the controller creates.

6 Specify the restart policy of the pod:

- **Never**. Do not restart the job.

- **OnFailure**. Restart the job only if it fails.

- **Always**. Always restart the job.
  For details on how OpenShift Dedicated uses restart policy with failed containers, see the Example States in the Kubernetes documentation.

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 create job**. The following command creates and launches a job similar to the one specified in the previous example:

```
$ oc create job pi --image=perl -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

5.2.3. Creating cron jobs

You create a cron job in OpenShift Dedicated by creating a job object.

Procedure

To create a cron job:

1. Create a YAML file similar to the following:

```
apiVersion: batch/v1
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   Schedule for the job specified in cron format. In this example, the job will run every minute.

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

- **Allow** allows cron jobs 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   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   An optional flag allowing the suspension of a cron job. If set to **true**, all subsequent executions will be suspended.

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

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 cron job.

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 cron job:

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

> **NOTE**
>
> You can also create and launch a cron job from a single command using **oc create
> cronjob**. The following command creates and launches a cron job similar to the one
> specified in the previous example:
>
> ```
> $ oc create cronjob pi --image=perl --schedule='*/1 * * * *' -- perl -Mbignum=bpi -wle
> 'print bpi(2000)'
> ```
>
> With **oc create cronjob**, the **--schedule** option accepts schedules in cron format.

CHAPTER 6. WORKING WITH NODES

6.1. VIEWING AND LISTING THE NODES IN YOUR OPENSHIFT DEDICATED 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.

6.1.1. 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
```

The following example is a cluster with healthy nodes:

```
$ oc get nodes
```

Example output

```
NAME              STATUS  ROLES   AGE     VERSION
master.example.com    Ready   master  7h      v1.33.4
node1.example.com     Ready   worker  7h      v1.33.4
node2.example.com     Ready   worker  7h      v1.33.4
```

The following example is a cluster with one unhealthy node:

```
$ oc get nodes
```

Example output

```
NAME              STATUS            ROLES   AGE     VERSION
master.example.com    Ready                 master  7h      v1.33.4
node1.example.com     NotReady,SchedulingDisabled worker  7h      v1.33.4
node2.example.com     Ready                 worker  7h      v1.33.4
```

The conditions that trigger a **NotReady** status are shown later in this section.

- The **-o wide** option provides additional information on nodes.

```
$ oc get nodes -o wide
```

Example output

```
NAME              STATUS  ROLES   AGE   VERSION  INTERNAL-IP   EXTERNAL-IP
```

```
OS-IMAGE                                KERNEL-VERSION          CONTAINER-
RUNTIME
master.example.com Ready   master  171m  v1.33.4  10.0.129.108  <none>      Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)   4.20.0-240.15.1.el8_3.x86_64
cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev
node1.example.com  Ready   worker  72m  v1.33.4  10.0.129.222  <none>       Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)   4.20.0-240.15.1.el8_3.x86_64
cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev
node2.example.com  Ready   worker  164m  v1.33.4  10.0.142.150  <none>       Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)   4.20.0-240.15.1.el8_3.x86_64
cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev
```

- The following command lists information about a single node:

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

For example:

```
$ oc get node node1.example.com
```

Example output

```
NAME              STATUS  ROLES   AGE    VERSION
node1.example.com    Ready   worker  7h     v1.33.4
```

- 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
```

> **NOTE**
>
> The following example contains some values that are specific to OpenShift Dedicated on AWS.

Example output

```
Name:            node1.example.com 1
Roles:           worker 2
Labels:          kubernetes.io/os=linux
                 kubernetes.io/hostname=ip-10-0-131-14
                 kubernetes.io/arch=amd64 3
                 node-role.kubernetes.io/worker=
                 node.kubernetes.io/instance-type=m4.large
                 node.openshift.io/os_id=rhcos
                 node.openshift.io/os_version=4.5
                 region=east
                 topology.kubernetes.io/region=us-east-1
```

```
                     topology.kubernetes.io/zone=us-east-1a
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://1.33.4-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
  Kubelet Version:          v1.33.4
  Kube-Proxy Version:        v1.33.4
PodCIDR:               10.128.4.0/24
```

```
ProviderID:                         aws:///us-east-2a/i-04e87b31dc6b3e171
Non-terminated Pods:                (12 in total) 🔟
  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                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-ovn-kubernetes                ovnkube-node-t4dsn                  80m (0%)
0 (0%)     1630Mi (0%)      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: 1️⃣1️⃣
  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 **worker**.

3   The labels applied to the node.

4   The annotations applied to the node.

5   The taints applied to the node.

6   The node conditions and status. The **conditions** stanza lists the **Ready**, **PIDPressure**, **MemoryPressure**, **DiskPressure** and **OutOfDisk** status. These condition are described later in this section.

7   The IP address and hostname 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.

Among the information shown for nodes, the following node conditions appear in the output of the commands shown in this section:

Table 6.1. Node Conditions

Condition	Description
Ready	If **true**, the node is healthy and ready to accept pods. If **false**, the node is not healthy and is not accepting pods. If **unknown**, the node controller has not received a heartbeat from the node for the **node-monitor-grace-period** (the default is 40 seconds).
DiskPressure	If **true**, the disk capacity is low.
MemoryPressure	If **true**, the node memory is low.
PIDPressure	If **true**, there are too many processes on the node.
OutOfDisk	If **true**, the node has insufficient free space on the node for adding new pods.
NetworkUnavailable	If **true**, the network for the node is not correctly configured.
NotReady	If **true**, 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.

6.1.2. 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 selected nodes:

  ```
  $ oc get pod --selector=<nodeSelector>
  ```

  ```
  $ oc get pod --selector=kubernetes.io/os
  ```

  Or:

  ```
  $ oc get pod -l=<nodeSelector>
  ```

  ```
  $ oc get pod -l kubernetes.io/os=linux
  ```

- To list all pods on a specific node, including terminated pods:

  ```
  $ oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename>
  ```

6.1.3. 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
  ```

  Example output

  ```
  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 **!=**.

6.2. USING THE NODE TUNING OPERATOR

Additional resources

Learn about the Node Tuning Operator and how you can use it to manage node-level tuning by orchestrating the tuned daemon.

The Node Tuning Operator helps you manage node-level tuning by orchestrating the TuneD daemon and achieves low latency performance by using the Performance Profile controller. 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 Dedicated as a Kubernetes daemon set. 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 uses the Performance Profile controller to implement automatic tuning to achieve low latency performance for OpenShift Dedicated applications.

The cluster administrator configures a performance profile to define node-level settings such as the following:

- Updating the kernel to kernel-rt.

- Choosing CPUs for housekeeping.

- Choosing CPUs for running workloads.

The Node Tuning Operator is part of a standard OpenShift Dedicated installation in version 4.1 and later.

> **NOTE**
>
> In earlier versions of OpenShift Dedicated, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Dedicated 4.11 and later, this functionality is part of the Node Tuning Operator.

6.2.1. Accessing an example Node Tuning Operator specification

Use this process to access an example Node Tuning Operator specification.

Procedure

- Run the following command to access an example Node Tuning Operator specification:

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

The default CR is meant for delivering standard node-level tuning for the OpenShift Dedicated platform and it can only be modified to set the Operator Management state. Any other 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 Dedicated nodes based on node or pod labels and profile priorities.

> **WARNING**
>
> While in certain situations the support for pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default Tuned CR ships without pod label matching. If a custom profile is created with pod label matching, then the functionality will be enabled at that time. The pod label functionality will be deprecated in future versions of the Node Tuning Operator.

6.2.2. 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.

Management state

The Operator Management state is set by adjusting the default Tuned CR. By default, the Operator is in the Managed state and the **spec.managementState** field is not present in the default Tuned CR. Valid values for the Operator Management state are as follows:

- Managed: the Operator will update its operands as configuration resources are updated

- Unmanaged: the Operator will ignore changes to the configuration resources

- Removed: the Operator will remove its operands and resources the Operator provisioned

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. The **recommend:** section is a list of items to recommend the profiles based on a selection criteria.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

The individual items of the list:

```
- machineConfigLabels: 1
    <mcLabels> 2
  match: 3
    <match> 4
  priority: <priority> 5
  profile: <tuned_profile_name> 6
  operand: 7
    debug: <bool> 8
    tunedConfig:
      reapply_sysctl: <bool> 9
```

1   Optional.

2   A dictionary of key/value **MachineConfig** labels. The keys must be unique.

3   If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.

4   An optional list.

5   Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).

6   A TuneD profile to apply on a match. For example **tuned_profile_1**.

7   Optional operand configuration.

8   Turn debugging on or off for the TuneD daemon. Options are **true** for on or **false** for off. The default is **false**.

9   Turn **reapply_sysctl** functionality on or off for the TuneD daemon. Options are **true** for on and **false** for off.

<match> is an optional list recursively defined as follows:

```
- label: <label_name>     ❶
  value: <label_value>     ❷
  type: <label_type>       ❸
    <match>                ❹
```

❶ Node or pod label name.

❷ Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.

❸ Optional object type (**node** or **pod**). If omitted, **node** is assumed.

❹ An optional **<match>** list.

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>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.

> **IMPORTANT**
>
> When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

Example: Node or pod label based matching

```
- 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.
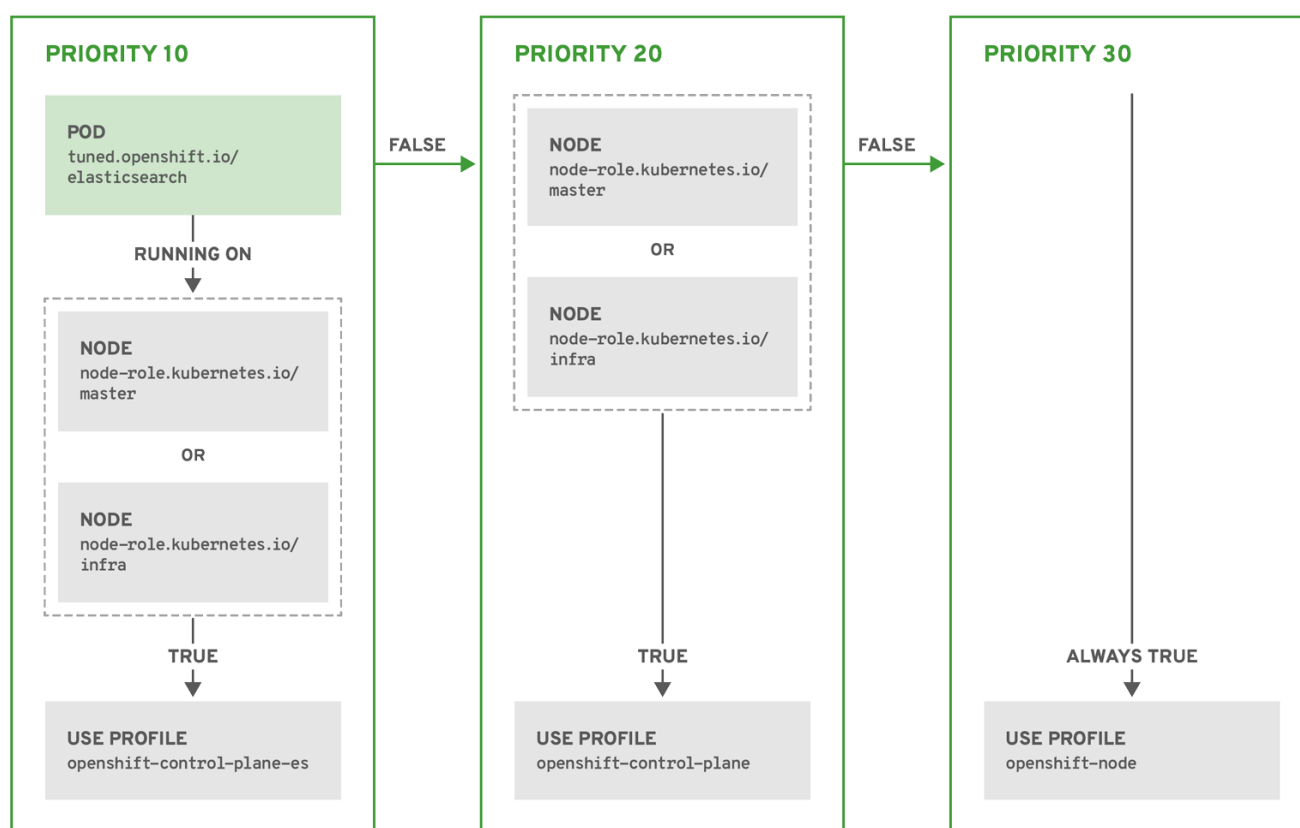
OPENSHIFT_10_0319

Example: Machine config pool based matching

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
```

```
      name: openshift-node-custom
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Custom OpenShift node profile with an additional kernel parameter
          include=openshift-node
          [bootloader]
          cmdline_openshift_node_custom=+skew_tick=1
        name: openshift-node-custom

      recommend:
      - machineConfigLabels:
          machineconfiguration.openshift.io/role: "worker-custom"
        priority: 20
        profile: openshift-node-custom
```

To minimize node reboots, label the target nodes with a label the machine config pool's node selector will match, then create the Tuned CR above and finally create the custom machine config pool itself.

Cloud provider–specific TuneD profiles

With this functionality, all Cloud provider–specific nodes can conveniently be assigned a TuneD profile specifically tailored to a given Cloud provider on a OpenShift Dedicated cluster. This can be accomplished without adding additional node labels or grouping nodes into machine config pools.

This functionality takes advantage of **spec.providerID** node object values in the form of **<cloud-provider>://<cloud-provider-specific-id>** and writes the file **/var/lib/ocp-tuned/provider** with the value **<cloud-provider>** in NTO operand containers. The content of this file is then used by TuneD to load **provider-<cloud-provider>** profile if such profile exists.

The **openshift** profile that both **openshift-control-plane** and **openshift-node** profiles inherit settings from is now updated to use this functionality through the use of conditional profile loading. Neither NTO nor TuneD currently include any Cloud provider–specific profiles. However, it is possible to create a custom profile **provider-<cloud-provider>** that will be applied to all Cloud provider–specific cluster nodes.

Example GCE Cloud provider profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
    name: provider-gce
```

> **NOTE**
>
> Due to profile inheritance, any setting specified in the **provider-<cloud-provider>** profile will be overwritten by the **openshift** profile and its child profiles.

6.2.3. Default profiles set on a cluster

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/ocp-tuned/provider},openshift
    name: openshift
  recommend:
  - profile: openshift-control-plane
    priority: 30
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
  - profile: openshift-node
    priority: 40
```

Starting with OpenShift Dedicated 4.9, all OpenShift TuneD profiles are shipped with the TuneD package. You can use the **oc exec** command to view the contents of these profiles:

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{,-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

6.2.4. Supported TuneD daemon plugins

Excluding the **[main]** section, the following TuneD plugins 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

- bootloader

There is some dynamic tuning functionality provided by some of these plugins that is not supported. The following TuneD plugins are currently not supported:

- script

- systemd

> **NOTE**
>
> The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

Additional resources

- Available TuneD Plugins

- Getting Started with TuneD

6.3. REMEDIATING, FENCING, AND MAINTAINING NODES

When node-level failures occur, such as the kernel hangs or network interface controllers (NICs) fail, the work required from the cluster does not decrease, and workloads from affected nodes need to be restarted somewhere. Failures affecting these workloads risk data loss, corruption, or both. It is important to isolate the node, known as **fencing**, before initiating recovery of the workload, known as **remediation**, and recovery of the node.

For more information on remediation, fencing, and maintaining nodes, see the Workload Availability for Red Hat OpenShift documentation.

CHAPTER 7. WORKING WITH CONTAINERS

7.1. UNDERSTANDING CONTAINERS

The basic units of OpenShift Dedicated 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 Dedicated and Kubernetes add the ability to orchestrate containers across multi-host installations.

7.1.1. About containers and RHEL kernel memory

Due to Red Hat Enterprise Linux (RHEL) behavior, a container on a node with high CPU usage might seem to consume more memory than expected. The higher memory consumption could be caused by the **kmem_cache** in the RHEL kernel. The RHEL kernel creates a **kmem_cache** for each cgroup. For added performance, the **kmem_cache** contains a **cpu_cache**, and a node cache for any NUMA nodes. These caches all consume kernel memory.

The amount of memory stored in those caches is proportional to the number of CPUs that the system uses. As a result, a higher number of CPUs results in a greater amount of kernel memory being held in these caches. Higher amounts of kernel memory in these caches can cause OpenShift Dedicated containers to exceed the configured memory limits, resulting in the container being killed.

To avoid losing containers due to kernel memory issues, ensure that the containers request sufficient memory. You can use the following formula to estimate the amount of memory consumed by the **kmem_cache**, where **nproc** is the number of processing units available that are reported by the **nproc** command. The lower limit of container requests should be this value plus the container memory requirements:

> $(nproc) X 1/2 MiB

7.1.2. About the container engine and container runtime

A *container engine* is a piece of software that processes user requests, including command-line options and image pulls. The container engine uses a *container runtime*, also called a *lower-level container runtime*, to run and manage the components required to deploy and operate containers. You likely will not need to interact with the container engine or container runtime.
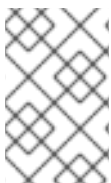
> **NOTE**
>
> The OpenShift Dedicated documentation uses the term *container runtime* to refer to the lower-level container runtime. Other documentation can refer to the container engine as the container runtime.

OpenShift Dedicated uses CRI-O as the container engine and crun or runC as the container runtime. The default container runtime is crun.

7.2. USING INIT CONTAINERS TO PERFORM TASKS BEFORE A POD IS DEPLOYED

OpenShift Dedicated 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.

7.2.1. 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.

7.2.2. 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**. After both containers complete, the pod begins.

Procedure

1. Create the pod for the Init Container:

   a. Create a YAML file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: myapp-pod
     labels:
       app: myapp
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     containers:
     - name: myapp-container
       image: registry.access.redhat.com/ubi9/ubi:latest
       command: ['sh', '-c', 'echo The app is running! && sleep 3600']
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           drop: [ALL]
     initContainers:
     - name: init-myservice
       image: registry.access.redhat.com/ubi9/ubi:latest
       command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for myservice; sleep 2; done;']
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           drop: [ALL]
     - name: init-mydb
       image: registry.access.redhat.com/ubi9/ubi:latest
       command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep 2; done;']
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           drop: [ALL]
   ```

   b. Create the pod:

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

   c. View the status of the pod:

   ```
   $ oc get pods
   ```

   Example output

   ```
   NAME            READY   STATUS     RESTARTS   AGE
   myapp-pod       0/1     Init:0/2   0          5s
   ```

The pod status, **Init:0/2**, indicates it is waiting for the two services.

2. Create the **myservice** service.

    a. Create a YAML file similar to the following:

    ```
    kind: Service
    apiVersion: v1
    metadata:
      name: myservice
    spec:
      ports:
      - protocol: TCP
        port: 80
        targetPort: 9376
    ```

    b. Create the pod:

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

    c. View the status of the pod:

    ```
    $ oc get pods
    ```

    Example output

    ```
    NAME                 READY   STATUS       RESTARTS  AGE
    myapp-pod            0/1     Init:1/2     0         5s
    ```

    The pod status, **Init:1/2**, indicates it is waiting for one service, in this case the **mydb** service.

3. Create the **mydb** service:

    a. Create a YAML file similar to the following:

    ```
    kind: Service
    apiVersion: v1
    metadata:
      name: mydb
    spec:
      ports:
      - protocol: TCP
        port: 80
        targetPort: 9377
    ```

    b. Create the pod:

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

    c. View the status of the pod:

    ```
    $ oc get pods
    ```

Example output

```
NAME            READY    STATUS          RESTARTS  AGE
myapp-pod       1/1      Running         0         2m
```

The pod status indicated that it is no longer waiting for the services and is running.

7.3. 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.

7.3.1. 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 Dedicated 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.

7.3.2. Working with volumes using the OpenShift Dedicated 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 deployment configs. 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 the **object_selection** parameter in the **oc set volume** command:

Table 7.1. Object Selection

Syntax	Description	Example
<object_type> <name>	Selects *<name>* of type *<object_type>*.	**deploymentConfig registry**

Syntax	Description	Example
<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** or **--remove** for the **operation** parameter 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.

7.3.3. 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> [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
  ```

- To list volume **v1** defined on all deployment configs:

  ```
  $ oc set volume dc --all --name=v1
  ```

7.3.4. 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 7.2. Supported Options for Adding Volumes

Option	Description	Default
--name	Name of the volume.	Automatically generated, if not specified.
-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. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host /**dev**/**pts** files. It is safe to mount the host by using /**host**.	
--path	Host path. Mandatory parameter for **--type=hostPath**. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host /**dev**/**pts** files. It is safe to mount the host by using /**host**.	

Option	Description	Default
--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 the **registry DeploymentConfig** object:

  ```
  $ oc set volume dc/registry --add
  ```

TIP

You can alternatively apply the following YAML to add the volume:

Example 7.1. Sample deployment config with an added volume

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: registry
  namespace: registry
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes: 1
        - name: volume-pppsw
          emptyDir: {}
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
```

1   Add the volume source **emptyDir**.

- To add volume **v1** with secret **secret1** for replication controller **r1** and mount inside the containers at */data*:

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='secret1' --mount-path=/data
```

TIP

You can alternatively apply the following YAML to add the volume:

Example 7.2. Sample replication controller with added volume and secret

```yaml
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes: 1
      - name: v1
        secret:
          secretName: secret1
          defaultMode: 420
      containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        volumeMounts: 2
        - name: v1
          mountPath: /data
```

1 Add the volume and secret.
2 Add the container mount path.

- 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** object on the server:

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
  --claim-name=pvc1 --mount-path=/data --containers=c1
```

TIP

You can alternatively apply the following YAML to add the volume:

Example 7.3. Sample deployment config with persistent volume added

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v1 ❶
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: ❷
            - name: v1
              mountPath: /data
```

❶ Add the persistent volume claim named `pvc1`.
❷ Add the container mount path.

- 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"
     }}'
```

7.3.5. 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
  ```

TIP

You can alternatively apply the following YAML to replace the volume:

Example 7.4. Sample replication controller with persistent volume claim named pvc1

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes:
        - name: v1 ❶
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - name: v1
              mountPath: /data
```

❶   Set persistent volume claim to **pvc1**.

- To change the **DeploymentConfig** object **d1** mount point to */opt* for volume **v1**:

  ```
  $ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
  ```

TIP

You can alternatively apply the following YAML to change the mount point:

Example 7.5. Sample deployment config with mount point set to **opt**.

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v2
          persistentVolumeClaim:
            claimName: pvc1
        - name: v1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 1
            - name: v1
              mountPath: /opt
```

1   Set the mount point to **/opt**.

7.3.6. 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 7.3. 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 the **DeploymentConfig** object **d1**:

  ```
  $ oc set volume dc/d1 --remove --name=v1
  ```

- To unmount volume **v1** from container **c1** for the **DeploymentConfig** object **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
  ```

7.3.7. Configuring volumes for multiple uses in a pod

You can configure a volume to share one volume for multiple uses in a single pod using the **volumeMounts.subPath** property to specify a **subPath** value inside a volume instead of the volume's root.

> **NOTE**
>
> You cannot add a **subPath** parameter to an existing scheduled pod.

Procedure

1. To view the list of files in the volume, run the **oc rsh** command:

   ```
   $ oc rsh <pod>
   ```

Example output

```
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. Specify the **subPath**:

Example Pod spec with subPath parameter

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
        subPath: mysql 1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: php
      image: php
      volumeMounts:
      - mountPath: /var/www/html
        name: site-data
        subPath: html 2
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    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.

7.4. 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.

7.4.1. 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, config maps, 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, config maps, and with downward API information, explicitly specifying paths for each item, so that I can have full control over the contents of that volume.

> **IMPORTANT**
>
> When the **RunAsUser** permission is set in the security context of a Linux-based pod, the projected files have the correct permissions set, including container user ownership. However, when the Windows equivalent **RunAsUsername** permission is set in a Windows pod, the kubelet is unable to correctly set ownership on the files in the projected volume.
>
> Therefore, the **RunAsUsername** permission set in the security context of a Windows pod is not honored for Windows projected volumes running in OpenShift Dedicated.

The following general scenarios show how you can use projected volumes.

Config map, secrets, Downward API.

Projected volumes allow you to deploy containers with configuration data that includes passwords. An application using these resources could be deploying Red Hat OpenStack Platform (RHOSP) 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 RHOSP configs.

Config map + secrets.

Projected volumes allow you to deploy containers involving configuration data and passwords. For example, you might execute a config map 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 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.

7.4.1.1. Example Pod specs

The following are examples of **Pod** specs for creating projected volumes.

Pod with a secret, a Downward API, and a config map

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts: 1
    - name: all-in-one
      mountPath: "/projected-volume" 2
      readOnly: true 3
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  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-username*.

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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

```
  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
```
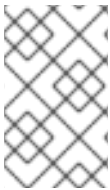
> **NOTE**
>
> The **defaultMode** can only be specified at the projected level and not for each volume source. However, as illustrated above, you can explicitly set the **mode** for each individual projection.

7.4.1.2. 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:
   securityContext:
     runAsNonRoot: true
     seccompProfile:
       type: RuntimeDefault
   containers:
   - name: container-test
     image: busybox
     volumeMounts:
     - name: all-in-one
       mountPath: "/projected-volume"
       readOnly: true
     securityContext:
       allowPrivilegeEscalation: false
       capabilities:
         drop: [ALL]
   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

7.4.2. Configuring a Projected Volume for a Pod

When creating projected volumes, consider the volume file path situations described in *Understanding projected volumes*.

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.

The user name and password values can be any valid string that is **base64** encoded.

The following example shows **admin** in base64:

```
$ echo -n "admin" | base64
```

Example output

```
YWRtaW4=
```

The following example shows the password **1f2d1e2e67df** in base64:

```
$ echo -n "1f2d1e2e67df" | base64
```

Example output

```
MWYyZDFlMmU2N2Rm
```

Procedure

To use a projected volume to mount an existing secret volume source.

1. Create the secret:

   a. Create a YAML file similar to the following, replacing the password and user information as appropriate:

      ```
      apiVersion: v1
      kind: Secret
      metadata:
        name: mysecret
      type: Opaque
      data:
        pass: MWYyZDFlMmU2N2Rm
        user: YWRtaW4=
      ```

   b. Use the following command to create the secret:

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

      For example:

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

      Example output

      ```
      secret "mysecret" created
      ```

   c. You can check that the secret was created using the following commands:

      ```
      $ oc get secret <secret-name>
      ```

      For example:

      ```
      $ oc get secret mysecret
      ```

      Example output

      ```
      NAME      TYPE     DATA    AGE
      mysecret  Opaque   2       17h
      ```

      ```
      $ oc get secret <secret-name> -o yaml
      ```

      For example:

      ```
      $ 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
```

2. Create a pod with a projected volume.

   a. Create a YAML file similar to the following, including a **volumes** section:

```
kind: Pod
metadata:
  name: test-projected-volume
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret  1
```

   1   The name of the secret you created.

   b. Create the pod from the configuration file:

```
$ oc create -f <your_yaml_file>.yaml
```

   For example:

```
$ oc create -f secret-pod.yaml
```

   Example output

```
pod "test-projected-volume" created
```

3. Verify that the pod container is running, and then watch for changes to the pod:

```
$ oc get pod <name>
```

For example:

```
$ oc get pod test-projected-volume
```

The output should appear similar to the following:

Example output

```
NAME                  READY    STATUS    RESTARTS  AGE
test-projected-volume  1/1      Running  0         14s
```

4. 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
```

5. In your shell, verify that the **projected-volumes** directory contains your projected sources:

```
/ # ls
```

Example output

```
bin         home           root         tmp
dev          proc          run          usr
etc         projected-volume  sys           var
```

7.5. 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 Dedicated. 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 plugin.

7.5.1. 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 plugin.

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.

7.5.2. Understanding how to consume container values using the downward API

You containers can consume API values using environment variables or a volume plugin. 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 plugin.

7.5.2.1. 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

1. Create a new pod spec that contains the environment variables you want the container to consume:

   a. Create a **pod.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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
         securityContext:
           allowPrivilegeEscalation: false
           capabilities:
             drop: [ALL]
     restartPolicy: Never
   # ...
   ```

   b. Create the pod from the **pod.yaml** file:

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

Verification

- Check the container's logs for the **MY_POD_NAME** and **MY_POD_NAMESPACE** values:

  ```
  $ oc logs -p dapi-env-test-pod
  ```

7.5.2.2. Consuming container values using a volume plugin

You containers can consume API values using a volume plugin.

Containers can consume:

- Pod name

- Pod project/namespace

- Pod annotations

- Pod labels

Procedure

To use the volume plugin:

1. Create a new pod spec that contains the environment variables you want the container to consume:

    a. Create a **volume-pod.yaml** file similar to the following:

    ```yaml
    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:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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
         securityContext:
           allowPrivilegeEscalation: false
           capabilities:
             drop: [ALL]
     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
  # ...
```

b. Create the pod from the **volume-pod.yaml** file:

```
$ oc create -f volume-pod.yaml
```

Verification

- Check the container's logs and verify the presence of the configured fields:

  ```
  $ oc logs -p dapi-volume-test-pod
  ```

  Example output

  ```
  cluster=downward-api-test-cluster1
  rack=rack-123
  zone=us-east-coast
  annotation1=345
  annotation2=456
  kubernetes.io/config.source=api
  ```

7.5.3. 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 plugin.

7.5.3.1. 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.

When creating the pod configuration, specify environment variables that correspond to the contents of the **resources** field in the **spec.container** field.

> **NOTE**
>
> If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

Procedure

1. Create a new pod spec that contains the resources you want to inject:

   a. Create a **pod.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   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
   # ...
   ```

   b. Create the pod from the **pod.yaml** file:

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

7.5.3.2. Consuming container resources using a volume plugin

When creating pods, you can use the Downward API to inject information about computing resource requests and limits using a volume plugin.

When creating the pod configuration, use the **spec.volumes.downwardAPI.items** field to describe the desired resources that correspond to the **spec.resources** field.

NOTE

If the resource limits are not included in the container configuration, the Downward API defaults to the node's CPU and memory allocatable values.

Procedure

1. Create a new pod spec that contains the resources you want to inject:

   a. Create a **pod.yaml** file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
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
# ...
```

b. Create the pod from the ***volume-pod.yaml*** file:

```
$ oc create -f volume-pod.yaml
```

7.5.4. 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 to inject:

   a. Create a **secret.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: mysecret
   data:
     password: <password>
     username: <username>
   type: kubernetes.io/basic-auth
   ```

   b. Create the secret object from the **secret.yaml** file:

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

2. Create a pod that references the **username** field from the above **Secret** object:

   a. Create a **pod.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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
         securityContext:
           allowPrivilegeEscalation: false
           capabilities:
   ```

```
        drop: [ALL]
    restartPolicy: Never
    # ...
```

b. Create the pod from the **pod.yaml** file:

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

Verification

- Check the container's logs for the **MY_SECRET_USERNAME** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.5. 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 config map with the values to inject:

    a. Create a *configmap.yaml* file similar to the following:

    ```
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: myconfigmap
    data:
      mykey: myvalue
    ```

    b. Create the config map from the **configmap.yaml** file:

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

2. Create a pod that references the above config map:

    a. Create a **pod.yaml** file similar to the following:

    ```
    apiVersion: v1
    kind: Pod
    metadata:
      name: dapi-env-test-pod
    spec:
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      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
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  restartPolicy: Always
# ...
```

b. Create the pod from the **pod.yaml** file:

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

Verification

- Check the container's logs for the **MY_CONFIGMAP_VALUE** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.6. 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 that references an existing environment variable:

   a. Create a **pod.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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)
         securityContext:
   ```

```
              allowPrivilegeEscalation: false
              capabilities:
                drop: [ALL]
        restartPolicy: Never
        # ...
```

b. Create the pod from the ***pod.yaml*** file:

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

Verification

- Check the container's logs for the **MY_ENV_VAR_REF_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.7. 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 that references an existing environment variable:

   a. Create a **pod.yaml** file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-env-test-pod
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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)
         securityContext:
           allowPrivilegeEscalation: false
           capabilities:
             drop: [ALL]
     restartPolicy: Never
     # ...
   ```

   b. Create the pod from the ***pod.yaml*** file:

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

Verification

- Check the container's logs for the **MY_NEW_ENV** value:

  ```
  $ oc logs -p dapi-env-test-pod
  ```

7.6. COPYING FILES TO OR FROM OPENSHIFT DEDICATED CONTAINERS

You can use the CLI to copy local files to or from a remote directory in a container using the **rsync** command.

7.6.1. 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>]
```

7.6.1.1. 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**.

7.6.2. 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 rsync**, 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.
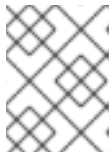
> **NOTE**
>
> In Windows, the **cwRsync** client should be installed and added to the PATH for use with the **oc rsync** command.

Procedure

- To copy a local directory to a pod directory:

  ```
  $ oc rsync <local-dir> <pod-name>:/<remote-dir> -c <container-name>
  ```

  For example:

  ```
  $ oc rsync /home/user/source devpod1234:/src -c user-container
  ```

- To copy a pod directory to a local directory:

  ```
  $ oc rsync devpod1234:/src /home/user/source
  ```

  Example output

  ```
  $ oc rsync devpod1234:/src/status.txt /home/user/
  ```

7.6.3. Using advanced Rsync features

The **oc rsync** command exposes fewer command-line options than standard **rsync**. In the case that you want to use a standard **rsync** command-line option that is not available in **oc rsync**, for example the **--exclude-from=FILE** option, it might 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_name> <local-dir> <pod-name>:/<remote-dir>
```

or:

Export the **RSYNC_RSH** variable:

```
$ export RSYNC_RSH='oc rsh'
```

Then, run the rsync command:

```
$ rsync --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

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**.

7.7. EXECUTING REMOTE COMMANDS IN AN OPENSHIFT DEDICATED CONTAINER

You can use the CLI to execute remote commands in OpenShift Dedicated containers.

7.7.1. 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
```

Example output

```
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.

7.7.2. 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 **HTTP/2**.

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.

7.8. USING PORT FORWARDING TO ACCESS APPLICATIONS IN A CONTAINER

OpenShift Dedicated supports port forwarding to pods.

7.8.1. 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 using 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 Dedicated handles port-forward requests from clients. Upon receiving a request, OpenShift Dedicated upgrades the response and waits for the client to create port-forwarding streams. When OpenShift Dedicated 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 Dedicated 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.

7.8.2. 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
  ```

  Example output

  ```
  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
  ```

  Example output

  ```
  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
  ```

  Example output

  ```
  Forwarding from 127.0.0.1:42390 -> 5000
  Forwarding from [::1]:42390 -> 5000
  ```

Or:

```
$ oc port-forward <pod> 0:5000
```

7.8.3. 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 Hyptertext Transfer Protocol Version 2 (HTTP/2).

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.

CHAPTER 8. WORKING WITH CLUSTERS

8.1. VIEWING SYSTEM EVENT INFORMATION IN OPENSHIFT DEDICATED CLUSTERS

You can view events in OpenShift Dedicated, which are based on events that happen to API objects in an OpenShift Dedicated cluster.

8.1.1. Understanding events

Review the following information to learn how OpenShift Dedicated uses *events* to record information about real-world events in a resource-agnostic manner. Events also allow developers and administrators to consume information about system components in a unified way.

8.1.2. Viewing events using the CLI

You can get a list of events in a given project by using the CLI.

Procedure

- View events in a project by using a command similar to the following:

  ```
  $ oc get events [-n <project>]
  ```

  where:

  project

  Specifies the name of the project.

  For example:

  ```
  $ oc get events -n openshift-config
  ```

  Example output

  ```
  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 "ovn-kubernetes": cannot set "ovn-kubernetes" 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
  #...
  ```

- View events in your project from the OpenShift Dedicated console:

  1. Launch the OpenShift Dedicated 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, also have an **Events** tab, which shows events related to that object.

8.1.3. List of events

Review the information in this section to learn about OpenShift Dedicated events.

Table 8.1. Configuration events

Name	Description
FailedValidation	Failed pod configuration validation.

Table 8.2. 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.
ExceededGrace Period	Container runtime did not stop the pod within specified grace period.

Table 8.3. Health events

Name	Description
Unhealthy	Container is unhealthy.

Table 8.4. Image events

Name	Description
BackOff	Back off Ctr Start, image pull.
ErrImageNeverPull	The image's **NeverPull Policy** is violated.
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 8.5. Image Manager events

Name	Description
FreeDiskSpaceFailed	Free disk space failed.
InvalidDiskCapacity	Invalid disk capacity.

Table 8.6. Node events

Name	Description
FailedMount	Volume mount failed.
HostNetworkNotSupported	Host network not supported.
HostPortConflict	Host/port conflict.
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 8.7. Pod worker events

Name	Description
FailedSync	Pod sync failed.

Table 8.8. System Events

Name	Description
SystemOOM	There is an OOM (out of memory) situation on the cluster.

Table 8.9. Pod events

Name	Description
FailedKillPod	Failed to stop a pod.
FailedCreatePodContainer	Failed to create a pod container.
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 8.10. 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.

Name	Description
SucceededGetScale	HPA controller was able to get the target's current scale.
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 8.11. 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.
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.

Name	Description
ProvisioningCleanupFailed	Error cleaning provisioned volume.
ProvisioningSucceeded	Occurs when the volume is provisioned successfully.
WaitForFirstConsumer	Delay binding until pod scheduling.

Table 8.12. Lifecycle hooks

Name	Description
FailedPostStartHook	Handler failed for pod start.
FailedPreStopHook	Handler failed for pre-stop.
UnfinishedPreStopHook	Pre-stop hook unfinished.

Table 8.13. Deployments

Name	Description
DeploymentCancellationFailed	Failed to cancel deployment.
DeploymentCancelled	Canceled deployment.
DeploymentCreated	Created new replication controller.
IngressIPRangeFull	No available Ingress IP to allocate to service.

Table 8.14. 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 8.15. Daemon set 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 8.16. 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\>**.

Name	Description
ExternalIP	Lists external IP address. For example, **Added: <external-ip>**.
UID	Lists the new UID. For example, **<old-service-uid> → <new-service-uid>**.
ExternalTrafficPolicy	Lists the new **ExternalTrafficPolicy**. For example, **<old-policy> → <new-policy>**.
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.

8.2. ESTIMATING THE NUMBER OF PODS YOUR OPENSHIFT DEDICATED NODES CAN HOLD

As a cluster administrator, you can use the OpenShift Cluster Capacity Tool to view the number of pods that can be scheduled in your cluster. This allows you 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.

8.2.1. Understanding the OpenShift Cluster Capacity Tool

Review the following information to learn how to use the OpenShift Cluster Capacity Tool to simulate a sequence of scheduling decisions that determine how many instances of an input pod can be scheduled on the cluster before the cluster is exhausted of resources.

> **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 the 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 OpenShift Cluster Capacity Tool as a stand-alone utility from the command line, or as a job in a pod inside an OpenShift Dedicated cluster. Running the tool as job inside of a pod enables you to run it multiple times without intervention.

8.2.2. Running the OpenShift Cluster Capacity Tool on the command line

You can run the OpenShift Cluster Capacity Tool from the command line to estimate the number of pods that can be scheduled onto your cluster.

You create a sample pod spec file, which the tool uses for estimating resource usage. The pod spec specifies its resource requirements as **limits** or **requests**. The cluster capacity tool takes the pod's resource requirements into account for its estimation analysis.

Prerequisites

1. Run the OpenShift Cluster Capacity Tool, which is available as a container image from the Red Hat Ecosystem Catalog. See the link in the "Additional resources" section.

2. Create a sample pod spec file:

   a. Create a YAML file similar to the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: small-pod
     labels:
       app: guestbook
       tier: frontend
   spec:
     securityContext:
       runAsNonRoot: true
       seccompProfile:
         type: RuntimeDefault
     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
   ```

```
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: [ALL]
```

b. Create the cluster role:

```
$ oc create -f <file_name>.yaml
```

For example:

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

Procedure

1. From the terminal, log in to the Red Hat Registry:

```
$ podman login registry.redhat.io
```

2. Pull the cluster capacity tool image:

```
$ podman pull registry.redhat.io/openshift4/ose-cluster-capacity
```

3. Run the cluster capacity tool:

```
$ podman run -v $HOME/.kube:/kube:Z -v $(pwd):/cc:Z  ose-cluster-capacity \
/bin/cluster-capacity --kubeconfig /kube/config --<pod_spec>.yaml /cc/<pod_spec>.yaml \
--verbose
```

where:

<pod_spec>.yaml

Specifies the pod spec to use.

verbose

Outputs a detailed description of how many pods can be scheduled on each node in the cluster.

Example output

```
small-pod pod requirements:
 - CPU: 150m
 - Memory: 100Mi

The cluster can schedule 88 instance(s) of the pod small-pod.

Termination reason: Unschedulable: 0/5 nodes are available: 2 Insufficient cpu,
3 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't
tolerate.

Pod distribution among nodes:
```

small-pod
- 192.168.124.214: 45 instance(s)
- 192.168.124.120: 43 instance(s)

In the above example, the number of estimated pods that can be scheduled onto the cluster is 88.

8.2.3. Running the OpenShift Cluster Capacity Tool as a job inside a pod

You can run the OpenShift Cluster Capacity Tool as a job inside of a pod by using a **ConfigMap** object. This allows you to run the tool multiple times without needing user intervention.

Prerequisites

- Download and install the OpenShift Cluster Capacity Tool from the **cluster-capacity** repository. See the link in the "Additional resources" section.

Procedure

1. Create the cluster role:

    a. Create a YAML file similar to the following:

    ```
    kind: ClusterRole
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: cluster-capacity-role
    rules:
    - apiGroups: [""]
      resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services",
    "replicationcontrollers"]
      verbs: ["get", "watch", "list"]
    - apiGroups: ["apps"]
      resources: ["replicasets", "statefulsets"]
      verbs: ["get", "watch", "list"]
    - apiGroups: ["policy"]
      resources: ["poddisruptionbudgets"]
      verbs: ["get", "watch", "list"]
    - apiGroups: ["storage.k8s.io"]
      resources: ["storageclasses"]
      verbs: ["get", "watch", "list"]
    ```

    b. Create the cluster role by running the following command:

    ```
    $ oc create -f <file_name>.yaml
    ```

    For example:

    ```
    $ oc create sa cluster-capacity-sa
    ```

2. Create the service account:

    ```
    $ oc create sa cluster-capacity-sa -n default
    ```

3. Add the role to the service account:

```
$ oc adm policy add-cluster-role-to-user cluster-capacity-role \
    system:serviceaccount:<namespace>:cluster-capacity-sa
```

where:

<namespace>

Specifies the namespace where the pod is located.

4. Define and create the pod spec:

a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  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
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

b. Create the pod by running the following command:

```
$ oc create -f <file_name>.yaml
```

For example:

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

5. Created a config map object by running the following command:

```
$ oc create configmap cluster-capacity-configmap \
    --from-file=pod.yaml=pod.yaml
```

The cluster capacity analysis is mounted in a volume using a config map object named **cluster-capacity-configmap** to mount the input pod spec file **pod.yaml** into a volume **test-volume** at the path **/test-pod**.

6. Create the job using the below example of a job specification file:

   a. Create a YAML file similar to the following:

   ```
   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
               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
   ```

   where:

   spec.template.spec.containers.env

   > Specifies a required environment variable that indicates the Cluster Capacity Tool is running inside a cluster as a pod.
   > The **pod.yaml** key of the **ConfigMap** object is the same as the **Pod** spec 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**.

   b. Run the cluster capacity image as a job in a pod by running the following command:

   ```
   $ oc create -f cluster-capacity-job.yaml
   ```

Verification

1. Check the job logs to find the number of pods that can be scheduled in the cluster:

```
$ oc logs jobs/cluster-capacity-job
```

Example output

```
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)
```

8.2.4. Additional resources

- [OpenShift Cluster Capacity Tool](#)

- [cluster-capacity repository](#)

8.3. RESTRICT RESOURCE CONSUMPTION WITH LIMIT RANGES

You can use limit ranges to restrict resource consumption for specific objects in a project.

By default, containers run with unbounded compute resources on an OpenShift Dedicated cluster.

You can configure resource consumption for the following objects:

- pods and containers: You can set minimum and maximum requirements for CPU and memory for pods and their containers.

- Image streams: You can set limits on the number of images and tags in an **ImageStream** object.

- Images: You can limit the size of images that can be pushed to an internal registry.

- Persistent volume claims (PVC): You can restrict the size of the PVCs that can be requested.

If a pod does not meet the constraints imposed by the limit range, the pod cannot be created in the namespace.

8.3.1. About limit ranges

You can set specific resource limits for a pod, container, image, image stream, or persistent volume claim (PVC) in a specific project by defining a **LimitRange** object. A limit range allows you to restrict resource consumption in that project.

All requests to create and modify resources are evaluated against each **LimitRange** object in the project. If the resource violates any of the enumerated constraints, the resource is rejected.

The following shows a limit range object for all components: pod, container, image, image stream, or PVC. You can configure limits for any or all of these components in the same object. You create a different limit range object for each project where you want to control resources.

Sample limit range object for a container

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

8.3.2. About component limits

Review the following examples to learn the limit range parameters for each component for when you create or edit a **LimitRange** object.

The examples are broken out for clarity. You can create a single **LimitRange** object for any or all components as necessary.

Container limits

A limit range allows you to specify the minimum and maximum CPU and memory that each container in a pod can request for a specific project. If a container is created in the project, the container CPU and memory requests in the **Pod** spec must comply with the values set in the **LimitRange** object. If not, the pod does not get created. The following requirements must hold true:

- The container CPU or memory request and limit must be greater than or equal to the **min** resource constraint for containers that are specified in the **LimitRange** object.

- The container CPU or memory request and limit must be less than or equal to the **max** resource constraint for containers that are specified in the **LimitRange** object.

If the **LimitRange** object defines a **max** CPU, you do not need to define a CPU **request** value in the **Pod** spec. But you must specify a CPU **limit** value that satisfies the maximum CPU constraint specified in the limit range. The following requirements must hold true:

- The ratio of the container limits to requests must be less than or equal to the **maxLimitRequestRatio** value for containers that is specified in the **LimitRange** object.

If the **LimitRange** object defines a **maxLimitRequestRatio** constraint, any new containers must have both a **request** and a **limit** value. OpenShift Dedicated calculates the limit-to-request ratio by dividing the **limit** by the **request**. This value should be a non-negative integer greater than 1.

For example, if a container has **cpu: 500** in the **limit** value, and **cpu: 100** in the **request** value, the limit-to-request ratio for **cpu** is **5**. This ratio must be less than or equal to the **maxLimitRequestRatio**.

If the **Pod** spec does not specify a container resource memory or limit, the **default** or **defaultRequest** CPU and memory values for containers specified in the limit range object are assigned to the container.

Container **LimitRange** object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.cpu

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

spec.limit.max.memory

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

spec.limit.min.cpu

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

spec.limit.min.memory

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

spec.limit.default.cpu

Specifies the default amount of CPU that a container can use if not specified in the **Pod** spec.

spec.limit.default.memory

Specifies the default amount of memory that a container can use if not specified in the **Pod** spec.

spec.limit.defaultRequest.cpu

Specifies the default amount of CPU that a container can request if not specified in the **Pod** spec.

spec.limit.defaultRequest.memory

Specifies the default amount of memory that a container can request if not specified in the **Pod** spec.

spec.limit.maxLimitRequestRatio.cpu

Specifies the maximum limit-to-request ratio for a container.

Pod limits

A limit range allows you to specify the minimum and maximum CPU and memory limits for all containers across a pod in a given project. To create a container in the project, the container CPU and memory requests in the **Pod** spec must comply with the values set in the **LimitRange** object. If not, the pod does not get created.

If the **Pod** spec does not specify a container resource memory or limit, the **default** or **defaultRequest** CPU and memory values for containers specified in the limit range object are assigned to the container.

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

- The container CPU or memory request and limit must be greater than or equal to the **min** resource constraints for pods that are specified in the **LimitRange** object.

- The container CPU or memory request and limit must be less than or equal to the **max** resource constraints for pods that are specified in the **LimitRange** object.

- The ratio of the container limits to requests must be less than or equal to the **maxLimitRequestRatio** constraint specified in the **LimitRange** object.

Pod **LimitRange** object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.cpu

Specifies the maximum amount of CPU that a pod can request across all containers.

spec.limit.max.memory

Specifies the maximum amount of memory that a pod can request across all containers.

spec.limit.min.cpu

Specifies the minimum amount of CPU that a pod can request across all containers.

spec.limit.min.memory

Specifies the minimum amount of memory that a pod can request across all containers.

spec.limit.maxLimitRequestRatio.cpu

Specifies the maximum limit-to-request ratio for a container.

Image limits

A limit range allows you to specify the maximum size of an image that can be pushed to an OpenShift image registry.
When pushing images to an OpenShift image registry, the following requirement must hold true:

- The size of the image must be less than or equal to the **max** size for images that is specified in the **LimitRange** object.

Image **LimitRange** object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi
```

where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.storage

Specifies the maximum size of an image that can be pushed to an OpenShift image registry.

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

Image stream limits

A limit range allows you to specify limits for image streams.
For each image stream, the following requirements must hold true:

- The number of image tags in an **ImageStream** specification must be less than or equal to the **openshift.io/image-tags** constraint in the **LimitRange** object.

- The number of unique references to images in an **ImageStream** specification must be less than or equal to the **openshift.io/images** constraint in the limit range object.

Imagestream **LimitRange** object definition

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

where

metadata.name

Specifies the name of the **LimitRange** object.

spec.limit.max.openshift.io/image-tags

Specifies the maximum number of unique image tags in the **imagestream.spec.tags** parameter in imagestream spec.

spec.limit.max.openshift.io/images

Specifies the maximum number of unique image references in the **imagestream.status.tags** parameter in the **imagestream** spec.

The **openshift.io/image-tags** resource represents unique image references. Possible references are an **ImageStreamTag**, an **ImageStreamImage** and a **DockerImage**. Tags can be created using the **oc tag** and **oc import-image** commands. No distinction is made between internal and external

references. However, each unique reference tagged in an **ImageStream** specification is counted just once. It does not restrict pushes to an internal container image registry in any way, but is useful for tag restriction.

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

Persistent volume claim limits

A limit range allows you to restrict the storage requested in a persistent volume claim (PVC). Across all persistent volume claims in a project, the following requirements must hold true:

- The resource request in a persistent volume claim (PVC) must be greater than or equal the **min** constraint for PVCs that is specified in the **LimitRange** object.

- The resource request in a persistent volume claim (PVC) must be less than or equal the **max** constraint for PVCs that is specified in the **LimitRange** object.

PVC **LimitRange** object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi"
      max:
        storage: "50Gi"
```

where:

metadata.name

Specifies the name of the **LimitRange** object.

spec.limit.min.storage

Specifies the minimum amount of storage that can be requested in a persistent volume claim.

spec.limit.max.storage

Specifies the maximum amount of storage that can be requested in a persistent volume claim.

8.3.3. Creating a Limit Range

You can define **LimitRange** objects to set specific resource limits for a pod, container, image, image stream, or persistent volume claim (PVC) in a specific project. A limit range allows you to restrict resource consumption in that project.

Procedure

1. Create a **LimitRange** object with your required specifications:

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
 limits:
   - type: "Pod"
     max:
       cpu: "2"
       memory: "1Gi"
     min:
       cpu: "200m"
       memory: "6Mi"
   - type: "Container"
     max:
       cpu: "2"
       memory: "1Gi"
     min:
       cpu: "100m"
       memory: "4Mi"
     default:
       cpu: "300m"
       memory: "200Mi"
     defaultRequest:
       cpu: "200m"
       memory: "100Mi"
     maxLimitRequestRatio:
       cpu: "10"
   - type: openshift.io/Image
     max:
       storage: 1Gi
   - type: openshift.io/ImageStream
     max:
       openshift.io/image-tags: 20
       openshift.io/images: 30
   - type: "PersistentVolumeClaim"
     min:
       storage: "2Gi"
     max:
       storage: "50Gi"
```

where:

metadata.name

Specifies a name for the **LimitRange** object.

spec.limit.type.Pod

Specifies limits for a pod, specify the minimum and maximum CPU and memory requests as needed.

spec.limit.type.Container

Specifies limits for a container, specify the minimum and maximum CPU and memory requests as needed.

spec.limit.type.default

For a container, specifies the default amount of CPU or memory that a container can use, if not specified in the **Pod** spec. This parameter is optional.

spec.limit.type.defaultRequest

For a container, specifies the default amount of CPU or memory that a container can request, if not specified in the **Pod** spec. This parameter is optional.

spec.limit.type.maxLimitRequestRatio

For a container, specifies the maximum limit-to-request ratio that can be specified in the **Pod** spec. This parameter is optional.

spec.limit.type.openshift.io/Image

Specifies limits for an image object. Set the maximum size of an image that can be pushed to an OpenShift image registry.

spec.limit.type.openshift.io/ImageStream

Specifies limits for an image stream. Set the maximum number of image tags and references that can be in the **ImageStream** object file, as needed.

spec.limit.type.openshift.io/PersistentVolueClaim

Specifies limits for a persistent volume claim. Set the minimum and maximum amount of storage that can be requested.

2. Create the object:

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

   where:

   <limit_range_file>

   Specifies the name of the YAML file you created.

   <project>

   Specifies the project where you want the limits to apply.

8.3.4. Viewing a limit

You can view the limits defined in a project by navigating in the web console to the project's **Quota** page. This allows you to see details about each of the limit ranges in a project.

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

Procedure

1. Get the list of **LimitRange** objects defined in the project. For example, for a project called **demoproject**:

   ```
   $ oc get limits -n demoproject
   ```

   ```
   NAME              CREATED AT
   resource-limits   2020-07-15T17:14:23Z
   ```

2. Describe the **LimitRange** object you are interested in, for example the **resource-limits** limit range:

```
$ oc describe limits resource-limits -n demoproject
```

```
Name:                     resource-limits
Namespace:                demoproject
Type                      Resource              Min    Max    Default Request Default Limit   Max
Limit/Request Ratio
----                      --------              ---    ---    --------------- -------------   -----------------------
Pod                       cpu                   200m   2      -               -               -
Pod                       memory                6Mi    1Gi    -               -               -
Container                 cpu                   100m   2      200m            300m            10
Container                 memory                4Mi    1Gi    100Mi           200Mi           -
openshift.io/Image        storage               -      1Gi    -               -               -
openshift.io/ImageStream  openshift.io/image    -      12     -               -               -
openshift.io/ImageStream  openshift.io/image-tags -    10     -               -               -
PersistentVolumeClaim     storage               -      50Gi   -               -               -
```

8.3.5. Deleting a Limit Range

You can remove any active **LimitRange** object so that it no longer enforces the limits in a project.

Procedure

- Run the following command:

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

8.4. CONFIGURING CLUSTER MEMORY TO MEET CONTAINER MEMORY AND RISK REQUIREMENTS

As a cluster administrator, you can manage application memory usage to help your clusters operate more efficiently.

You can perform any of the following tasks to manage application memory:

- Determine the memory and risk requirements of a containerized application component and configuring the container memory parameters to suit those requirements.

- Configure containerized application runtimes (for example, OpenJDK) to adhere optimally to the configured container memory parameters.

- Diagnose and resolve memory-related error conditions associated with running in a container.

8.4.1. Understanding how to manage application memory

You can review the following concepts to learn how OpenShift Dedicated manages compute resources so that you can lean how to keep your cluster running efficiently.

For each kind of resource (memory, CPU, storage), OpenShift Dedicated allows optional **request** and **limit** values to be placed on each container in a pod.

Note the following information about memory requests and memory limits:

- **Memory request**

- The memory request value, if specified, influences the OpenShift Dedicated 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 Dedicated prioritizes evicting its containers whose memory usage most exceeds their memory request. In serious cases of memory exhaustion, the node OOM killer might 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 can override the memory request values that a developer specifies, 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 Out of Memory (OOM) killer immediately selects and kills 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.

  - The minimum memory limit is 12 MB. If a container fails to start due to a **Cannot allocate memory** pod event, the memory limit is too low. Either increase or remove the memory limit. Removing the limit allows pods to consume unbounded node resources.

The steps for sizing application memory on OpenShift Dedicated are as follows:

1. **Determine expected container memory usage**
   Determine expected mean and peak container memory usage. For example, you could perform separate load testing. Remember to consider all the processes that could potentially run in parallel in the container, such as any ancillary scripts that might be spawned by the main application.

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 might be more appropriate to request memory according to the expected mean usage.

3. **Set container memory request**
   Set the container memory request based on the above. The request should represent the application memory usage as accurately as possible. 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 the 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. Setting a limit might make unanticipated excess memory usage obvious early (*fail fast*). However, setting a limit also terminates processes abruptly.

Note that some OpenShift Dedicated clusters might require a limit value to be set; some might 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 applications are tuned**
   Ensure your applications are 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.

8.4.2. Understanding OpenJDK settings for OpenShift Dedicated

You can review the following concepts to learn about how to deploy OpenJDK applications in your cluster effectively.

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:

Overriding the JVM maximum heap size

OpenJDK defaults to using a maximum of 25% of available memory (recognizing any container memory limits in place) for *heap* memory. This default value is conservative, and, in a properly-configured container environment, would result in 75% of the memory assigned to a container being mostly unused. A much higher percentage for the JVM to use for heap memory, such as 80%, is more suitable in a container context where memory limits are imposed on the container level. Most of the Red Hat containers include a startup script that replaces the OpenJDK default by setting updated values when the JVM launches.

For example, the Red Hat build of OpenJDK containers have a default value of 80%. This value can be set to a different percentage by defining the **JAVA_MAX_RAM_RATIO** environment variable.

For other OpenJDK deployements, the default value of 25% can be changed using the following command:

Example

```
$ java -XX:MaxRAMPercentage=80.0
```

Encouraging the JVM to release unused memory to the operating system, if appropriate

By default, the OpenJDK does not aggressively return unused memory to the operating system. This could 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.
Java-based agents can use 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.

Ensuring 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**, 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 Java-based agent image, the OpenShift Dedicated Jenkins Maven agent image sets the following variable:

```
JAVA_TOOL_OPTIONS="-Dsun.zip.disableMemoryMapping=true"
```

This does not guarantee that additional options are not required, but is intended to be a helpful starting point. Optimally tuning JVM workloads for running in a container is beyond the scope of this documentation, and may involve setting multiple additional JVM options.

8.4.3. Finding the memory request and limit from within a pod

You can configure your container to use the Downward API to dynamically discover its memory request and limit from within a pod. This allows your applications to better manage these resources without needing to use the API server.

Procedure

- Configure the pod to add the **MEMORY_REQUEST** and **MEMORY_LIMIT** stanzas:

    a. Create a YAML file similar to the following:

    ```
    apiVersion: v1
    kind: Pod
    metadata:
      name: test
    spec:
     securityContext:
       runAsNonRoot: false
       seccompProfile:
         type: RuntimeDefault
     containers:
    ```

```
    - name: test
      image: fedora:latest
      command:
      - sleep
      - "3600"
      env:
      - name: MEMORY_REQUEST
        valueFrom:
          resourceFieldRef:
            containerName: test
            resource: requests.memory
      - name: MEMORY_LIMIT
        valueFrom:
          resourceFieldRef:
            containerName: test
            resource: limits.memory
      resources:
        requests:
          memory: 384Mi
        limits:
          memory: 512Mi
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

where:

spec.consinters.env.name.MEMORY_REQUEST

This stanza discovers the application memory request value.

spec.consinters.env.name.MEMORY_LIMIT

This stanza discovers the application memory limit value.

b. Create the pod by running the following command:

```
$ oc create -f <file_name>.yaml
```

Verification

1. Access the pod using a remote shell:

```
$ oc rsh test
```

2. Check that the requested values were applied:

```
$ env | grep MEMORY | sort
```

Example output

```
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.

8.4.4. Understanding OOM kill policy

OpenShift Dedicated can 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.

If a process is Out of Memory (OOM) killed, the container could exit immediately. If the container PID 1 process receives the **SIGKILL**, the container does 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, use the following stepts to detect if an OOM kill occurred.

Procedure

1. Access the pod using a remote shell:

    ```
    # oc rsh <pod name>
    ```

2. Run the following command to see the current OOM kill count in **/sys/fs/cgroup/memory/memory.oom_control**:

    ```
    $ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
    ```

    Example output

    ```
    oom_kill 0
    ```

3. Run the following command to provoke an OOM kill:

    ```
    $ sed -e '' </dev/zero
    ```

    Example output

    ```
    Killed
    ```

4. Run the following command to see that the OOM kill counter in **/sys/fs/cgroup/memory/memory.oom_control** incremented:

    ```
    $ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
    ```

    Example output

    ```
    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 might

be restarted depending on the value of **restartPolicy**. If not restarted, controllers such as the replication controller will notice the pod's failed status and create a new pod to replace the old one.

Use the following command to get the pod status:

```
$ oc get pod test
```

Example output

```
NAME    READY   STATUS     RESTARTS  AGE
test    0/1     OOMKilled  0         1m
```

- If the pod has not restarted, run the following command to view the pod:

  ```
  $ oc get pod test -o yaml
  ```

  Example output

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: test
  # ...
  status:
    containerStatuses:
    - name: test
      ready: false
      restartCount: 0
      state:
        terminated:
          exitCode: 137
          reason: OOMKilled
    phase: Failed
  ```

- If restarted, run the following command to view the pod:

  ```
  $ oc get pod test -o yaml
  ```

  Example output

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: test
  # ...
  status:
    containerStatuses:
    - name: test
      ready: true
      restartCount: 1
      lastState:
        terminated:
  ```

```
        exitCode: 137
        reason: OOMKilled
    state:
      running:
  phase: Running
```

8.4.5. Understanding pod eviction

You can review the following concepts to learn the OpenShift Dedicated pod eviction policy.

OpenShift Dedicated can evict a pod from its node when the node's memory is exhausted. Depending on the extent of memory exhaustion, the eviction might or might 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 has phase **Failed** and reason **Evicted**. It is not restarted, regardless of the value of **restartPolicy**. However, controllers such as the replication controller will notice the pod's failed status and create a new pod to replace the old one.

```
$ oc get pod test
```

Example output

```
NAME    READY   STATUS    RESTARTS   AGE
test    0/1     Evicted   0          1m
```

```
$ oc get pod test -o yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: test
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
  phase: Failed
  reason: Evicted
```

8.5. CONFIGURING YOUR CLUSTER TO PLACE PODS ON OVERCOMMITTED NODES

OpenShift Dedicated administrators can manage container density on nodes by configuring pod placement behavior and per-project resource limits that overcommit cannot exceed.

Alternatively, administrators can disable project-level resource overcommitment on customer-created namespaces that are not managed by Red Hat.

For more information about container resource management, see Additional resources.

In an *overcommitted* state, the sum of the container compute resource requestsand limits exceeds the resources available on the system. For example, you might want to use overcommitment in development environments where a trade-off of guaranteed performance for capacity is acceptable.

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

The scheduler attempts to optimize the compute resource use across all nodes in your cluster. It places pods onto specific nodes, taking the pods' compute resource requests and nodes' available capacity into consideration.

8.5.1. Project-level limits

In OpenShift Dedicated, because overcommitment of project-level resources is enabled by default, if required by your use case, you can disable overcommitment on projects that are not managed by Red Hat.

For the list of projects that are managed by Red Hat and cannot be modified, see "Red Hat Managed resources" in *Support*.

8.5.1.1. Disabling overcommitment for a project

If required by your use case, you can disable overcommitment on any project that is not managed by Red Hat. For a list of projects that cannot be modified, see "Red Hat Managed resources" in *Support*.

Prerequisites

- You are logged in to the cluster using an account with cluster administrator or cluster editor permissions.

Procedure

1. Edit the namespace object file:

   a. If you are using the web console:

      i. Click **Administration → Namespaces** and click the namespace for the project.

      ii. In the **Annotations** section, click the **Edit** button.

      iii. Click **Add more** and enter a new annotation that uses a **Key** of **quota.openshift.io/cluster-resource-override-enabled** and a **Value** of **false**.

      iv. Click **Save**.

   b. If you are using the OpenShift CLI (**oc**):

      i. Edit the namespace:

         ```
         $ oc edit namespace/<project_name>
         ```

      ii. Add the following annotation:

         ```
         apiVersion: v1
         kind: Namespace
         ```

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

where:

metadata.annotations.quota.openshift.io/cluster-resource-override-enabled.false

Specifies that overcommit is disabled for this namespace.

8.5.2. Additional resources

- Restrict resource consumption with limit ranges