# OpenShift Dedicated 4

## Nodes

Configuring and managing nodes in OpenShift Dedicated 4

Last Updated: 2020-06-18

# OpenShift Dedicated 4 Nodes

Configuring and managing nodes in OpenShift Dedicated 4

## Abstract

This document provides instructions for working with the nodes, Pods, and containers in your cluster.

# Table of Contents

# CHAPTER 1. WORKING WITH PODS

## 1.1. USING PODS

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

### 1.1.1. Understanding pods

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

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

OpenShift 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.

**IMPORTANT**

The recommended maximum number of pods per OpenShift Dedicated node host is 35. You can have no more than 40 pods per node.

**WARNING**

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

### 1.1.2. Example pod configurations

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

**Pod Object Definition (YAML)**

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

```
metadata:
  name: example
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/example
  uid: 5cc30063-0265780783bc
  resourceVersion: '165032'
  creationTimestamp: '2019-02-13T20:31:37Z'
  labels:              1
    app: hello-openshift
  annotations:
    openshift.io/scc: anyuid
spec:
  restartPolicy: Always     2
  serviceAccountName: default
  imagePullSecrets:
    - name: default-dockercfg-5zrhb
  priority: 0
  schedulerName: default-scheduler
  terminationGracePeriodSeconds: 30
  nodeName: ip-10-0-140-16.us-east-2.compute.internal
  securityContext:     3
    seLinuxOptions:
      level: 's0:c11,c10'
  containers:          4
    - resources: {}
      terminationMessagePath: /dev/termination-log
      name: hello-openshift
      securityContext:
        capabilities:
          drop:
            - MKNOD
        procMount: Default
      ports:
        - containerPort: 8080
          protocol: TCP
      imagePullPolicy: Always
      volumeMounts:          5
        - name: default-token-wbqsl
          readOnly: true
          mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      terminationMessagePolicy: File
      image: registry.redhat.io/openshift4/ose-ogging-eventrouter:v4.3 6
  serviceAccount: default     7
  volumes:            8
    - name: default-token-wbqsl
      secret:
        secretName: default-token-wbqsl
        defaultMode: 420
  dnsPolicy: ClusterFirst
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
  - type: Ready
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: ContainersReady
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: PodScheduled
    status: 'True'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
hostIP: 10.0.140.16
startTime: '2019-02-13T20:31:37Z'
containerStatuses:
  - name: hello-openshift
    state:
      waiting:
        reason: ContainerCreating
    lastState: {}
    ready: false
    restartCount: 0
    image: openshift/hello-openshift
    imageID: ''
qosClass: BestEffort
```

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

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

3 OpenShift 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 Container definitions; in this case (as with most), just one.

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

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

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

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

account credentials.

NOTE

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

## 1.2. VIEWING PODS

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

### 1.2.1. About pods

OpenShift 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. Pods are the rough equivalent of a machine instance (physical or virtual) to a container.

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

### 1.2.2. Viewing pods in a project

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

#### Procedure

To view the pods in a project:

1. Change to the project:

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

2. Run the following command:

```
$ oc get pods
```

For example:

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

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

```
$ oc get pods -o wide

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

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

### 1.2.3. Viewing pod usage statistics

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

#### Prerequisites

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

#### Procedure

To view the usage statistics:

1. Run the following command:

   ```
   $ oc adm top pods
   ```

   For example:

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

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

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

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

## 1.3. AUTOMATICALLY SCALING PODS

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

### 1.3.1. Understanding horizontal pod autoscalers

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

IMPORTANT

Autoscaling for Memory Utilization is a Technology Preview feature only.

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

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

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

### 1.3.1.1. Supported metrics

The following metrics are supported by horizontal pod autoscalers:

**Table 1.1. Metrics**

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

IMPORTANT

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

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

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

## 1.3.2. Creating a horizontal pod autoscaler for CPU utilization

You can create a horizontal pod autoscaler (HPA) for an existing DeploymentConfig or ReplicationController object that automatically scales the Pods associated with that object in order to maintain the CPU usage you specify.

The HPA increases and decreases the number of replicas between the minimum and maximum numbers to maintain the specified CPU utilization across all Pods.

When autoscaling for CPU utilization, you can use the **oc autoscale** command and specify the minimum and maximum number of Pods you want to run at any given time and the average CPU utilization your Pods should target. If you do not specify a minimum, the Pods are given default values from the OpenShift Dedicated server. To autoscale for a specific CPU value, create a **HorizontalPodAutoscaler** object with the target CPU and Pod limits.

### Prerequisites

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

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

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

### Procedure

To create a horizontal pod autoscaler for CPU utilization:

1. Perform one of the following one of the following:

    - To scale based on the percent of CPU utilization, create a **HorizontalPodAutoscaler** object for an existing DeploymentConfig:

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

      (1) Specify the name of the DeploymentConfig. The object must exist.

2 Optionally, specify the minimum number of replicas when scaling down.

3 Specify the maximum number of replicas when scaling up.

4 Specify the target average CPU utilization over all the Pods, represented as a percent of requested CPU. If not specified or negative, a default autoscaling policy is used.

- To scale based on the percent of CPU utilization, create a **HorizontalPodAutoscaler** object for an existing ReplicationController:

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

1 Specify the name of the ReplicationController. The object must exist.

2 Specify the minimum number of replicas when scaling down.

3 Specify the maximum number of replicas when scaling up.

4 Specify the target average CPU utilization over all the Pods, represented as a percent of requested CPU. If not specified or negative, a default autoscaling policy is used.

- To scale for a specific CPU value, create a YAML file similar to the following for an existing DeploymentConfig or ReplicationController:

  a. Create a YAML file similar to the following:

```
apiVersion: autoscaling/v2beta2 1
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: v1 3
    kind: ReplicationController 4
    name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
  - type: Resource
    resource:
      name: cpu 9
      target:
        type: Utilization 10
        averageValue: 500Mi 11
```

1 Use the **autoscaling/v2beta2** API.

2 Specify a name for this horizontal pod autoscaler object.

3 Specify the API version of the object to scale:
   - For a ReplicationController, use **v1**,
   - For a DeploymentConfig, use **apps.openshift.io/v1**.

4 Specify the kind of object to scale, either **ReplicationController** or **DeploymentConfig**.

5 Specify the name of the object to scale. The object must exist.

6 Specify the minimum number of replicas when scaling down.

7 Specify the maximum number of replicas when scaling up.

8 Use the **metrics** parameter for memory utilization.

9 Specify **cpu** for CPU utilization.

10 Set to **Utilization**.

11 Set the type to **averageValue**.

b. Create the horizontal pod autoscaler:

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

2. Verify that the horizontal pod autoscaler was created:

```
$ oc get hpa hpa-resource-metrics-memory

NAME                         REFERENCE                      TARGETS        MINPODS
MAXPODS   REPLICAS   AGE
oc get hpa hpa-resource-metrics-memory   ReplicationController/example   2441216/500Mi
1         10         1          20m
```

For example, the following command creates a horizontal pod autoscaler that maintains between 3 and 7 replicas of the Pods that are controlled by the **image-registry** DeploymentConfig in order to maintain an average CPU utilization of 75% across all Pods.

```
$ oc autoscale dc/image-registry --min 3 --max 7 --cpu-percent=75
deploymentconfig "image-registry" autoscaled
```

The command creates a horizontal pod autoscaler with the following definition:

```
$ oc edit hpa frontend -n openshift-image-registry
```

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: "2020-02-21T20:19:28Z"
  name: image-registry
  namespace: default
  resourceVersion: "32452"
  selfLink: /apis/autoscaling/v1/namespaces/default/horizontalpodautoscalers/frontend
  uid: 1a934a22-925d-431e-813a-d00461ad7521
spec:
  maxReplicas: 7
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1
    kind: DeploymentConfig
    name: image-registry
  targetCPUUtilizationPercentage: 75
status:
  currentReplicas: 5
  desiredReplicas: 0
```

The following example shows autoscaling for the **image-registry** DeploymentConfig. The initial deployment requires 3 Pods. The HPA object increased that minimum to 5 and will increase the Pods up to 7 if CPU usage on the Pods reaches 75%:

```
$ oc get dc image-registry
NAME            REVISION   DESIRED   CURRENT   TRIGGERED BY
image-registry   1          3         3         config

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

$ oc get dc image-registry
NAME            REVISION   DESIRED   CURRENT   TRIGGERED BY
image-registry   1          5         5         config
```

### 1.3.3. Creating a horizontal pod autoscaler object for memory utilization

You can create a horizontal pod autoscaler (HPA) for an existing DeploymentConfig or ReplicationController object that automatically scales the Pods associated with that object in order to maintain the average memory utilization you specify, either a direct value or a percentage of requested memory.

The HPA increases and decreases the number of replicas between the minimum and maximum numbers to maintain the specified memory utilization across all Pods.

For memory utilization, you can specify the minimum and maximum number of Pods and the average memory utilization your Pods should target. If you do not specify a minimum, the Pods are given default values from the OpenShift Dedicated server.

IMPORTANT

Autoscaling for memory utilization is a Technology Preview feature only.

#### Prerequisites

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

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-129-223.compute.internal -n openshift-kube-scheduler
```

```
Name:         openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  scheduler
  Usage:
    Cpu:     2m
    Memory:  41056Ki
  Name:      wait-for-host-port
  Usage:
    Memory:  0
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2020-02-14T22:21:14Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Timestamp:             2020-02-14T22:21:14Z
Window:                5m0s
Events:                <none>
```

### Procedure

To create a horizontal pod autoscaler for memory utilization:

1. Create a YAML file for one of the following:

   - To scale for a specific memory value, create a **HorizontalPodAutoscaler** object similar to the following for an existing DeploymentConfig or ReplicationController:

```
apiVersion: autoscaling/v2beta2 (1)
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory (2)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: v1 (3)
    kind: ReplicationController (4)
    name: example (5)
  minReplicas: 1 (6)
  maxReplicas: 10 (7)
  metrics: (8)
  - type: Resource
    resource:
      name: memory (9)
      target:
        type: AverageValue (10)
        averageValue: 500Mi (11)
```

1. Use the **autoscaling/v2beta2** API.
2. Specify a name for this horizontal pod autoscaler object.
3. Specify the API version of the object to scale:
   - For a ReplicationController, use **v1**,
   - For a DeploymentConfig, use **apps.openshift.io/v1**.
4. Specify the kind of object to scale, either **ReplicationController** or **DeploymentConfig**.
5. Specify the name of the object to scale. The object must exist.
6. Specify the minimum number of replicas when scaling down.
7. Specify the maximum number of replicas when scaling up.
8. Use the **metrics** parameter for memory utilization.
9. Specify **memory** for memory utilization.
10. Set the type to **AverageValue**.
11. Specify **averageValue** and a specific memory value.

- To scale for a percentage, create a **HorizontalPodAutoscaler** object similar to the following:

```
apiVersion: autoscaling/v2beta2 1
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1 3
    kind: DeploymentConfig 4
    name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
  - type: Resource
    resource:
      name: memory 9
      target:
        type: Utilization 10
        averageUtilization: 50 11
```

1. Use the **autoscaling/v2beta2** API.
2. Specify a name for this horizontal pod autoscaler object.

3 Specify the API version of the object to scale:

- For a ReplicationController, use **v1**,
- For a DeploymentConfig, use **apps.openshift.io/v1**.

4 Specify the kind of object to scale, either **ReplicationController** or **DeploymentConfig**.

5 Specify the name of the object to scale. The object must exist.

6 Specify the minimum number of replicas when scaling down.

7 Specify the maximum number of replicas when scaling up.

8 Use the **metrics** parameter for memory utilization.

9 Specify **memory** for memory utilization.

10 Set to **Utilization**.

11 Specify **averageUtilization** and a target average memory utilization over all the Pods, represented as a percent of requested memory. The target pods must have memory requests configured.

2. Create the horizontal pod autoscaler:

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

For example:

```
$ oc create -f hpa.yaml

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

3. Verify that the horizontal pod autoscaler was created:

```
$ oc get hpa hpa-resource-metrics-memory

NAME                           REFERENCE                     TARGETS        MINPODS
MAXPODS   REPLICAS   AGE
oc get hpa hpa-resource-metrics-memory   ReplicationController/example   2441216/500Mi
1         10         1          20m
```

```
$ oc describe hpa hpa-resource-metrics-memory
Name:                        hpa-resource-metrics-memory
Namespace:                   default
Labels:                      <none>
Annotations:                 <none>
CreationTimestamp:           Wed, 04 Mar 2020 16:31:37 +0530
Reference:                   ReplicationController/example
Metrics:                     ( current / target )
  resource memory on pods:   2441216 / 500Mi
Min replicas:                1
```

```
Max replicas:              10
ReplicationController pods:  1 current / 1 desired
Conditions:
  Type           Status  Reason            Message
  ----           ------  ------            -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a
replica count from memory resource
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable
range
Events:
  Type     Reason                Age              From                  Message
  ----     ------                ----             ----                  -------
  Normal   SuccessfulRescale       6m34s             horizontal-pod-autoscaler  New size: 1;
reason: All metrics below target
```

## 1.3.4. Understanding horizontal pod autoscaler status conditions

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

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

The HPA responds with the following status conditions:

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

    ```
    $ oc describe hpa cm-test
    Name:                        cm-test
    Namespace:                     prom
    Labels:                      <none>
    Annotations:                   <none>
    CreationTimestamp:             Fri, 16 Jun 2017 18:09:22 +0000
    Reference:                     ReplicationController/cm-test
    Metrics:                     ( current / target )
      "http_requests" on pods:      66m / 500m
    Min replicas:                  1
    ```

```
Max replicas:                4
ReplicationController pods:    1 current / 1 desired
Conditions: (1)
  Type            Status    Reason              Message
  ----            ------    ------              -------
  AbleToScale       True      ReadyForNewScale    the last scale time was sufficiently old
as to warrant a new scale
  ScalingActive     True      ValidMetricFound    the HPA was able to successfully
calculate a replica count from pods metric http_request
  ScalingLimited    False     DesiredWithinRange  the desired replica count is within the
acceptable range
Events:
```

(1) The horizontal pod autoscaler status messages.

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

```
Conditions:
  Type         Status  Reason          Message
  ----         ------  ------          -------
  AbleToScale  False   FailedGetScale  the HPA controller was unable to get the target's current
scale: no matches for kind "ReplicationController" in group "apps"
Events:
  Type     Reason          Age               From                       Message
  ----     ------          ----              ----                       -------
  Warning  FailedGetScale  6s (x3 over 36s)  horizontal-pod-autoscaler  no matches for kind
"ReplicationController" in group "apps"
```

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

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

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

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

#### 1.3.4.1. Viewing horizontal pod autoscaler status conditions

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

NOTE

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

### Prerequisites

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

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

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

### Procedure

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

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

For example:

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

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

```
Name:                          cm-test
Namespace:                         prom
Labels:                       <none>
Annotations:                     <none>
```

```
CreationTimestamp:            Fri, 16 Jun 2017 18:09:22 +0000
Reference:                    ReplicationController/cm-test
Metrics:                      ( current / target )
  "http_requests" on pods:     66m / 500m
Min replicas:                  1
Max replicas:                  4
ReplicationController pods:    1 current / 1 desired
Conditions: (1)
  Type            Status    Reason            Message
  ----            ------    ------            -------
  AbleToScale      True     ReadyForNewScale    the last scale time was sufficiently old as to warrant
a new scale
  ScalingActive    True     ValidMetricFound    the HPA was able to successfully calculate a replica
count from pods metric http_request
  ScalingLimited   False    DesiredWithinRange  the desired replica count is within the acceptable
range
```

### 1.3.5. Additional resources

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

## 1.4. PROVIDING SENSITIVE DATA TO PODS

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

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

### 1.4.1. 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 plug-in or the system can use secrets to perform actions on behalf of a pod.

Key properties include:

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

**YAML Secret Object Definition**

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque (1)
```

```
data: 2
  username: dmFsdWUtMQ0K 3
  password: dmFsdWUtMg0KDQo=
stringData: 4
  hostname: myapp.mydomain.com 5
```

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

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

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

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

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

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

When creating secrets:

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

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

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

NOTE

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

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

### 1.4.1.2. Example secret configurations

The following are sample secret configuration files.

#### YAML Secret That Will Create Four Files

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

(1) File contains decoded values.

(2) File contains decoded values.

(3) File contains the provided string.

(4) File contains the provided data.

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

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

#### YAML of a Pod Populating Environment Variables with Secret Data

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

**YAML of a Build Config Populating Environment Variables with Secret Data**

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username
```

#### 1.4.1.3. Secret data keys

Secret keys must be in a DNS subdomain.

### 1.4.2. 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:

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

#### 1.4.2.1. 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 namespaces.

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

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

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

### 1.4.2.2. Creating an opaque secret

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

**Procedure**

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

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

1 Specifies an opaque secret.

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

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

Then:

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

### 1.4.3. 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, then the version of the secret will be used for the pod will not be defined.

NOTE

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

### 1.4.4. 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 specification configured for a service serving certificates secret.**

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

1 Specify the name for the certificate

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

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

#### 1.4.4.1. Generating signed certificates for use with secrets

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

**Procedure**

To create a *service serving certificate secret*:

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

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

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

3. Create the service:

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

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

```
$ oc get secrets

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

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

Type:  kubernetes.io/service-account-token

Data

ca.crt:     5802 bytes
namespace:  17 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Ii
wia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJvcGVuc2hpZnQtY29uc
29sZSIsImt1YmVyb
cnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJhYmE4Y2UyZC00MzVlLTExZTkt
OTg4YS0wZWI0ZTFiNGEz
OTYiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6b3BlbnNoaWZ
```

5. Edit your pod specification with that secret.

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

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

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

NOTE

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

### 1.4.5. Troubleshooting secrets

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

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

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

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

NOTE

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

# CHAPTER 2. USING JOBS AND DAEMONSETS

## 2.1. 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.

See the Kubernetes documentation for more information about jobs.

### 2.1.1. Understanding Jobs and CronJobs

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

There are two possible resource types that allow creating run-once objects in OpenShift 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.

**CronJob**

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

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

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

WARNING

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

### 2.1.2. Understanding how to create Jobs

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

- A pod template, which describes the pod that OpenShift 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.

#### 2.1.2.1. Understanding how to set a maximum duration for Jobs

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

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

#### 2.1.2.2. Understanding how to set a Job back off policy for pod failure

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

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

#### 2.1.2.3. Understanding how to configure a CronJob to remove artifacts

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

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

TIP

- Delete CronJobs that you no longer need:

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

  Doing this prevents them from generating unnecessary artifacts.

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

### 2.1.3. Known limitations

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

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

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

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

### 2.1.4. 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. Optionally, 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. Optionally, 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. Optionally, specify the maximum duration the job can run.
4. Optionally, 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.

1. Create the job:

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

NOTE

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

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

### 2.1.5. Creating CronJobs

You create a CronJob in OpenShift Dedicated by creating a job object.

#### Procedure

To create a CronJob:

1. Create a YAML file similar to the following:

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *"  1
  concurrencyPolicy: "Replace" 2
  startingDeadlineSeconds: 200 3
  suspend: true            4
  successfulJobsHistoryLimit: 3 5
  failedJobsHistoryLimit: 1     6
  jobTemplate:             7
    spec:
      template:
        metadata:
          labels:          8
            parent: "cronjobpi"
        spec:
          containers:
          - name: pi
            image: perl
            command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: OnFailure 9
```

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

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

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

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

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

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

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

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

8 Sets a label for jobs spawned by this CronJob.

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

NOTE

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

2. Create the CronJob:

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

NOTE

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

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

With **oc run**, the **--schedule** option accepts schedules in cron format.

When creating a CronJob, **oc run** only supports the **Never** or **OnFailure** restart policies (**--restart**).

# CHAPTER 3. WORKING WITH CONTAINERS

## 3.1. 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.

## 3.2. 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.

### 3.2.1. 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.

### 3.2.2. 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 DeploymentConfigs. You can also list volumes in pods or any object that has a pod template.

The **oc set volume** command uses the following general syntax:

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

**Object selection**

Specify one of the following for **object_seletion** in the **oc set volume** command:

Table 3.1. Object Selection

| Syntax | Description | Example |
|---|---|---|
| ***<object_type> <name>*** | Selects ***<name>*** of type ***<object_type>***. | **deploymentConfig registry** |
| ***<object_type>*/*<name>*** | Selects ***<name>*** of type ***<object_type>***. | **deploymentConfig/registry** |
| ***<object_type>*--selector=*<object_label_selector>*** | Selects resources of type ***<object_type>*** that matched the given label selector. | **deploymentConfig--selector="name=registry"** |
| ***<object_type>* --all** | Selects all resources of type ***<object_type>***. | **deploymentConfig --all** |
| **-f** or **--filename=*<file_name>*** | File name, directory, or URL to file to use to edit the resource. | **-f registry-deployment-config.json** |

Operation

Specify **--add**, **--remove**, or **--list** for **operation** in the **oc set volume** command.

Mandatory parameters

Any ***<mandatory_parameters>*** are specific to the selected operation and are discussed in later sections.

Options

Any ***<options>*** are specific to the selected operation and are discussed in later sections.

## 3.2.3. Listing volumes and volume mounts in a pod

You can list volumes and volume mounts in pods or pod templates:

### Procedure

To list volumes:

```
$ oc set volume <object_type>/<name> --list [options]
```

List volume supported options:

| Option | Description | Default |
|---|---|---|
| **--name** | Name of the volume. | |
| **-c, --containers** | Select containers by name. It can also take wildcard **'*'** that matches any character. | **'*'** |

For example:

- To list all volumes for pod **p1**:

```
$ oc set volume pod/p1 --list
```

- To list volume **v1** defined on all DeploymentConfigs:

```
$ oc set volume dc --all --name=v1
```

### 3.2.4. 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 3.2. 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. | |
| **--path** | Host path. Mandatory parameter for **--type=hostPath**. | |
| **--secret-name** | Name of the secret. Mandatory parameter for **--type=secret**. | |
| **--configmap-name** | Name of the configmap. Mandatory parameter for **--type=configmap**. | |

| Option | Description | Default |
|---|---|---|
| **--claim-name** | Name of the persistent volume claim. Mandatory parameter for **--type=persistentVolumeClaim**. | |
| **--source** | Details of volume source as a JSON string. Recommended if the desired volume source is not supported by **--type**. | |
| **-o, --output** | Display the modified objects instead of updating them on the server. Supported values: **json**, **yaml**. | |
| **--output-version** | Output the modified objects with the given version. | **api-version** |

For example:

- To add a new volume source **emptyDir** to DeploymentConfig **registry**:

```
$ oc set volume dc/registry --add
```

- To add volume **v1** with secret **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
```

- To add existing persistent volume **v1** with claim name **pvc1** to deployment configuration ***dc.json*** on disk, mount the volume on container **c1** at ***/data***, and update the DeploymentConfig on the server:

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
  --claim-name=pvc1 --mount-path=/data --containers=c1
```

- To add a volume **v1** based on Git repository **https://github.com/namespace1/project1** with revision **5125c45f9f563** for all replication controllers:

```
$ oc set volume rc --all --add --name=v1 \
  --source='{"gitRepo": {
                "repository": "https://github.com/namespace1/project1",
                "revision": "5125c45f9f563"
            }}'
```

### 3.2.5. Updating volumes and volume mounts in a pod

You can modify the volumes and volume mounts in a pod.

### Procedure

Updating existing volumes using the **--overwrite** option:

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

For example:

- To replace existing volume **v1** for replication controller **r1** with existing persistent volume claim **pvc1**:

  ```
  $ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --claim-
  name=pvc1
  ```

- To change DeploymentConfig **d1** mount point to ***/opt*** for volume **v1**:

  ```
  $ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
  ```

## 3.2.6. 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 3.3. Supported Options for Removing Volumes**

| Option | Description | Default |
|---|---|---|
| **--name** | Name of the volume. | |
| **-c, --containers** | Select containers by name. It can also take wildcard **'*'** that matches any character. | **'*'** |
| **--confirm** | Indicate that you want to remove multiple volumes at once. | |
| **-o, --output** | Display the modified objects instead of updating them on the server. Supported values: **json**, **yaml**. | |
| **--output-version** | Output the modified objects with the given version. | **api-version** |

For example:

- To remove a volume **v1** from DeploymentConfig **d1**:

```
$ oc set volume dc/d1 --remove --name=v1
```

- To unmount volume **v1** from container **c1** for DeploymentConfig **d1** and remove the volume **v1** if it is not referenced by any containers on **d1**:

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- To remove all volumes for replication controller **r1**:

```
$ oc set volume rc/r1 --remove --confirm
```

### 3.2.7. Configuring volumes for multiple uses in a pod

You can configure a volume to allows you to share one volume for multiple uses in a single pod using the **volumeMounts.subPath** property to specify a **subPath** inside a volume instead of the volume's root.

#### Procedure

1. View the list of files in the volume, run the **oc rsh** command:

```
$ oc rsh <pod>
sh-4.3$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. Specify the **subPath**:

**Example subPath Usage**

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
        subPath: mysql (1)
    - name: php
      image: php
      volumeMounts:
      - mountPath: /var/www/html
        name: site-data
        subPath: html (2)
    volumes:
    - name: site-data
      persistentVolumeClaim:
        claimName: my-site-data
```

(1) Databases are stored in the **mysql** folder.

2 HTML content is stored in the **html** folder.

## 3.3. 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.

### 3.3.1. Understanding projected volumes

Projected volumes can map any combination of these volume sources into a single directory, allowing the user to:

- automatically populate a single volume with the keys from multiple secrets, configmaps, and with downward API information, so that I can synthesize a single directory with various sources of information;
- populate a single volume with the keys from multiple secrets, configmaps, and with downward API information, explicitly specifying paths for each item, so that I can have full control over the contents of that volume.

The following general scenarios show how you can use projected volumes.

**ConfigMap, Secrets, Downward API.**

Projected volumes allow you to deploy containers with configuration data that includes passwords. An application using these resources could be deploying OpenStack on Kubernetes. The configuration data might have to be assembled differently depending on if the services are going to be used for production or for testing. If a pod is labeled with production or testing, the downward API selector **metadata.labels** can be used to produce the correct OpenStack configs.

**ConfigMap + Secrets.**

Projected volumes allow you to deploy containers involving configuration data and passwords. For example, you might execute a configmap with some sensitive encrypted tasks that are decrypted using a vault password file.

**ConfigMap + Downward API.**

Projected volumes allow you to generate a config including the pod name (available via the **metadata.name** selector). This application can then pass the pod name along with requests in order to easily determine the source without using IP tracking.

**Secrets + Downward API.**

Projected volumes allow you to use a secret as a public key to encrypt the namespace of the pod (available via the **metadata.namespace** selector). This example allows the operator to use the application to deliver the namespace information securely without using an encrypted transport.

#### 3.3.1.1. Example Pod Specifications

The following are examples of pod specifications for creating projected volumes.

**Pod with a secret, a downward API, and a configmap**

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts: (1)
    - name: all-in-one
      mountPath: "/projected-volume"(2)
      readOnly: true (3)
  volumes: (4)
  - name: all-in-one (5)
    projected:
      defaultMode: 0400 (6)
      sources:
      - secret:
          name: mysecret (7)
          items:
            - key: username
              path: my-group/my-username (8)
      - downwardAPI: (9)
          items:
            - path: "labels"
              fieldRef:
                fieldPath: metadata.labels
            - path: "cpu_limit"
              resourceFieldRef:
                containerName: container-test
                resource: limits.cpu
      - configMap: (10)
          name: myconfigmap
          items:
            - key: config
              path: my-group/my-config
              mode: 0777 (11)
```

1. Add a **volumeMounts** section for each container that needs the secret.
2. Specify a path to an unused directory where the secret will appear.
3. Set **readOnly** to **true**.
4. Add a **volumes** block to list each projected volume source.
5. Specify any name for the volume.

6 Set the execute permission on the files.

7 Add a secret. Enter the name of the secret object. Each secret you want to use must be listed.

8 Specify the path to the secrets file under the **mountPath**. Here, the secrets file is in ***/projected-volume/my-group/my-config***.

9 Add a Downward API source.

10 Add a ConfigMap source.

11 Set the mode for the specific projection

NOTE

If there are multiple containers in the pod, each container needs a **volumeMounts** section, but only one **volumes** section is needed.

**Pod with multiple secrets with a non-default permission mode set**

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      defaultMode: 0755
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/my-username
      - secret:
          name: mysecret2
          items:
            - key: password
              path: my-group/my-password
              mode: 511
```

NOTE

The **defaultMode** can only be specified at the projected level and not for each volume source. However, as illustrated above, you can explicitly set the **mode** for each individual projection.

#### 3.3.1.2. Pathing Considerations

**Collisions Between Keys when Configured Paths are Identical**

If you configure any keys with the same path, the pod spec will not be accepted as valid. In the following example, the specified path for **mysecret** and **myconfigmap** are the same:

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/data
      - configMap:
          name: myconfigmap
          items:
            - key: config
              path: my-group/data
```

Consider the following situations related to the volume file paths.

**Collisions Between Keys without Configured Paths**

The only run-time validation that can occur is when all the paths are known at pod creation, similar to the above scenario. Otherwise, when a conflict occurs the most recent specified resource will overwrite anything preceding it (this is true for resources that are updated after pod creation as well).

**Collisions when One Path is Explicit and the Other is Automatically Projected**

In the event that there is a collision due to a user specified path matching data that is automatically projected, the latter resource will overwrite anything preceding it as before

### 3.3.2. 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.

### Procedure

To use a projected volume to mount an existing Secret volume source.

1. Create files containing the secrets, entering the following, replacing the password and user information as appropriate:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
```

   The **user** and **pass** values can be any valid string that is **base64** encoded. The examples used here are base64 encoded values **user: admin**, **pass:1f2d1e2e67df**.

```
$ echo -n "admin" | base64
YWRtaW4=
$ echo -n "1f2d1e2e67df" | base64
MWYyZDFlMmU2N2Rm
```

2. Use the following command to create the secrets:

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

   For example:

```
$ oc create -f secret.yaml
secret "mysecret" created
```

3. You can check that the secret was created using the following commands:

```
$ oc get secret <secret-name>
$ oc get secret <secret-name> -o yaml
```

   For example:

```
$ oc get secret mysecret
NAME       TYPE     DATA     AGE
mysecret   Opaque   2        17h
```

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
  name: mysecret
  namespace: default
  resourceVersion: "2107"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque
```

4. Create a pod configuration file similar to the following that includes a **volumes** section:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:      1
          name: user
      - secret:      2
          name: pass
```

1 2 The name of the secret you created.

5. Create the pod from the configuration file:

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

For example:

```
$ oc create -f secret-pod.yaml
pod "test-projected-volume" created
```

6. Verify that the pod container is running, and then watch for changes to the Pod:

```
$ oc get pod <name>
```

The output should appear similar to the following:

```
$ oc get pod test-projected-volume
NAME                    READY     STATUS    RESTARTS   AGE
test-projected-volume   1/1       Running   0          14s
```

7. In another terminal, use the **oc exec** command to open a shell to the running container:

   ```
   $ oc exec -it <pod> <command>
   ```

   For example:

   ```
   $ oc exec -it test-projected-volume -- /bin/sh
   ```

8. In your shell, verify that the **projected-volumes** directory contains your projected sources:

   ```
   / # ls
   bin          home          root          tmp
   dev           proc         run           usr
   etc          projected-volume  sys           var
   ```

## 3.4. 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 plug-in.

### 3.4.1. Expose Pod information to Containers using the Downward API

The Downward API contains such information as the pod's name, project, and resource values. Containers can consume information from the downward API using environment variables or a volume plug-in.

Fields within the pod are selected using the **FieldRef** API type. **FieldRef** has two fields:

| Field | Description |
|---|---|
| **fieldPath** | The path of the field to select, relative to the pod. |
| **apiVersion** | The API version to interpret the **fieldPath** selector within. |

Currently, the valid selectors in the v1 API include:

| Selector | Description |
|---|---|
| **metadata.name** | The pod's name. This is supported in both environment variables and volumes. |
| **metadata.namespace** | The pod's namespace.This is supported in both environment variables and volumes. |
| **metadata.labels** | The pod's labels. This is only supported in volumes and not in environment variables. |

| Selector | Description |
|---|---|
| **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.

### 3.4.2. Understanding how to consume container values using the downward API

You containers can consume API values using environment variables or a volume plug-in. Depending on the method you choose, containers can consume:

- Pod name
- Pod project/namespace
- Pod annotations
- Pod labels

Annotations and labels are available using only a volume plug-in.

#### 3.4.2.1. Consuming container values using environment variables

When using a container's environment variables, use the **EnvVar** type's **valueFrom** field (of type **EnvVarSource**) to specify that the variable's value should come from a **FieldRef** source instead of the literal value specified by the **value** field.

Only constant attributes of the pod can be consumed this way, as environment variables cannot be updated once a process is started in a way that allows the process to be notified that the value of a variable has changed. The fields supported using environment variables are:

- Pod name
- Pod project/namespace

#### Procedure

To use environment variables

1. Create a ***pod.yaml*** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_POD_NAME** and **MY_POD_NAMESPACE** values:

```
$ oc logs -p dapi-env-test-pod
```

### 3.4.2.2. Consuming container values using a volume plug-in

You containers can consume API values using a volume plug-in.

Containers can consume:

- Pod name
- Pod project/namespace
- Pod annotations
- Pod labels

#### Procedure

To use the volume plug-in:

1. Create a ***volume-pod.yaml*** file:

```
kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  containers:
    - name: volume-test-container
      image: gcr.io/google_containers/busybox
      command: ["sh", "-c", "cat /tmp/etc/pod_labels /tmp/etc/pod_annotations"]
      volumeMounts:
        - name: podinfo
          mountPath: /tmp/etc
          readOnly: false
  volumes:
  - name: podinfo
    downwardAPI:
      defaultMode: 420
      items:
      - fieldRef:
          fieldPath: metadata.name
        path: pod_name
      - fieldRef:
          fieldPath: metadata.namespace
        path: pod_namespace
      - fieldRef:
          fieldPath: metadata.labels
        path: pod_labels
      - fieldRef:
          fieldPath: metadata.annotations
        path: pod_annotations
  restartPolicy: Never
```

2. Create the pod from the ***volume-pod.yaml*** file:

```
$ oc create -f volume-pod.yaml
```

3. Check the container’s logs and verify the presence of the configured fields:

```
$ oc logs -p dapi-volume-test-pod
cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api
```

### 3.4.3. Understanding how to consume container resources using the downward API

When creating pods, you can use the downward API to inject information about computing resource requests and limits so that image and application authors can correctly create an image for specific environments.

You can do this using environment variable or a volume plug-in.

#### 3.4.3.1. Consuming container resources using environment variables

When creating pods, you can use the downward API to inject information about computing resource requests and limits using environment variables.

**Procedure**

To use environment variables:

1. When creating a pod configuration, specify environment variables that correspond to the contents of the **resources** field in the **spec.container** field:

```
....
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox:1.24
      command: [ "/bin/sh", "-c", "env" ]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      env:
        - name: MY_CPU_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.cpu
        - name: MY_CPU_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.cpu
        - name: MY_MEM_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.memory
        - name: MY_MEM_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.memory
....
```

   If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

2. Create the pod from the ***pod.yaml*** file:

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

### 3.4.3.2. Consuming container resources using a volume plug-in

When creating pods, you can use the downward API to inject information about computing resource requests and limits using a volume plug-in.

#### Procedure

To use the Volume Plug-in:

1. When creating a pod configuration, use the **spec.volumes.downwardAPI.items** field to describe the desired resources that correspond to the **spec.resources** field:

```
....
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat /etc/cpu_limit;
fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e /etc/mem_limit ]]; then cat
/etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat /etc/mem_request; fi; sleep 5; done"]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        items:
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.cpu
          - path: "cpu_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.cpu
          - path: "mem_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.memory
          - path: "mem_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.memory
....
```

If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

2. Create the pod from the ***volume-pod.yaml*** file:

```
$ oc create -f volume-pod.yaml
```

### 3.4.4. Consuming secrets using the downward API

When creating pods, you can use the downward API to inject Secrets so image and application authors can create an image for specific environments.

#### Procedure

1. Create a ***secret.yaml*** file:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: cGFzc3dvcmQ=
  username: ZGV2ZWxvcGVy
type: kubernetes.io/basic-auth
```

2. Create a **Secret** from the secret.yaml file:

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

3. Create a ***pod.yaml*** file that references the **username** field from the above **Secret**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
  restartPolicy: Never
```

4. Create the pod from the ***pod.yaml*** file:

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

5. Check the container's logs for the **MY_SECRET_USERNAME** value:

```
$ oc logs -p dapi-env-test-pod
```

### 3.4.5. Consuming configuration maps using the downward API

When creating pods, you can use the downward API to inject configuration map values so image and application authors can create an image for specific environments.

#### Procedure

1. Create a ***configmap.yaml*** file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

2. Create a **ConfigMap** from the ***configmap.yaml*** file:

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

3. Create a ***pod.yaml*** file that references the above **ConfigMap**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_CONFIGMAP_VALUE
          valueFrom:
            configMapKeyRef:
              name: myconfigmap
              key: mykey
  restartPolicy: Always
```

4. Create the pod from the ***pod.yaml*** file:

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

5. Check the container's logs for the **MY_CONFIGMAP_VALUE** value:

```
$ oc logs -p dapi-env-test-pod
```

### 3.4.6. Referencing environment variables

When creating pods, you can reference the value of a previously defined environment variable by using the **$()** syntax. If the environment variable reference can not be resolved, the value will be left as the provided string.

#### Procedure

1. Create a ***pod.yaml*** file that references an existing **environment variable**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_EXISTING_ENV
          value: my_value
        - name: MY_ENV_VAR_REF_ENV
          value: $(MY_EXISTING_ENV)
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_ENV_VAR_REF_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

### 3.4.7. Escaping environment variable references

When creating a pod, you can escape an environment variable reference by using a double dollar sign. The value will then be set to a single dollar sign version of the provided value.

#### Procedure

1. Create a ***pod.yaml*** file that references an existing **environment variable**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$(SOME_OTHER_ENV)
  restartPolicy: Never
```

2. Create the pod from the ***pod.yaml*** file:

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

3. Check the container's logs for the **MY_NEW_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

## 3.5. COPYING FILES TO OR FROM AN OPENSHIFT DEDICATED CONTAINER

You can use the CLI to copy local files to or from a remote directory in a container using the **rsync** command.

### 3.5.1. 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>]
```

#### 3.5.1.1. 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**.

### 3.5.2. Copying files to and from containers

Support for copying local files to or from a container is built into the CLI.

#### Prerequisites

When working with **oc sync**, note the following:

**rsync must be installed**

The **oc rsync** command uses the local **rsync** tool if present on the client machine and the remote container.

If **rsync** is not found locally or in the remote container, a **tar** archive is created locally and sent to the container where the **tar** utility is used to extract the files. If **tar** is not available in the remote container, the copy will fail.

The **tar** copy method does not provide the same functionality as **oc rsync**. For example, **oc rsync** creates the destination directory if it does not exist and only sends files that are different between the source and the destination.

NOTE

In Windows, the **cwRsync** client should be installed and added to the PATH for use with the **oc rsync** command.

**Procedure**

- To copy a local directory to a pod directory:

  ```
  oc rsync <local-dir> <pod-name>:/<remote-dir>
  ```

  For example:

  ```
  $ oc rsync /home/user/source devpod1234:/src

  WARNING: cannot use rsync: rsync not available in container
  status.txt
  ```

- To copy a pod directory to a local directory:

```
$ oc rsync devpod1234:/src /home/user/source

oc rsync devpod1234:/src/status.txt /home/user/
WARNING: cannot use rsync: rsync not available in container
status.txt
```

### 3.5.3. Using advanced Rsync features

The **oc rsync** command exposes fewer command line options than standard **rsync**. In the case that you wish to use a standard **rsync** command line option which is not available in **oc rsync** (for example the **--exclude-from=FILE** option), it may be possible to use standard **rsync** 's **--rsh** (**-e**) option or **RSYNC_RSH** environment variable as a workaround, as follows:

```
$ rsync --rsh='oc rsh' --exclude-from=FILE SRC POD:DEST
```

or:

```
$ export RSYNC_RSH='oc rsh'
$ rsync --exclude-from=FILE SRC POD:DEST
```

Both of the above examples configure standard **rsync** to use **oc rsh** as its remote shell program to enable it to connect to the remote pod, and are an alternative to running **oc rsync**.

## 3.6. EXECUTING REMOTE COMMANDS IN AN OPENSHIFT DEDICATED CONTAINER

You can use the CLI to execute remote commands in an OpenShift Dedicated container.

### 3.6.1. Executing remote commands in containers

Support for remote container command execution is built into the CLI.

#### Procedure

To run a command in a container:

```
$ oc exec <pod> [-c <container>] <command> [<arg_1> ... <arg_n>]
```

For example:

```
$ oc exec mypod date
Thu Apr  9 02:21:53 UTC 2015
```

**IMPORTANT**

For security purposes, the **oc exec** command does not work when accessing privileged containers except when the command is executed by a **cluster-admin** user.

### 3.6.2. Protocol for initiating a remote command from a client

Clients initiate the execution of a remote command in a container by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the project of the target pod.
- **<pod>** is the name of the target pod.
- **<container>** is the name of the target container.
- **<command>** is the desired command to be executed.

For example:

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

Additionally, the client can add parameters to the request to indicate if:

- the client should send input to the remote container's command (stdin).
- the client's terminal is a TTY.
- the remote container's command should send output from stdout to the client.
- the remote container's command should send output from stderr to the client.

After sending an **exec** request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses **SPDY**.

The client creates one stream each for stdin, stdout, and stderr. To distinguish among the streams, the client sets the **streamType** header on the stream to one of **stdin**, **stdout**, or **stderr**.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the remote command execution request.

## 3.7. USING PORT FORWARDING TO ACCESS APPLICATIONS IN A CONTAINER

OpenShift Dedicated supports port forwarding to pods.

### 3.7.1. Understanding port forwarding

You can use the CLI to forward one or more local ports to a pod. This allows you to listen on a given or random port locally, and have data forwarded to and from given ports in the pod.

Support for port forwarding is built into the CLI:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

The CLI listens on each local port specified by the user, forwarding via the protocol described below.

Ports may be specified using the following formats:

| | |
|---|---|
| **5000** | The client listens on port 5000 locally and forwards to 5000 in the pod. |
| **6000:5000** | The client listens on port 6000 locally and forwards to 5000 in the pod. |
| **:5000** or **0:5000** | The client selects a free local port and forwards to 5000 in the pod. |

OpenShift 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.

### 3.7.2. Using port forwarding

You can use the CLI to port-forward one or more local ports to a pod.

#### Procedure

Use the following command to listen on the specified port in a pod:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

For example:

- Use the following command to listen on ports **5000** and **6000** locally and forward data to and from ports **5000** and **6000** in the pod:

  ```
  $ oc port-forward <pod> 5000 6000

  Forwarding from 127.0.0.1:5000 -> 5000
  Forwarding from [::1]:5000 -> 5000
  Forwarding from 127.0.0.1:6000 -> 6000
  Forwarding from [::1]:6000 -> 6000
  ```

- Use the following command to listen on port **8888** locally and forward to **5000** in the pod:

  ```
  $ oc port-forward <pod> 8888:5000

  Forwarding from 127.0.0.1:8888 -> 5000
  Forwarding from [::1]:8888 -> 5000
  ```

- Use the following command to listen on a free port locally and forward to **5000** in the pod:

  ```
  $ oc port-forward <pod> :5000

  Forwarding from 127.0.0.1:42390 -> 5000
  Forwarding from [::1]:42390 -> 5000
  ```

  Or:

  ```
  $ oc port-forward <pod> 0:5000
  ```

### 3.7.3. Protocol for initiating port forwarding from a client

Clients initiate port forwarding to a pod by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the namespace of the target pod.
- **<pod>** is the name of the target pod.

For example:

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

After sending a port forward request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses SPDY.

The client creates a stream with the **port** header containing the target port in the pod. All data written to the stream is delivered via the Kubelet to the target pod and port. Similarly, all data sent from the pod for that forwarded connection is delivered back to the same stream in the client.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the port forwarding request.

## 3.8. USING SYSCTLS IN CONTAINERS

Sysctl settings are exposed via Kubernetes, allowing users to modify certain kernel parameters at runtime for namespaces within a container. Only sysctls that are namespaced can be set independently on pods. If a sysctl is not namespaced, called *node-level*, it cannot be set within OpenShift Dedicated. Moreover, only those sysctls considered *safe* are whitelisted by default; you can manually enable other *unsafe* sysctls on the node to be available to the user.

### 3.8.1. About sysctls

In Linux, the sysctl interface allows an administrator to modify kernel parameters at runtime. Parameters are available via the ***/proc/sys/*** virtual process file system. The parameters cover various subsystems, such as:

- kernel (common prefix: ***kernel.***)
- networking (common prefix: ***net.***)
- virtual memory (common prefix: ***vm.***)
- MDADM (common prefix: ***dev.***)

More subsystems are described in Kernel documentation. To get a list of all parameters, run:

```
$ sudo sysctl -a
```

#### 3.8.1.1. Namespaced versus node-level sysctls

A number of sysctls are *namespaced* in the Linux kernels. This means that you can set them independently for each pod on a node. Being namespaced is a requirement for sysctls to be accessible in a pod context within Kubernetes.

The following sysctls are known to be namespaced:

- ***kernel.shm****
- ***kernel.msg****
- ***kernel.sem***
- ***fs.mqueue.****

Additionally, most of the sysctls in the **net.*** group are known to be namespaced. Their namespace adoption differs based on the kernel version and distributor.

Sysctls that are not namespaced are called *node-level* and must be set manually by the cluster administrator, either by means of the underlying Linux distribution of the nodes, such as by modifying the ***/etc/sysctls.conf*** file, or by using a DaemonSet with privileged containers.

NOTE

Consider marking nodes with special sysctls as tainted. Only schedule pods onto them that need those sysctl settings. Use the taints and toleration feature to mark the nodes.

#### 3.8.1.2. Safe versus unsafe sysctls

Sysctls are grouped into *safe* and *unsafe* sysctls.

For a sysctl to be considered safe, it must use proper namespacing and must be properly isolated between pods on the same node. This means that if you set a sysctl for one pod it must not:

- Influence any other pod on the node
- Harm the node's health
- Gain CPU or memory resources outside of the resource limits of a pod

OpenShift Dedicated supports, or whitelists, the following sysctls in the safe set:

- ***kernel.shm_rmid_forced***
- ***net.ipv4.ip_local_port_range***
- ***net.ipv4.tcp_syncookies***

All safe sysctls are enabled by default. You can use a sysctl in a pod by modifying the pod specification.

Any sysctl not whitelisted by OpenShift Dedicated is considered unsafe for OpenShift Dedicated. Note that being namespaced alone is not sufficient for the sysctl to be considered safe.

All unsafe sysctls are disabled by default, and the cluster administrator must manually enable them on a per-node basis. Pods with disabled unsafe sysctls are scheduled but do not launch.

```
$ oc get pod

NAME        READY   STATUS            RESTARTS   AGE
hello-pod   0/1     SysctlForbidden   0          14s
```

### 3.8.2. Setting sysctls for a pod

You can set sysctls on pods using the pod's **securityContext**. The **securityContext** applies to all containers in the same pod.

Safe sysctls are allowed by default. A pod with unsafe sysctls fails to launch on any node unless the cluster administrator explicitly enables unsafe sysctls for that node. As with node-level sysctls, use the taints and toleration feature or labels on nodes to schedule those pods onto the right nodes.

The following example uses the pod **securityContext** to set a safe sysctl **kernel.shm_rmid_forced** and two unsafe sysctls, **net.ipv4.route.min_pmtu** and **kernel.msgmax**. There is no distinction between *safe* and *unsafe* sysctls in the specification.

**WARNING**

To avoid destabilizing your operating system, modify sysctl parameters only after you understand their effects.

### Procedure

To use safe and unsafe sysctls:

1. Modify the YAML file that defines the pod and add the **securityContext** spec, as shown in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
spec:
  securityContext:
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.ipv4.route.min_pmtu
      value: "552"
    - name: kernel.msgmax
      value: "65536"
  ...
```

2. Create the pod:

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

If the unsafe sysctls are not allowed for the node, the pod is scheduled, but does not deploy:

```
$ oc get pod

NAME        READY   STATUS            RESTARTS   AGE
hello-pod   0/1     SysctlForbidden   0          14s
```

## 3.8.3. Enabling unsafe sysctls

A cluster administrator can allow certain unsafe sysctls for very special situations such as high-performance or real-time application tuning.

If you want to use unsafe sysctls, a cluster administrator must enable them individually for a specific type of node. The sysctls must be namespaced.

**WARNING**

Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems, such as improper behavior of containers, resource shortage, or breaking a node.

**Procedure**

1. Add a label to the MachineConfigPool where the containers where containers with the unsafe sysctls will run:

```
$ oc edit machineconfigpool worker
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: sysctl (1)
```

(1) Add a **key: pair** label.

2. Create a KubeletConfig Custom Resource (CR):

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl (1)
  kubeletConfig:
    allowedUnsafeSysctls: (2)
      - "kernel.msg*"
      - "net.ipv4.route.min_pmtu"
```

(1) Specify the label from the MachineConfigPool.

(2) List the unsafe sysctls you want to allow.

3. Create the object:

```
$ oc apply -f set-sysctl-worker.yaml
```

A new MachineConfig named in the **99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet** format is created.

4. Wait for the cluster to reboot usng the **machineconfigpool** object **status** fields:
   For example:

   ```
   status:
     conditions:
       - lastTransitionTime: '2019-08-11T15:32:00Z'
         message: >-
           All nodes are updating to
           rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
         reason: ''
         status: 'True'
         type: Updating
   ```

   A message similar to the following appears when the cluster is ready:

   ```
       - lastTransitionTime: '2019-08-11T16:00:00Z'
         message: >-
           All nodes are updated with
           rendered-worker-ccbfb5d2838d65013ab36300b7b3dc13
         reason: ''
         status: 'True'
         type: Updated
   ```

5. When the cluster is ready, check for the merged **KubeletConfig** in the new MachineConfig:

   ```
   $ oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep
   ownerReference -A7
   ```

   ```
           "ownerReferences": [
               {
                   "apiVersion": "machineconfiguration.openshift.io/v1",
                   "blockOwnerDeletion": true,
                   "controller": true,
                   "kind": "KubeletConfig",
                   "name": "custom-kubelet",
                   "uid": "3f64a766-bae8-11e9-abe8-0a1a2a4813f2"
   ```

   You can now add unsafe sysctls to pods as needed.

# CHAPTER 4. WORKING WITH CLUSTERS

## 4.1. VIEWING SYSTEM EVENT INFORMATION IN AN OPENSHIFT DEDICATED CLUSTER

Events in OpenShift Dedicated are modeled based on events that happen to API objects in an OpenShift Dedicated cluster.

### 4.1.1. Understanding events

Events allow OpenShift Dedicated to record information about real-world events in a resource-agnostic manner. They also allow developers and administrators to consume information about system components in a unified way.

### 4.1.2. Viewing events using the CLI

You can get a list of events in a given project using the CLI.

**Procedure**

- To view events in a project use the following command:

  ```
  $ oc get events [-n <project>] (1)
  ```

  (1) The name of the project.

  For example:

  ```
  $ oc get events -n openshift-config

  LAST SEEN   TYPE      REASON                   OBJECT                      MESSAGE
  97m         Normal    Scheduled                pod/dapi-env-test-pod       Successfully assigned
  openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
  97m         Normal    Pulling                  pod/dapi-env-test-pod       pulling image
  "gcr.io/google_containers/busybox"
  97m         Normal    Pulled                   pod/dapi-env-test-pod       Successfully pulled image
  "gcr.io/google_containers/busybox"
  97m         Normal    Created                  pod/dapi-env-test-pod       Created container
  9m5s        Warning   FailedCreatePodSandBox   pod/dapi-volume-test-pod    Failed create
  pod sandbox: rpc error: code = Unknown desc = failed to create pod network sandbox
  k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
  0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9e22
  ): Multus: Err adding pod to network "openshift-sdn": cannot set "openshift-sdn" ifname to
  "eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or directory
  8m31s       Normal    Scheduled                pod/dapi-volume-test-pod    Successfully assigned
  openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal
  ```

- To view events in your project from the OpenShift 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, have their own **Events** tab as well, which shows events related to that object.

### 4.1.3. List of events

This section describes the events of OpenShift Dedicated.

Table 4.1. Configuration Events

| Name | Description |
|---|---|
| **FailedValidation** | Failed pod configuration validation. |

Table 4.2. Container Events

| Name | Description |
|---|---|
| **BackOff** | Back-off restarting failed the container. |
| **Created** | Container created. |
| **Failed** | Pull/Create/Start failed. |
| **Killing** | Killing the container. |
| **Started** | Container started. |
| **Preempting** | Preempting other pods. |
| **ExceededGracePeriod** | Container runtime did not stop the pod within specified grace period. |

Table 4.3. Health Events

| Name | Description |
|---|---|
| **Unhealthy** | Container is unhealthy. |

Table 4.4. Image Events

| Name | Description |
|---|---|
| **BackOff** | Back off Ctr Start, image pull. |
| **ErrImageNeverPull** | The image’s **NeverPull Policy** is violated. |

| Name | Description |
|---|---|
| **Failed** | Failed to pull the image. |
| **InspectFailed** | Failed to inspect the image. |
| **Pulled** | Successfully pulled the image or the container image is already present on the machine. |
| **Pulling** | Pulling the image. |

Table 4.5. Image Manager Events

| Name | Description |
|---|---|
| **FreeDiskSpaceFailed** | Free disk space failed. |
| **InvalidDiskCapacity** | Invalid disk capacity. |

Table 4.6. Node Events

| Name | Description |
|---|---|
| **FailedMount** | Volume mount failed. |
| **HostNetworkNotSupported** | Host network not supported. |
| **HostPortConflict** | Host/port conflict. |
| **InsufficientFreeCPU** | Insufficient free CPU. |
| **InsufficientFreeMemory** | Insufficient free memory. |
| **KubeletSetupFailed** | Kubelet setup failed. |
| **NilShaper** | Undefined shaper. |
| **NodeNotReady** | Node is not ready. |
| **NodeNotSchedulable** | Node is not schedulable. |

| Name | Description |
|---|---|
| **NodeReady** | Node is ready. |
| **NodeSchedulable** | Node is schedulable. |
| **NodeSelectorMismatching** | Node selector mismatch. |
| **OutOfDisk** | Out of disk. |
| **Rebooted** | Node rebooted. |
| **Starting** | Starting kubelet. |
| **FailedAttachVolume** | Failed to attach volume. |
| **FailedDetachVolume** | Failed to detach volume. |
| **VolumeResizeFailed** | Failed to expand/reduce volume. |
| **VolumeResizeSuccessful** | Successfully expanded/reduced volume. |
| **FileSystemResizeFailed** | Failed to expand/reduce file system. |
| **FileSystemResizeSuccessful** | Successfully expanded/reduced file system. |
| **FailedUnMount** | Failed to unmount volume. |
| **FailedMapVolume** | Failed to map a volume. |
| **FailedUnmapDevice** | Failed unmaped device. |
| **AlreadyMountedVolume** | Volume is already mounted. |
| **SuccessfulDetachVolume** | Volume is successfully detached. |

| Name | Description |
|---|---|
| **SuccessfulMountVolume** | Volume is successfully mounted. |
| **SuccessfulUnMountVolume** | Volume is successfully unmounted. |
| **ContainerGCFailed** | Container garbage collection failed. |
| **ImageGCFailed** | Image garbage collection failed. |
| **FailedNodeAllocatableEnforcement** | Failed to enforce System Reserved Cgroup limit. |
| **NodeAllocatableEnforced** | Enforced System Reserved Cgroup limit. |
| **UnsupportedMountOption** | Unsupported mount option. |
| **SandboxChanged** | Pod sandbox changed. |
| **FailedCreatePodSandBox** | Failed to create pod sandbox. |
| **FailedPodSandBoxStatus** | Failed pod sandbox status. |

Table 4.7. Pod Worker Events

| Name | Description |
|---|---|
| **FailedSync** | Pod sync failed. |

Table 4.8. System Events

| Name | Description |
|---|---|
| **SystemOOM** | There is an OOM (out of memory) situation on the cluster. |

Table 4.9. Pod Events

| Name | Description |
|---|---|
| **FailedKillPod** | Failed to stop a pod. |
| **FailedCreatePodContainer** | Failed to create a pod contianer. |
| **Failed** | Failed to make pod data directories. |
| **NetworkNotReady** | Network is not ready. |
| **FailedCreate** | Error creating: **<error-msg>**. |
| **SuccessfulCreate** | Created pod: **<pod-name>**. |
| **FailedDelete** | Error deleting: **<error-msg>**. |
| **SuccessfulDelete** | Deleted pod: **<pod-id>**. |

Table 4.10. Horizontal Pod AutoScaler Events

| Name | Description |
|---|---|
| SelectorRequired | Selector is required. |
| **InvalidSelector** | Could not convert selector into a corresponding internal selector object. |
| **FailedGetObjectMetric** | HPA was unable to compute the replica count. |
| **InvalidMetricSourceType** | Unknown metric source type. |
| **ValidMetricFound** | HPA was able to successfully calculate a replica count. |
| **FailedConvertHPA** | Failed to convert the given HPA. |
| **FailedGetScale** | HPA controller was unable to get the target's current scale. |
| **SucceededGetScale** | HPA controller was able to get the target's current scale. |

| Name | Description |
|---|---|
| **FailedComputeMetricsReplicas** | Failed to compute desired number of replicas based on listed metrics. |
| **FailedRescale** | New size: **<size>**; reason: **<msg>**; error: **<error-msg>**. |
| **SuccessfulRescale** | New size: **<size>**; reason: **<msg>**. |
| **FailedUpdateStatus** | Failed to update status. |

Table 4.11. Network Events (openshift-sdn)

| Name | Description |
|---|---|
| **Starting** | Starting OpenShift-SDN. |
| **NetworkFailed** | The pod's network interface has been lost and the pod will be stopped. |

Table 4.12. Network Events (kube-proxy)

| Name | Description |
|---|---|
| **NeedPods** | The service-port **<serviceName>:<port>** needs pods. |

Table 4.13. Volume Events

| Name | Description |
|---|---|
| **FailedBinding** | There are no persistent volumes available and no storage class is set. |
| **VolumeMismatch** | Volume size or class is different from what is requested in claim. |
| **VolumeFailedRecycle** | Error creating recycler pod. |
| **VolumeRecycled** | Occurs when volume is recycled. |
| **RecyclerPod** | Occurs when pod is recycled. |
| **VolumeDelete** | Occurs when volume is deleted. |

| Name | Description |
|---|---|
| **VolumeFailedDelete** | Error when deleting the volume. |
| **ExternalProvisioning** | Occurs when volume for the claim is provisioned either manually or via external software. |
| **ProvisioningFailed** | Failed to provision volume. |
| **ProvisioningCleanupFailed** | Error cleaning provisioned volume. |
| **ProvisioningSucceeded** | Occurs when the volume is provisioned successfully. |
| **WaitForFirstConsumer** | Delay binding until pod scheduling. |

Table 4.14. Lifecycle hooks

| Name | Description |
|---|---|
| **FailedPostStartHook** | Handler failed for pod start. |
| **FailedPreStopHook** | Handler failed for pre-stop. |
| **UnfinishedPreStopHook** | Pre-stop hook unfinished. |

Table 4.15. Deployments

| Name | Description |
|---|---|
| **DeploymentCancellationFailed** | Failed to cancel deployment. |
| **DeploymentCancelled** | Cancelled deployment. |
| **DeploymentCreated** | Created new replication controller. |
| **IngressIPRangeFull** | No available Ingress IP to allocate to service. |

Table 4.16. Scheduler Events

| Name | Description |
|---|---|
| **FailedScheduling** | Failed to schedule pod: **<pod-namespace>/<pod-name>**. This event is raised for multiple reasons, for example: **AssumePodVolumes** failed, Binding rejected etc. |
| **Preempted** | By **<preemptor-namespace>/<preemptor-name>** on node **<node-name>**. |
| **Scheduled** | Successfully assigned **<pod-name>** to **<node-name>**. |

Table 4.17. DaemonSet Events

| Name | Description |
|---|---|
| **SelectingAll** | This daemon set is selecting all pods. A non-empty selector is required. |
| **FailedPlacement** | Failed to place pod on **<node-name>**. |
| **FailedDaemonPod** | Found failed daemon pod **<pod-name>** on node **<node-name>**, will try to kill it. |

Table 4.18. LoadBalancer Service Events

| Name | Description |
|---|---|
| **CreatingLoadBalancerFailed** | Error creating load balancer. |
| **DeletingLoadBalancer** | Deleting load balancer. |
| **EnsuringLoadBalancer** | Ensuring load balancer. |
| **EnsuredLoadBalancer** | Ensured load balancer. |
| **UnAvailableLoadBalancer** | There are no available nodes for **LoadBalancer** service. |
| **LoadBalancerSourceRanges** | Lists the new **LoadBalancerSourceRanges**. For example, **<old-source-range> → <new-source-range>**. |
| **LoadbalancerIP** | Lists the new IP address. For example, **<old-ip> → <new-ip>**. |
| **ExternalIP** | Lists external IP address. For example, **Added: <external-ip>**. |

| Name | Description |
|---|---|
| **UID** | Lists the new UID. For example, **<old-service-uid> → <new-service-uid>**. |
| **ExternalTrafficPolicy** | Lists the new **ExternalTrafficPolicy**. For example,**<old-policy> → <new-ploicy>**. |
| **HealthCheckNodePort** | Lists the new **HealthCheckNodePort**. For example,**<old-node-port> → new-node-port>**. |
| **UpdatedLoadBalancer** | Updated load balancer with new hosts. |
| **LoadBalancerUpdateFailed** | Error updating load balancer with new hosts. |
| **DeletingLoadBalancer** | Deleting load balancer. |
| **DeletingLoadBalancerFailed** | Error deleting load balancer. |
| **DeletedLoadBalancer** | Deleted load balancer. |