



# OpenShift Container Platform 4.4

## Applications

Creating and managing applications on OpenShift Container Platform



# OpenShift Container Platform 4.4 Applications

---

Creating and managing applications on OpenShift Container Platform

## Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document provides instructions for the various ways to create and manage instances of user-provisioned applications running on OpenShift Container Platform. This includes working with projects and provisioning applications using the Open Service Broker API.

## Table of Contents

<b>CHAPTER 1. PROJECTS</b> .....	<b>6</b>
1.1. WORKING WITH PROJECTS	6
1.1.1. Creating a project using the web console	6
1.1.2. Creating a project using the Developer perspective in the web console	6
1.1.3. Creating a project using the CLI	8
1.1.4. Viewing a project using the web console	8
1.1.5. Viewing a project using the CLI	9
1.1.6. Providing access permissions to your project using the Developer perspective	9
1.1.7. Adding to a project	10
1.1.8. Checking project status using the web console	10
1.1.9. Checking project status using the CLI	10
1.1.10. Deleting a project using the web console	10
1.1.11. Deleting a project using the CLI	11
1.2. CREATING A PROJECT AS ANOTHER USER	11
1.2.1. API impersonation	11
1.2.2. Impersonating a user when you create a project	11
1.3. CONFIGURING PROJECT CREATION	11
1.3.1. About project creation	12
1.3.2. Modifying the template for new projects	12
1.3.3. Disabling project self-provisioning	13
1.3.4. Customizing the project request message	14
<b>CHAPTER 2. APPLICATION LIFE CYCLE MANAGEMENT</b> .....	<b>16</b>
2.1. CREATING APPLICATIONS USING THE DEVELOPER PERSPECTIVE	16
2.1.1. Prerequisites	16
2.1.2. Importing a codebase from Git to create an application	16
2.2. CREATING APPLICATIONS FROM INSTALLED OPERATORS	18
2.2.1. Creating an etcd cluster using an Operator	19
2.3. CREATING APPLICATIONS USING THE CLI	21
2.3.1. Creating an application from source code	21
2.3.1.1. Local	22
2.3.1.2. Remote	22
2.3.1.3. Build strategy detection	22
2.3.1.4. Language detection	23
2.3.2. Creating an application from an image	24
2.3.2.1. Docker Hub MySQL image	24
2.3.2.2. Image in a private registry	24
2.3.2.3. Existing image stream and optional image stream tag	24
2.3.3. Creating an application from a template	25
2.3.3.1. Template parameters	25
2.3.4. Modifying application creation	25
2.3.4.1. Specifying environment variables	26
2.3.4.2. Specifying build environment variables	26
2.3.4.3. Specifying labels	27
2.3.4.4. Viewing the output without creation	27
2.3.4.5. Creating objects with different names	27
2.3.4.6. Creating objects in a different project	27
2.3.4.7. Creating multiple objects	28
2.3.4.8. Grouping images and source in a single pod	28
2.3.4.9. Searching for images, templates, and other inputs	28
2.4. VIEWING APPLICATION COMPOSITION USING THE TOPOLOGY VIEW	28

2.4.1. Prerequisites	28
2.4.2. Viewing the topology of your application	29
2.4.3. Interacting with the application and the components	30
2.4.4. Scaling application pods and checking builds and routes	31
2.4.5. Grouping multiple components within an application	31
2.4.6. Connecting components within an application and across applications	32
2.4.6.1. Creating a visual connection between components	33
2.4.6.2. Creating a binding connection between components	35
2.4.7. Labels and annotations used for the Topology view	36
2.5. EDITING APPLICATIONS	37
2.5.1. Prerequisites	37
2.5.2. Editing the source code of an application using the Developer perspective	37
2.5.3. Editing the application configuration using the Developer perspective	37
2.6. WORKING WITH HELM CHARTS USING THE DEVELOPER PERSPECTIVE	40
2.6.1. Understanding Helm	40
2.6.1.1. Key features	40
2.6.2. Installing Helm charts	40
2.7. DELETING APPLICATIONS	42
2.7.1. Deleting applications using the Developer perspective	42
<b>CHAPTER 3. SERVICE BROKERS</b>	<b>43</b>
3.1. INSTALLING SERVICE CATALOG	43
3.1.1. About Service Catalog	43
3.1.2. Installing Service Catalog	43
3.2. UNINSTALLING SERVICE CATALOG	44
3.2.1. Uninstalling Service Catalog	44
3.3. INSTALLING THE TEMPLATE SERVICE BROKER	45
3.3.1. About the Template Service Broker	45
3.3.2. Installing the Template Service Broker Operator	46
3.3.3. Starting the Template Service Broker	46
3.4. UNINSTALLING THE TEMPLATE SERVICE BROKER	47
3.4.1. Uninstalling the Template Service Broker	48
<b>CHAPTER 4. DEPLOYMENTS</b>	<b>50</b>
4.1. UNDERSTANDING DEPLOYMENT AND DEPLOYMENTCONFIG OBJECTS	50
4.1.1. Building blocks of a deployment	50
4.1.1.1. Replication controllers	50
4.1.1.2. Replica sets	51
4.1.2. DeploymentConfig objects	52
4.1.3. Deployments	54
4.1.4. Comparing Deployment and DeploymentConfig objects	54
4.1.4.1. Design	55
4.1.4.2. DeploymentConfig object-specific features	55
Automatic rollbacks	55
Triggers	55
Lifecycle hooks	55
Custom strategies	55
4.1.4.3. Deployment-specific features	55
Rollover	55
Proportional scaling	55
Pausing mid-rollout	56
4.2. MANAGING DEPLOYMENT PROCESSES	56
4.2.1. Managing DeploymentConfig objects	56

4.2.1.1. Starting a deployment	56
4.2.1.2. Viewing a deployment	56
4.2.1.3. Retrying a deployment	57
4.2.1.4. Rolling back a deployment	57
4.2.1.5. Executing commands inside a container	57
4.2.1.6. Viewing deployment logs	58
4.2.1.7. Deployment triggers	58
Config change deployment triggers	59
Image change deployment triggers	59
4.2.1.7.1. Setting deployment triggers	60
4.2.1.8. Setting deployment resources	60
4.2.1.9. Scaling manually	61
4.2.1.10. Accessing private repositories from DeploymentConfig objects	61
4.2.1.11. Assigning pods to specific nodes	62
4.2.1.12. Running a pod with a different service account	62
4.3. USING DEPLOYMENT STRATEGIES	62
4.3.1. Rolling strategy	63
4.3.1.1. Canary deployments	65
4.3.1.2. Creating a rolling deployment	65
4.3.1.3. Starting a rolling deployment using the Developer perspective	66
4.3.2. Recreate strategy	67
4.3.3. Starting a recreate deployment using the Developer perspective	68
4.3.4. Custom strategy	69
4.3.5. Lifecycle hooks	71
Pod-based lifecycle hook	71
4.3.5.1. Setting lifecycle hooks	72
4.4. USING ROUTE-BASED DEPLOYMENT STRATEGIES	72
4.4.1. Proxy shards and traffic splitting	73
4.4.2. N-1 compatibility	73
4.4.3. Graceful termination	73
4.4.4. Blue-green deployments	74
4.4.4.1. Setting up a blue-green deployment	74
4.4.5. A/B deployments	75
4.4.5.1. Load balancing for A/B testing	75
4.4.5.1.1. Managing weights of an existing route using the web console	76
4.4.5.1.2. Managing weights of a new route using the web console	77
4.4.5.1.3. Managing weights using the CLI	77
4.4.5.1.4. One service, multiple DeploymentConfig objects	78
<b>CHAPTER 5. QUOTAS</b> .....	<b>80</b>
5.1. RESOURCE QUOTAS PER PROJECT	80
5.1.1. Resources managed by quotas	80
5.1.2. Quota scopes	82
5.1.3. Quota enforcement	83
5.1.4. Requests versus limits	83
5.1.5. Sample resource quota definitions	83
5.1.6. Creating a quota	87
5.1.6.1. Creating object count quotas	87
5.1.6.2. Setting resource quota for extended resources	88
5.1.7. Viewing a quota	90
5.1.8. Configuring explicit resource quotas	90
5.2. RESOURCE QUOTAS ACROSS MULTIPLE PROJECTS	93
5.2.1. Selecting multiple projects during quota creation	93

5.2.2. Viewing applicable cluster resource quotas	95
5.2.3. Selection granularity	95
<b>CHAPTER 6. MONITORING PROJECT AND APPLICATION METRICS USING THE DEVELOPER PERSPECTIVE</b>	<b>96</b>
6.1. PREREQUISITES	96
6.2. MONITORING YOUR PROJECT METRICS	96
6.3. MONITORING YOUR APPLICATION METRICS	98
<b>CHAPTER 7. MONITORING APPLICATION HEALTH BY USING HEALTH CHECKS</b>	<b>100</b>
7.1. UNDERSTANDING HEALTH CHECKS	100
Example probes	101
7.2. CONFIGURING HEALTH CHECKS	103
<b>CHAPTER 8. IDLING APPLICATIONS</b>	<b>106</b>
8.1. IDLING APPLICATIONS	106
8.1.1. Idling a single service	106
8.1.2. Idling multiple services	106
8.2. UNIDLING APPLICATIONS	106
<b>CHAPTER 9. PRUNING OBJECTS TO RECLAIM RESOURCES</b>	<b>108</b>
9.1. BASIC PRUNING OPERATIONS	108
9.2. PRUNING GROUPS	108
9.3. PRUNING DEPLOYMENTCONFIG OBJECTS	108
9.4. PRUNING BUILDS	109
9.5. AUTOMATICALLY PRUNING IMAGES	110
9.6. MANUALLY PRUNING IMAGES	112
9.6.1. Image prune conditions	115
9.6.2. Running the image prune operation	117
9.6.3. Using secure or insecure connections	117
9.6.4. Image pruning problems	117
Images not being pruned	117
Using a secure connection against insecure registry	118
Using an insecure connection against a secured registry	118
Using the wrong certificate authority	119
9.7. HARD PRUNING THE REGISTRY	119
9.8. PRUNING CRON JOBS	122
<b>CHAPTER 10. USING THE RED HAT MARKETPLACE</b>	<b>123</b>
10.1. RED HAT MARKETPLACE FEATURES	123
10.1.1. Connect OpenShift Container Platform clusters to the Marketplace	123
10.1.2. Install applications	123
10.1.3. Deploy applications from different perspectives	123
The Developer perspective	123
The Administrator perspective	123





# CHAPTER 1. PROJECTS

## 1.1. WORKING WITH PROJECTS

A *project* allows a community of users to organize and manage their content in isolation from other communities.



### NOTE

Projects starting with **openshift-** and **kube-** are [default projects](#). These projects host cluster components that run as pods and other infrastructure components. As such, OpenShift Container Platform does not allow you to create projects starting with **openshift-** or **kube-** using the **oc new-project** command. Cluster administrators can create these projects using the **oc adm new-project** command.



### NOTE

You cannot assign an SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, and **openshift**. You cannot use these namespaces for running pods or services.

### 1.1.1. Creating a project using the web console

If allowed by your cluster administrator, you can create a new project.



### NOTE

Projects starting with **openshift-** and **kube-** are considered critical by OpenShift Container Platform. As such, OpenShift Container Platform does not allow you to create Projects starting with **openshift-** using the web console.



### NOTE

You cannot assign an SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, and **openshift**. You cannot use these namespaces for running pods or services.

### Procedure

1. Navigate to **Home** → **Projects**.
2. Click **Create Project**.
3. Enter your project details.
4. Click **Create**.

### 1.1.2. Creating a project using the Developer perspective in the web console

You can use the **Developer** perspective in the OpenShift Container Platform web console to create a project in your cluster.



## NOTE

Projects starting with **openshift-** and **kube-** are considered critical by OpenShift Container Platform. As such, OpenShift Container Platform does not allow you to create projects starting with **openshift-** or **kube-** using the **Developer** perspective. Cluster administrators can create these projects using the **oc adm new-project** command.



## NOTE

You cannot assign an SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, and **openshift**. You cannot use these namespaces for running pods or services.

## Prerequisites

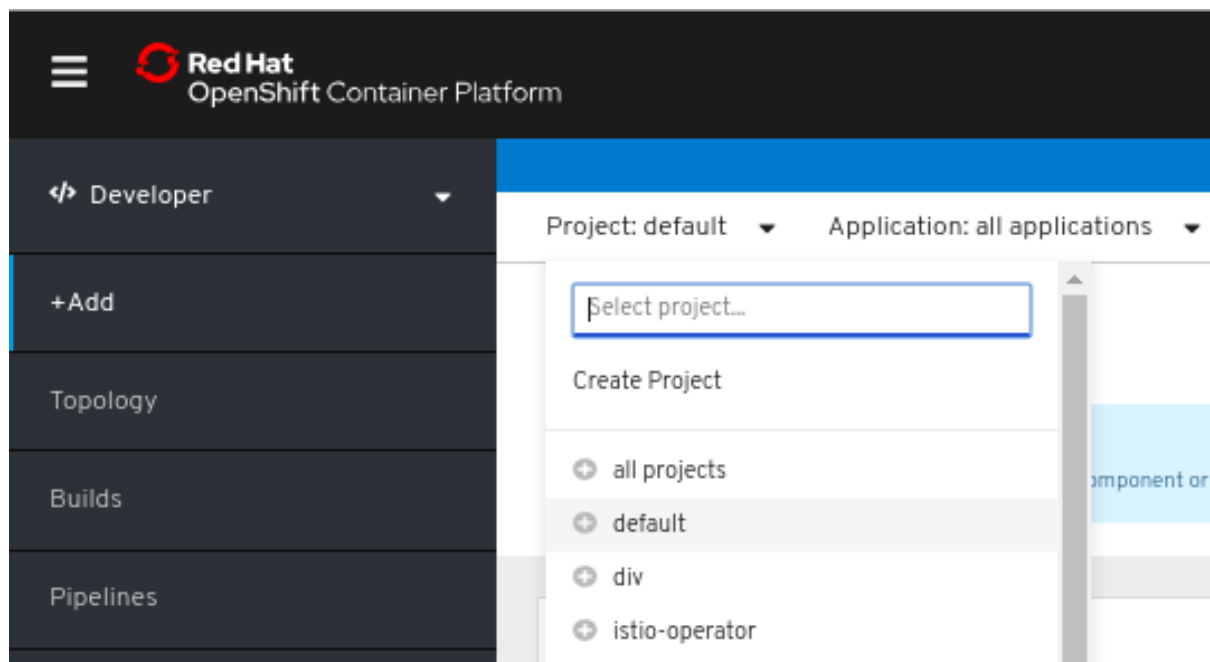
- Ensure that you have the appropriate [roles and permissions](#) to create projects, applications, and other workloads in OpenShift Container Platform.

## Procedure

You can create a project using the **Developer** perspective, as follows:

1. Click the **Project** drop-down menu to see a list of all available projects. Select **Create Project**.

Figure 1.1. Create project



2. In the **Create Project** dialog box, enter a unique name, such as **myproject**, in the **Name** field.
3. Optional: Add the **Display Name** and **Description** details for the project.
4. Click **Create**.
5. Use the left navigation panel to navigate to the **Project** view and see the dashboard for your project.
6. Optional:
  - Use the **Project** drop-down menu at the top of the screen and select **all projects** to list all

of the projects in your cluster.

- Use the **Details** tab to see the project details.
- If you have adequate permissions for a project, you can use the **Project Access** tab to provide or revoke *administrator*, *edit*, and *view* privileges for the project.

### 1.1.3. Creating a project using the CLI

If allowed by your cluster administrator, you can create a new project.



#### NOTE

Projects starting with **openshift-** and **kube-** are considered critical by OpenShift Container Platform. As such, OpenShift Container Platform does not allow you to create Projects starting with **openshift-** or **kube-** using the **oc new-project** command. Cluster administrators can create these Projects using the **oc adm new-project** command.



#### NOTE

You cannot assign an SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, and **openshift**. You cannot use these namespaces for running pods or services.

#### Procedure

1. Run:

```
$ oc new-project <project_name> \
  --description="<description>" --display-name="<display_name>"
```

For example:

```
$ oc new-project hello-openshift \
  --description="This is an example project" \
  --display-name="Hello OpenShift"
```



#### NOTE

The number of projects you are allowed to create may be limited by the system administrator. After your limit is reached, you might have to delete an existing project in order to create a new one.

### 1.1.4. Viewing a project using the web console

#### Procedure

1. Navigate to **Home** → **Projects**.
2. Select a project to view.  
On this page, click the **Workloads** button to see workloads in the project.

## 1.1.5. Viewing a project using the CLI

When viewing projects, you are restricted to seeing only the projects you have access to view based on the authorization policy.

### Procedure

1. To view a list of projects, run:

```
$ oc get projects
```

2. You can change from the current project to a different project for CLI operations. The specified project is then used in all subsequent operations that manipulate project-scoped content:

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

## 1.1.6. Providing access permissions to your project using the Developer perspective

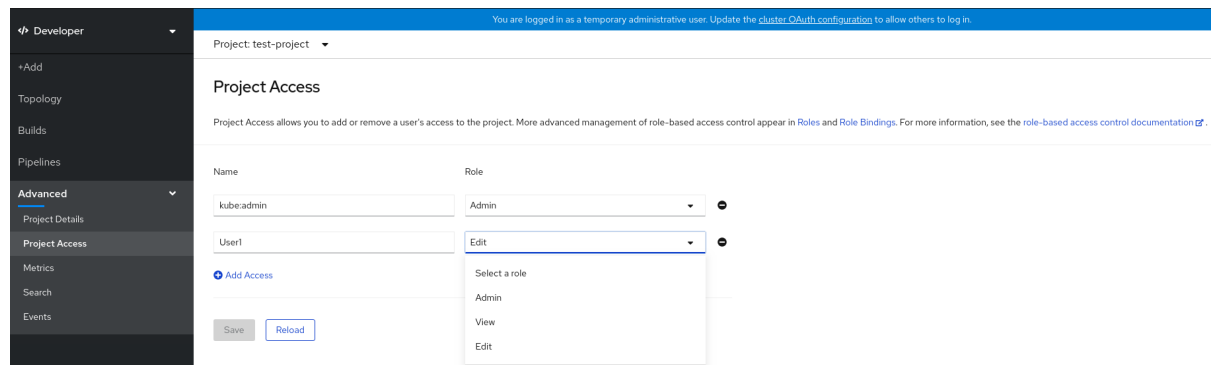
You can use the **Project Access** view in the **Developer** perspective to grant or revoke access permissions to your project.

### Procedure

To add users to your project and provide **Admin**, **View**, or **Edit** access to them:

1. In the **Developer** perspective, navigate to the **Advanced → Project Access** page.
2. In the **Project Access** page, click **Add Access** to add a new row.

Figure 1.2. Project permissions



3. Enter the user name, click the **Select a role** drop-down list, and select an appropriate role.
4. Click **Save**.

You can also use:

- The **Select a role** drop-down list, to modify the access permissions of an existing user.
- The **Remove Access** icon, to completely remove the access permissions of an existing user to the project.

**NOTE**

Advanced role-based access control is managed in the **Roles** and **Roles Binding** views in the **Administrator** perspective.

### 1.1.7. Adding to a project

**Procedure**

1. Select **Developer** from the context selector at the top of the web console navigation menu.
2. Click **+Add**
3. At the top of the page, select the name of the project that you want to add to.
4. Click on a method for adding to your project, and then follow the workflow.

### 1.1.8. Checking project status using the web console

**Procedure**

1. Navigate to **Home → Projects**.
2. Select a project to see its status.

### 1.1.9. Checking project status using the CLI

**Procedure**

1. Run:

```
❯ $ oc status
```

This command provides a high-level overview of the current project, with its components and their relationships.

### 1.1.10. Deleting a project using the web console


You can delete a project by using the OpenShift Container Platform web console.

**NOTE**

If you do not have permissions to delete the project, the **Delete Project** option is not available.

**Procedure**

1. Navigate to **Home → Projects**.
2. Locate the project that you want to delete from the list of projects.

3. On the far right side of the project listing, select **Delete Project** from the Options menu  .
4. When the **Delete Project** pane opens, enter the name of the project that you want to delete in the field.
5. Click **Delete**.

### 1.1.11. Deleting a project using the CLI

When you delete a project, the server updates the project status to **Terminating** from **Active**. Then, the server clears all content from a project that is in the **Terminating** state before finally removing the project. While a project is in **Terminating** status, you cannot add new content to the project. Projects can be deleted from the CLI or the web console.

#### Procedure

1. Run:

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

## 1.2. CREATING A PROJECT AS ANOTHER USER

Impersonation allows you to create a project as a different user.

### 1.2.1. API impersonation

You can configure a request to the OpenShift Container Platform API to act as though it originated from another user. For more information, see [User impersonation](#) in the Kubernetes documentation.

### 1.2.2. Impersonating a user when you create a project

You can impersonate a different user when you create a project request. Because **system:authenticated:oauth** is the only bootstrap group that can create project requests, you must impersonate that group.

#### Procedure

- To create a project request on behalf of a different user:

```
$ oc new-project <project> --as=<user> \
  --as-group=system:authenticated --as-group=system:authenticated:oauth
```

## 1.3. CONFIGURING PROJECT CREATION

In OpenShift Container Platform, *projects* are used to group and isolate related objects. When a request is made to create a new project using the web console or **oc new-project** command, an endpoint in OpenShift Container Platform is used to provision the project according to a template, which can be customized.

As a cluster administrator, you can allow and configure how developers and service accounts can create, or *self-provision*, their own projects.

### 1.3.1. About project creation

The OpenShift Container Platform API server automatically provisions new projects based on the project template that is identified by the **projectRequestTemplate** parameter in the cluster's project configuration resource. If the parameter is not defined, the API server creates a default template that creates a project with the requested name, and assigns the requesting user to the **admin** role for that project.

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

**Table 1.1. Default project template parameters**

Parameter	Description
<b>PROJECT_NAME</b>	The name of the project. Required.
<b>PROJECT_DISPLAYNAME</b>	The display name of the project. May be empty.
<b>PROJECT_DESCRIPTION</b>	The description of the project. May be empty.
<b>PROJECT_ADMIN_USER</b>	The user name of the administrating user.
<b>PROJECT_REQUESTING_USER</b>	The user name of the requesting user.

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

### 1.3.2. Modifying the template for new projects

As a cluster administrator, you can modify the default project template so that new projects are created using your custom requirements.

To create your own custom project template:

#### Procedure

1. Log in as a user with **cluster-admin** privileges.
2. Generate the default project template:
 

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```
3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.
4. The project template must be created in the **openshift-config** namespace. Load your modified template:

```
$ oc create -f template.yaml -n openshift-config
```

5. Edit the project configuration resource using the web console or CLI.



- Using the web console:
  - i. Navigate to the **Administration** → **Cluster Settings** page.
  - ii. Click **Global Configuration** to view all configuration resources.
  - iii. Find the entry for **Project** and click **Edit YAML**.
- Using the CLI:
  - i. Edit the **project.config.openshift.io/cluster** resource:

```
$ oc edit project.config.openshift.io/cluster
```

6. Update the **spec** section to include the **projectRequestTemplate** and **name** parameters, and set the name of your uploaded project template. The default name is **project-request**.

### Project configuration resource with custom project template

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. After you save your changes, create a new project to verify that your changes were successfully applied.

### 1.3.3. Disabling project self-provisioning

You can prevent an authenticated user group from self-provisioning new projects.

#### Procedure

1. Log in as a user with **cluster-admin** privileges.
2. View the **self-provisioners** cluster role binding usage by running the following command:

```
$ oc describe clusterrolebinding.rbac self-provisioners

Name: self-provisioners
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name  Namespace
  ----
  Group system:authenticated:oauth
```

Review the subjects in the **self-provisioners** section.

- Remove the **self-provisioner** cluster role from the group **system:authenticated:oauth**.

- If the **self-provisioners** cluster role binding binds only the **self-provisioner** role to the **system:authenticated:oauth** group, run the following command:

```
$ oc patch clusterrolebinding.rbac self-provisioners -p '{"subjects": null}'
```

- If the **self-provisioners** cluster role binding binds the **self-provisioner** role to more users, groups, or service accounts than the **system:authenticated:oauth** group, run the following command:

```
$ oc adm policy \
  remove-cluster-role-from-group self-provisioner \
  system:authenticated:oauth
```

- Edit the **self-provisioners** cluster role binding to prevent automatic updates to the role. Automatic updates reset the cluster roles to the default state.

- To update the role binding using the CLI:

- Run the following command:

```
$ oc edit clusterrolebinding.rbac self-provisioners
```

- In the displayed role binding, set the **rbac.authorization.kubernetes.io/autoupdate** parameter value to **false**, as shown in the following example:

```
apiVersion: authorization.openshift.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false"
  ...
```

- To update the role binding by using a single command:

```
$ oc patch clusterrolebinding.rbac self-provisioners -p '{"metadata": {"annotations": {
  "rbac.authorization.kubernetes.io/autoupdate": "false" } } }'
```

- Log in as an authenticated user and verify that it can no longer self-provision a project:

```
$ oc new-project test
```

```
Error from server (Forbidden): You may not request a new project via this API.
```

Consider customizing this project request message to provide more helpful instructions specific to your organization.

### 1.3.4. Customizing the project request message

When a developer or a service account that is unable to self-provision projects makes a project creation request using the web console or CLI, the following error message is returned by default:

```
You may not request a new project via this API.
```

Cluster administrators can customize this message. Consider updating it to provide further instructions on how to request a new project specific to your organization. For example:

- To request a project, contact your system administrator at **projectname@example.com**.
- To request a new project, fill out the project request form located at **https://internal.example.com/openshift-project-request**.

To customize the project request message:

## Procedure

1. Edit the project configuration resource using the web console or CLI.
  - Using the web console:
    - i. Navigate to the **Administration → Cluster Settings** page.
    - ii. Click **Global Configuration** to view all configuration resources.
    - iii. Find the entry for **Project** and click **Edit YAML**.
  - Using the CLI:
    - i. Log in as a user with **cluster-admin** privileges.
    - ii. Edit the **project.config.openshift.io/cluster** resource:

```
$ oc edit project.config.openshift.io/cluster
```

2. Update the **spec** section to include the **projectRequestMessage** parameter and set the value to your custom message:

### Project configuration resource with custom project request message

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestMessage: <message_string>
```

For example:

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestMessage: To request a project, contact your system administrator at
  projectname@example.com.
```

3. After you save your changes, attempt to create a new project as a developer or service account that is unable to self-provision projects to verify that your changes were successfully applied.

## CHAPTER 2. APPLICATION LIFE CYCLE MANAGEMENT

### 2.1. CREATING APPLICATIONS USING THE DEVELOPER PERSPECTIVE

The **Developer** perspective in the web console provides you the following options from the **+Add** view to create applications and associated services and deploy them on OpenShift Container Platform:

- **From Git** Use this option to import an existing codebase in a Git repository to create, build, and deploy an application on OpenShift Container Platform.
- **Container Image**: Use existing images from an image stream or registry to deploy it on to OpenShift Container Platform.
- **From Catalog**: Explore the **Developer Catalog** to select the required applications, services, or source to image builders and add it to your project.
- **From Dockerfile**: Import a dockerfile from your Git repository to build and deploy an application.
- **YAML**: Use the editor to add YAML or JSON definitions to create and modify resources.
- **Database**: See the **Developer Catalog** to select the required database service and add it to your application.



#### NOTE

Serverless options in the **Developer** perspective are displayed only if the [OpenShift Serverless Operator](#) is installed in your cluster.

#### 2.1.1. Prerequisites

To create applications using the **Developer** perspective ensure that:

- You have [logged in to the web console](#) .
- You are in the [Developer perspective](#) .
- You have the appropriate [roles and permissions](#) in a project to create applications and other workloads in OpenShift Container Platform.

To create serverless applications, in addition to the preceding prerequisites, ensure that:

- You have [installed the OpenShift Serverless Operator](#) .
- You have [created a knative-serving namespace and a KnativeServing resource in the knative-serving namespace](#) .

#### 2.1.2. Importing a codebase from Git to create an application

The following procedure walks you through the **Import from Git** option in the **Developer** perspective to create an application.

Create, build, and deploy an application on OpenShift Container Platform using an existing codebase in GitHub as follows:

## Procedure

1. In the **+Add** view, click **From Git** to see the **Import from git** form.
2. In the **Git** section, enter the Git repository URL for the codebase you want to use to create an application. For example, enter the URL of this sample Node.js application <https://github.com/sclorg/nodejs-ex>. The URL is then validated.
3. Optional: You can click **Show Advanced Git Options** to add details such as:
  - **Git Reference** to point to code in a specific branch, tag, or commit to be used to build the application.
  - **Context Dir** to specify the subdirectory for the application source code you want to use to build the application.
  - **Source Secret** to create a **Secret Name** with credentials for pulling your source code from a private repository.
4. In the **Builder** section, after the URL is validated, an appropriate builder image is detected, indicated by a star, and automatically selected. For the <https://github.com/sclorg/nodejs-ex> Git URL, the Node.js builder image is selected by default. If required, you can change the version using the **Builder Image Version** drop-down list.
5. In the **General** section:
  - a. In the **Application** field, enter a unique name for the application grouping, for example, **myapp**. Ensure that the application name is unique in a namespace.
  - b. The **Name** field to identify the resources created for this application is automatically populated based on the Git repository URL.



### NOTE

The resource name must be unique in a namespace. Modify the resource name if you get an error.

6. In the **Resources** section, select:
  - **Deployment**, to create an application in plain Kubernetes style.
  - **Deployment Config**, to create an OpenShift style application.
  - **Knative Service**, to create a microservice.



### NOTE

The **Knative Service** option is displayed in the **Import from git** form only if the **Serverless Operator** is installed in your cluster. For further details refer to documentation on installing OpenShift Serverless.

7. In the **Advanced Options** section, the **Create a route to the application** is selected by default so that you can access your application using a publicly available URL. You can clear the check box if you do not want to expose your application on a public route.
8. Optional: You can use the following advanced options to further customize your application:

## Routing

Click the **Routing** link to:

- Customize the hostname for the route.
- Specify the path the router watches.
- Select the target port for the traffic from the drop-down list.
- Secure your route by selecting the **Secure Route** check box. Select the required TLS termination type and set a policy for insecure traffic from the respective drop-down lists.

For serverless applications, the Knative Service manages all the routing options above. However, you can customize the target port for traffic, if required. If the target port is not specified, the default port of **8080** is used.

## Build and Deployment Configuration

Click the **Build Configuration** and **Deployment Configuration** links to see the respective configuration options. Some of the options are selected by default; you can customize them further by adding the necessary triggers and environment variables. For serverless applications, the **Deployment Configuration** option is not displayed as the Knative configuration resource maintains the desired state for your deployment instead of a DeploymentConfig.

## Scaling

Click the **Scaling** link to define the number of pods or instances of the application you want to deploy initially.

For serverless applications, you can:

- Set the upper and lower limit for the number of pods that can be set by the autoscaler. If the lower limit is not specified, it defaults to zero.
- Define the soft limit for the required number of concurrent requests per instance of the application at a given time. It is the recommended configuration for autoscaling. If not specified, it takes the value specified in the cluster configuration.
- Define the hard limit for the number of concurrent requests allowed per instance of the application at a given time. This is configured in the revision template. If not specified, it defaults to the value specified in the cluster configuration.

## Resource Limit

Click the **Resource Limit** link to set the amount of **CPU** and **Memory** resources a container is guaranteed or allowed to use when running.

## Labels

Click the **Labels** link to add custom labels to your application.

9. Click **Create** to create the application and see its build status in the **Topology** view.

## 2.2. CREATING APPLICATIONS FROM INSTALLED OPERATORS

*Operators* are a method of packaging, deploying, and managing a Kubernetes application. You can create applications on OpenShift Container Platform using Operators that have been installed by a cluster administrator.

This guide walks developers through an example of creating applications from an installed Operator using the OpenShift Container Platform web console.

### Additional resources

- See the [Operators](#) guide for more on how Operators work and how the Operator Lifecycle Manager is integrated in OpenShift Container Platform.

## 2.2.1. Creating an etcd cluster using an Operator

This procedure walks through creating a new etcd cluster using the etcd Operator, managed by the Operator Lifecycle Manager (OLM).

### Prerequisites

- Access to an OpenShift Container Platform 4.4 cluster.
- The etcd Operator already installed cluster-wide by an administrator.

### Procedure

1. Create a new project in the OpenShift Container Platform web console for this procedure. This example uses a project called **my-etcd**.
2. Navigate to the **Operators → Installed Operators** page. The Operators that have been installed to the cluster by the cluster administrator and are available for use are shown here as a list of ClusterServiceVersions (CSVs). CSVs are used to launch and manage the software provided by the Operator.

### TIP

You can get this list from the CLI using:

```
$ oc get csv
```

3. On the **Installed Operators** page, click **Copied**, and then click the etcd Operator to view more details and available actions:

Figure 2.1. etcd Operator overview

etcd  
0.9.2 provided by CoreOS, Inc

Actions ▾

Overview | YAML | Events | All Instances | etcd Cluster | etcd Backup | etcd Restore

PROVIDER  
CoreOS, Inc

CREATED AT  
Feb 4, 3:10 pm

LINKS  
Blog  
<https://coreos.com/etcd>

Documentation  
<https://coreos.com/operators/etcd/docs/latest/>

etcd Operator Source Code  
<https://github.com/coreos/etcd-operator>

MAINTAINERS  
CoreOS, Inc  
[support@coreos.com](mailto:support@coreos.com)

### Provided APIs

**EC etcd Cluster**  
Represents a cluster of etcd nodes.  
[Create New](#)

**EB etcd Backup**  
Represents the intent to backup an etcd cluster.  
[Create New](#)

**ER etcd Restore**  
Represents the intent to restore an etcd cluster from a backup.  
[Create New](#)

As shown under **Provided APIs**, this Operator makes available three new resource types, including one for an **etcd Cluster** (the **EtcCluster** resource). These objects work similar to the built-in native Kubernetes ones, such as **Deployments** or **ReplicaSets**, but contain logic specific to managing etcd.

4. Create a new etcd cluster:
  - a. In the **etcd Cluster** API box, click **Create New**.
  - b. The next screen allows you to make any modifications to the minimal starting template of an **EtcCluster** object, such as the size of the cluster. For now, click **Create** to finalize. This triggers the Operator to start up the pods, services, and other components of the new etcd cluster.
5. Click the **Resources** tab to see that your project now contains a number of resources created and configured automatically by the Operator.



Figure 2.2. etcd Operator resources

etcdoperator.v0.9.2 > EtcdCluster Details

EC example Actions ▾

---

Overview    YAML    **Resources**

---

Filter Resources by name...

---

2 Service    3 Pod    Select All Filters    5 Items

NAME ↑	TYPE	STATUS	CREATED
<span>S</span> example	Service	Created	🕒 3 minutes ago
<span>S</span> example-client	Service	Created	🕒 3 minutes ago
<span>P</span> example-dccdn267hl	Pod	Running	🕒 2 minutes ago
<span>P</span> example-g2shm4cz4l	Pod	Running	🕒 2 minutes ago
<span>P</span> example-sgm2hcktcn	Pod	Running	🕒 3 minutes ago

Verify that a Kubernetes service has been created that allows you to access the database from other pods in your project.

- All users with the **edit** role in a given project can create, manage, and delete application instances (an etcd cluster, in this example) managed by Operators that have already been created in the project, in a self-service manner, just like a cloud service. If you want to enable additional users with this ability, project administrators can add the role using the following command:

```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

You now have an etcd cluster that will react to failures and rebalance data as Pods become unhealthy or are migrated between nodes in the cluster. Most importantly, cluster administrators or developers with proper access can now easily use the database with their applications.

## 2.3. CREATING APPLICATIONS USING THE CLI

You can create an OpenShift Container Platform application from components that include source or binary code, images, and templates by using the OpenShift Container Platform CLI.

The set of objects created by **new-app** depends on the artifacts passed as input: source repositories, images, or templates.

### 2.3.1. Creating an application from source code

With the **new-app** command you can create applications from source code in a local or remote Git repository.

The **new-app** command creates a build configuration, which itself creates a new application image from your source code. The **new-app** command typically also creates a **DeploymentConfig** object to deploy the new image, and a service to provide load-balanced access to the deployment running your image.

OpenShift Container Platform automatically detects whether the pipeline or source build strategy should be used, and in the case of source builds, detects an appropriate language builder image.

### 2.3.1.1. Local

To create an application from a Git repository in a local directory:

```
$ oc new-app /<path to source code>
```



#### NOTE

If you use a local Git repository, the repository must have a remote named **origin** that points to a URL that is accessible by the OpenShift Container Platform cluster. If there is no recognized remote, running the **new-app** command will create a binary build.

### 2.3.1.2. Remote

To create an application from a remote Git repository:

```
$ oc new-app https://github.com/sclorg/cakephp-ex
```

To create an application from a private remote Git repository:

```
$ oc new-app https://github.com/youruser/yourprivaterepo --source-secret=yoursecret
```



#### NOTE

If you use a private remote Git repository, you can use the **--source-secret** flag to specify an existing source clone secret that will get injected into your build config to access the repository.

You can use a subdirectory of your source code repository by specifying a **--context-dir** flag. To create an application from a remote Git repository and a context subdirectory:

```
$ oc new-app https://github.com/sclorg/s2i-ruby-container.git \  
  --context-dir=2.0/test/puma-test-app
```

Also, when specifying a remote URL, you can specify a Git branch to use by appending **#<branch\_name>** to the end of the URL:

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

### 2.3.1.3. Build strategy detection

If a Jenkins file exists in the root or specified context directory of the source repository when creating a new application, OpenShift Container Platform generates a pipeline build strategy. Otherwise, it generates a source build strategy.

Override the build strategy by setting the `--strategy` flag to either `pipeline` or `source`.

```
$ oc new-app /home/user/code/myapp --strategy=docker
```



#### NOTE

The `oc` command requires that files containing build sources are available in a remote Git repository. For all source builds, you must use `git remote -v`.

#### 2.3.1.4. Language detection

If you use the source build strategy, `new-app` attempts to determine the language builder to use by the presence of certain files in the root or specified context directory of the repository:

Table 2.1. Languages detected by `new-app`

Language	Files
<code>dotnet</code>	<code>project.json</code> , <code>*.csproj</code>
<code>jee</code>	<code>pom.xml</code>
<code>nodejs</code>	<code>app.json</code> , <code>package.json</code>
<code>perl</code>	<code>cpanfile</code> , <code>index.pl</code>
<code>php</code>	<code>composer.json</code> , <code>index.php</code>
<code>python</code>	<code>requirements.txt</code> , <code>setup.py</code>
<code>ruby</code>	<code>Gemfile</code> , <code>Rakefile</code> , <code>config.ru</code>
<code>scala</code>	<code>build.sbt</code>
<code>golang</code>	<code>Godeps</code> , <code>main.go</code>

After a language is detected, `new-app` searches the OpenShift Container Platform server for image stream tags that have a `supports` annotation matching the detected language, or an image stream that matches the name of the detected language. If a match is not found, `new-app` searches the [Docker Hub registry](#) for an image that matches the detected language based on name.

You can override the image the builder uses for a particular source repository by specifying the image, either an image stream or container specification, and the repository with a `~` as a separator. Note that if this is done, build strategy detection and language detection are not carried out.

For example, to use the `myproject/my-ruby` imagestream with the source in a remote repository:

```
$ oc new-app myproject/my-ruby~https://github.com/openshift/ruby-hello-world.git
```

To use the **openshift/ruby-20-centos7:latest** container image stream with the source in a local repository:

```
$ oc new-app openshift/ruby-20-centos7:latest~/home/user/code/my-ruby-app
```



#### NOTE

Language detection requires the Git client to be locally installed so that your repository can be cloned and inspected. If Git is not available, you can avoid the language detection step by specifying the builder image to use with your repository with the **<image>~<repository>** syntax.

The **-i <image> <repository>** invocation requires that **new-app** attempt to clone **repository** in order to determine what type of artifact it is, so this will fail if Git is not available.

The **-i <image> --code <repository>** invocation requires **new-app** clone **repository** in order to determine whether **image** should be used as a builder for the source code, or deployed separately, as in the case of a database image.

### 2.3.2. Creating an application from an image

You can deploy an application from an existing image. Images can come from image streams in the OpenShift Container Platform server, images in a specific registry, or images in the local Docker server.

The **new-app** command attempts to determine the type of image specified in the arguments passed to it. However, you can explicitly tell **new-app** whether the image is a container image using the **--docker-image** argument or an image stream using the **-i|--image-stream** argument.



#### NOTE

If you specify an image from your local Docker repository, you must ensure that the same image is available to the OpenShift Container Platform cluster nodes.

#### 2.3.2.1. Docker Hub MySQL image

Create an application from the Docker Hub MySQL image, for example:

```
$ oc new-app mysql
```

#### 2.3.2.2. Image in a private registry

Create an application using an image in a private registry, specify the full container image specification:

```
$ oc new-app myregistry:5000/example/myimage
```

#### 2.3.2.3. Existing image stream and optional image stream tag

Create an application from an existing image stream and optional image stream tag:

```
$ oc new-app my-stream:v1
```

### 2.3.3. Creating an application from a template

You can create an application from a previously stored template or from a template file, by specifying the name of the template as an argument. For example, you can store a sample application template and use it to create an application.

Create an application from a stored template, for example:

```
$ oc create -f examples/sample-app/application-template-stibuild.json
$ oc new-app ruby-helloworld-sample
```

To directly use a template in your local file system, without first storing it in OpenShift Container Platform, use the **-f|--file** argument. For example:

```
$ oc new-app -f examples/sample-app/application-template-stibuild.json
```

#### 2.3.3.1. Template parameters

When creating an application based on a template, use the **-p|--param** argument to set parameter values that are defined by the template:

```
$ oc new-app ruby-helloworld-sample \
  -p ADMIN_USERNAME=admin -p ADMIN_PASSWORD=mypassword
```

You can store your parameters in a file, then use that file with **--param-file** when instantiating a template. If you want to read the parameters from standard input, use **--param-file=-**:

```
$ cat helloworld.params
ADMIN_USERNAME=admin
ADMIN_PASSWORD=mypassword
$ oc new-app ruby-helloworld-sample --param-file=helloworld.params
$ cat helloworld.params | oc new-app ruby-helloworld-sample --param-file=-
```

### 2.3.4. Modifying application creation

The **new-app** command generates OpenShift Container Platform objects that build, deploy, and run the application that is created. Normally, these objects are created in the current project and assigned names that are derived from the input source repositories or the input images. However, with **new-app** you can modify this behavior.

Table 2.2. **new-app** output objects

Object	Description
<b>BuildConfig</b>	A <b>BuildConfig</b> object is created for each source repository that is specified in the command line. The <b>BuildConfig</b> object specifies the strategy to use, the source location, and the build output location.

Object	Description
<b>ImageStreams</b>	For the <b>BuildConfig</b> object, two image streams are usually created. One represents the input image. With source builds, this is the builder image. With <b>Docker</b> builds, this is the <b>FROM</b> image. The second one represents the output image. If a container image was specified as input to <b>new-app</b> , then an image stream is created for that image as well.
<b>DeploymentConfig</b>	A <b>DeploymentConfig</b> object is created either to deploy the output of a build, or a specified image. The <b>new-app</b> command creates <b>emptyDir</b> volumes for all Docker volumes that are specified in containers included in the resulting <b>DeploymentConfig</b> object.
<b>Service</b>	The <b>new-app</b> command attempts to detect exposed ports in input images. It uses the lowest numeric exposed port to generate a service that exposes that port. In order to expose a different port, after <b>new-app</b> has completed, simply use the <b>oc expose</b> command to generate additional services.
Other	Other objects can be generated when instantiating templates, according to the template.

### 2.3.4.1. Specifying environment variables

When generating applications from a template, source, or an image, you can use the **-e|--env** argument to pass environment variables to the application container at run time:

```
$ oc new-app openshift/postgresql-92-centos7 \
  -e POSTGRESQL_USER=user \
  -e POSTGRESQL_DATABASE=db \
  -e POSTGRESQL_PASSWORD=password
```

The variables can also be read from file using the **--env-file** argument:

```
$ cat postgresql.env
POSTGRESQL_USER=user
POSTGRESQL_DATABASE=db
POSTGRESQL_PASSWORD=password
$ oc new-app openshift/postgresql-92-centos7 --env-file=postgresql.env
```

Additionally, environment variables can be given on standard input by using **--env-file=-**:

```
$ cat postgresql.env | oc new-app openshift/postgresql-92-centos7 --env-file=-
```



#### NOTE

Any **BuildConfig** objects created as part of **new-app** processing are not updated with environment variables passed with the **-e|--env** or **--env-file** argument.

### 2.3.4.2. Specifying build environment variables

When generating applications from a template, source, or an image, you can use the **--build-env** argument to pass environment variables to the build container at run time:

```
$ oc new-app openshift/ruby-23-centos7 \
  --build-env HTTP_PROXY=http://myproxy.net:1337/ \
  --build-env GEM_HOME=~/.gem
```

The variables can also be read from a file using the **--build-env-file** argument:

```
$ cat ruby.env
HTTP_PROXY=http://myproxy.net:1337/
GEM_HOME=~/.gem
$ oc new-app openshift/ruby-23-centos7 --build-env-file=ruby.env
```

Additionally, environment variables can be given on standard input by using **--build-env-file=-**:

```
$ cat ruby.env | oc new-app openshift/ruby-23-centos7 --build-env-file=-
```

### 2.3.4.3. Specifying labels

When generating applications from source, images, or templates, you can use the **-l|--label** argument to add labels to the created objects. Labels make it easy to collectively select, configure, and delete objects associated with the application.

```
$ oc new-app https://github.com/openshift/ruby-hello-world -l name=hello-world
```

### 2.3.4.4. Viewing the output without creation

To see a dry-run of running the **new-app** command, you can use the **-o|--output** argument with a **yaml** or **json** value. You can then use the output to preview the objects that are created or redirect it to a file that you can edit. After you are satisfied, you can use **oc create** to create the OpenShift Container Platform objects.

To output **new-app** artifacts to a file, edit them, then create them:

```
$ oc new-app https://github.com/openshift/ruby-hello-world \
  -o yaml > myapp.yaml
$ vi myapp.yaml
$ oc create -f myapp.yaml
```

### 2.3.4.5. Creating objects with different names

Objects created by **new-app** are normally named after the source repository, or the image used to generate them. You can set the name of the objects produced by adding a **--name** flag to the command:

```
$ oc new-app https://github.com/openshift/ruby-hello-world --name=myapp
```

### 2.3.4.6. Creating objects in a different project

Normally, **new-app** creates objects in the current project. However, you can create objects in a different project by using the **-n|--namespace** argument:

```
$ oc new-app https://github.com/openshift/ruby-hello-world -n myproject
```

### 2.3.4.7. Creating multiple objects

The **new-app** command allows creating multiple applications specifying multiple parameters to **new-app**. Labels specified in the command line apply to all objects created by the single command. Environment variables apply to all components created from source or images.

To create an application from a source repository and a Docker Hub image:

```
$ oc new-app https://github.com/openshift/ruby-hello-world mysql
```



#### NOTE

If a source code repository and a builder image are specified as separate arguments, **new-app** uses the builder image as the builder for the source code repository. If this is not the intent, specify the required builder image for the source using the `~` separator.

### 2.3.4.8. Grouping images and source in a single pod

The **new-app** command allows deploying multiple images together in a single pod. In order to specify which images to group together, use the `+` separator. The **--group** command line argument can also be used to specify the images that should be grouped together. To group the image built from a source repository with other images, specify its builder image in the group:

```
$ oc new-app ruby+mysql
```

To deploy an image built from source and an external image together:

```
$ oc new-app \
  ruby~https://github.com/openshift/ruby-hello-world \
  mysql \
  --group=ruby+mysql
```

### 2.3.4.9. Searching for images, templates, and other inputs

To search for images, templates, and other inputs for the **oc new-app** command, add the **--search** and **--list** flags. For example, to find all of the images or templates that include PHP:

```
$ oc new-app --search php
```

## 2.4. VIEWING APPLICATION COMPOSITION USING THE TOPOLOGY VIEW

The **Topology** view in the **Developer** perspective of the web console provides a visual representation of all the applications within a project, their build status, and the components and services associated with them.

### 2.4.1. Prerequisites


To view your applications in the **Topology** view and interact with them, ensure that:



- You have [logged in to the web console](#) .
- You are in the [Developer perspective](#).
- You have the appropriate [roles and permissions](#) in a project to create applications and other workloads in OpenShift Container Platform.
- You have [created and deployed an application on OpenShift Container Platform using the Developer perspective](#).

## 2.4.2. Viewing the topology of your application









You can navigate to the **Topology** view using the left navigation panel in the **Developer** perspective. After you create an application, you are directed automatically to the **Topology** view where you can see the status of the application pods, quickly access the application on a public URL, access the source code to modify it, and see the status of your last build. You can zoom in and out to see more details for a particular application.

A serverless application is visually indicated with the Knative symbol (  ).



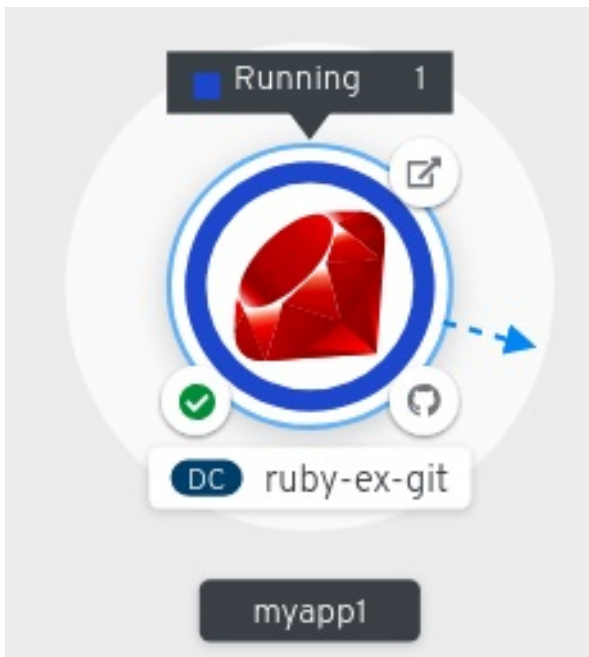
### NOTE

Serverless applications take some time to load and display on the **Topology** view. When you create a serverless application, it first creates a service resource and then a revision. After that it is deployed and displayed on the Topology view. If it is the only workload, you might be redirected to the **Add** page. Once the revision is deployed, the serverless application is displayed on the **Topology** view.

The status or phase of the Pod is indicated by different colors and tooltips as **Running** (  ), **Not Ready** (  ), **Warning**(  ), **Failed**(  ), **Pending**(  ), **Succeeded**(  ), **Terminating**(  ), or **Unknown**(  ). For more information about pod status, see the [Kubernetes documentation](#).

After you create an application and an image is deployed, the status is shown as **Pending**. After the application is built, it is displayed as **Running**.

Figure 2.3. Application topology




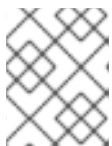
The application resource name is appended with indicators for the different types of resource objects as follows:

- **DC: DeploymentConfig**
- **D: Deployment**
- **SS: StatefulSet**
- **DS: DaemonSet**

### 2.4.3. Interacting with the application and the components







The **Topology** view in the **Developer** perspective of the web console provides the following options to interact with the application and the components:

- Click **Open URL** (  ) to see your application exposed by the route on a public URL.
- Click **Edit Source code** to access your source code and modify it.



#### NOTE

This feature is available only when you create applications using the **From Git**, **From Catalog**, and the **From Dockerfile** options.

- Hover your cursor over the lower left icon on the Pod to see the name of the latest build and its status. The status of the application build is indicated as **New** (  ), **Pending** (  ), **Running** (  ), **Completed** (  ), **Failed** (  ), and **Canceled** (  ).
- Use the **Shortcuts** menu listed on the upper-right of the screen to navigate components in the **Topology** view.
- Use the **List View** icon to see a list of all your applications and use the **Topology View** icon to switch back to the **Topology** view.

### 2.4.4. Scaling application pods and checking builds and routes

The **Topology** view provides the details of the deployed components in the **Overview** panel. You can use the **Overview** and **Resources** tabs to scale the application pods, check build status, services, and routes as follows:

- Click on the component node to see the **Overview** panel to the right. Use the **Overview** tab to:
  - Scale your pods using the up and down arrows to increase or decrease the number of instances of the application manually. For serverless applications, the pods are automatically scaled down to zero when idle and scaled up depending on the channel traffic.
  - Check the **Labels**, **Annotations**, and **Status** of the application.
- Click the **Resources** tab to:
  - See the list of all the pods, view their status, access logs, and click on the pod to see the pod details.
  - See the builds, their status, access logs, and start a new build if needed.
  - See the services and routes used by the component.

For serverless applications, the **Resources** tab provides information on the revision, routes, and the configurations used for that component.

### 2.4.5. Grouping multiple components within an application

You can use the **Add** page to add multiple components or services to your project and use the **Topology** page to group applications and resources within an application group. The following procedure adds a MongoDB database service to an existing application with a Node.js component.

#### Prerequisites

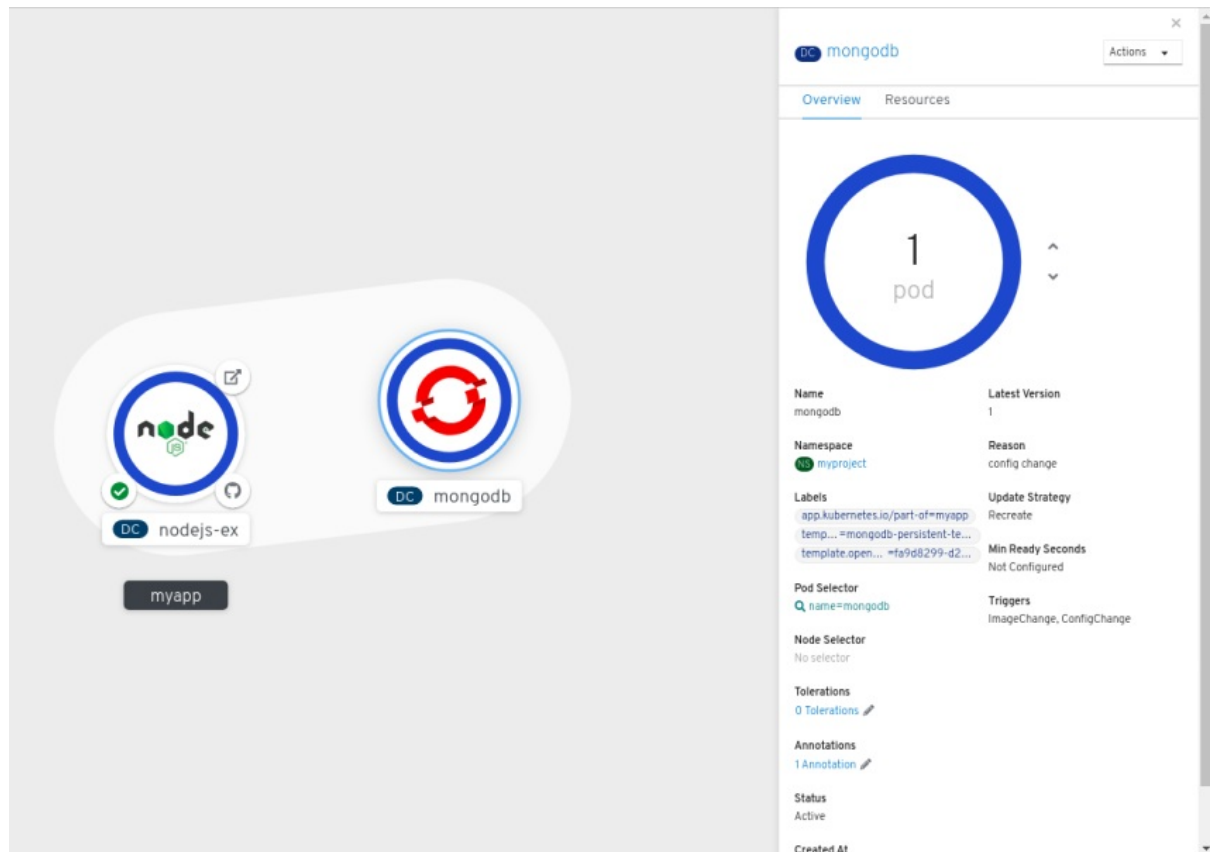
- Ensure that you have created and deployed a Node.js application on OpenShift Container Platform using the **Developer** perspective.

#### Procedure

1. Create and deploy the MongoDB service to your project as follows:
  - a. In the **Developer** perspective, navigate to the **Add** view and select the **Database** option to see the **Developer Catalog**, which has multiple options that you can add as components or services to your application.
  - b. Click on the **MongoDB** option to see the details for the service.
  - c. Click **Instantiate Template** to see an automatically populated template with details for the MongoDB service, and click **Create** to create the service.
2. On the left navigation panel, click **Topology** to see the MongoDB service deployed in your project.
3. To add the MongoDB service to the existing application group, select the **mongodb** pod and drag it to the application; the MongoDB service is added to the existing application group.

4. Dragging a component and adding it to an application group automatically adds the required labels to the component. Click on the MongoDB service node to see the label **app.kubernetes.io/part-of=myapp** added to the **Labels** section in the **Overview** Panel.

Figure 2.4. Application grouping



Alternatively, you can also add the component to an application as follows:

1. To add the MongoDB service to your application, click on the **mongodb** pod to see the **Overview** panel to the right.
2. Click the **Actions** drop-down menu on the upper right of the panel and select **Edit Application Grouping**.
3. In the **Edit Application Grouping** dialog box, click the **Select an Application** drop-down list, and select the appropriate application group.
4. Click **Save** to see the MongoDB service added to the application group.

You can remove a component from an application group by selecting the component and using **Shift+** drag to drag it out of the application group.

### 2.4.6. Connecting components within an application and across applications

In addition to grouping multiple components within an application, you can also use the **Topology** view to connect components with each other. You can either use a binding connector or a visual one to connect components.

A binding connection between the components can be established only if the target node is an Operator-backed service. This is indicated by the **Create a binding connector** tool-tip which appears when you drag an arrow to such a target node. When an application is connected to a service using a binding connector a service binding request is created. Then, the Service Binding Operator controller

uses an intermediate secret to inject the necessary binding data into the application deployment as environment variables. After the request is successful, the application is redeployed establishing an interaction between the connected components.

A visual connector establishes only a visual connection between the components, depicting an intent to connect. No interaction between the components is established. If the target node is not an Operator-backed service the **Create a visual connector** tool-tip is displayed when you drag an arrow to a target node.

### 2.4.6.1. Creating a visual connection between components

You can depict an intent to connect application components using the visual connector.

This procedure walks through an example of creating a visual connection between a MongoDB service and a Node.js application.

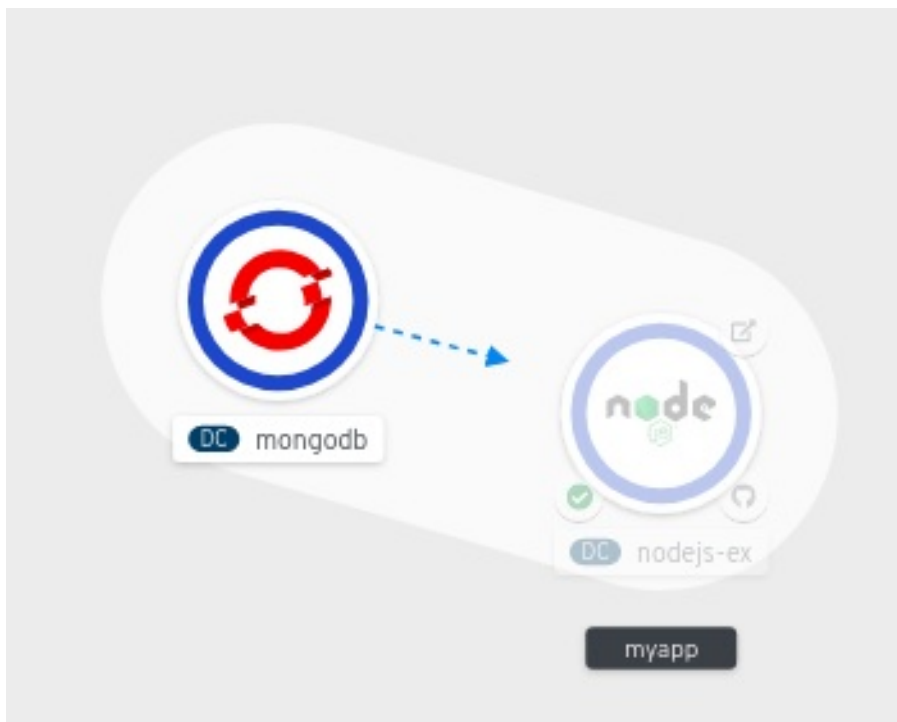
#### Prerequisites

- Ensure that you have created and deployed a Node.js application using the **Developer** perspective.
- Ensure that you have created and deployed a MongoDB service using the **Developer** perspective.

#### Procedure

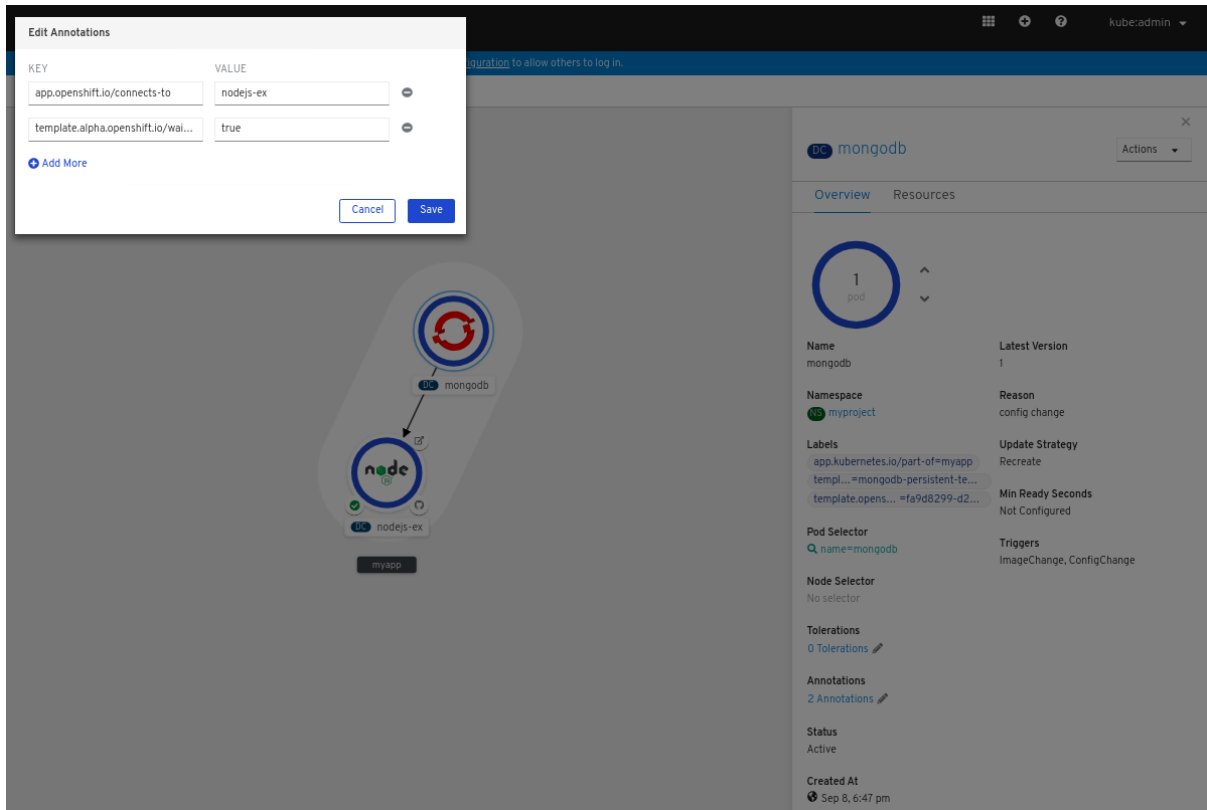
1. Hover over the MongoDB service to see a dangling arrow on the node.

**Figure 2.5. Connector**



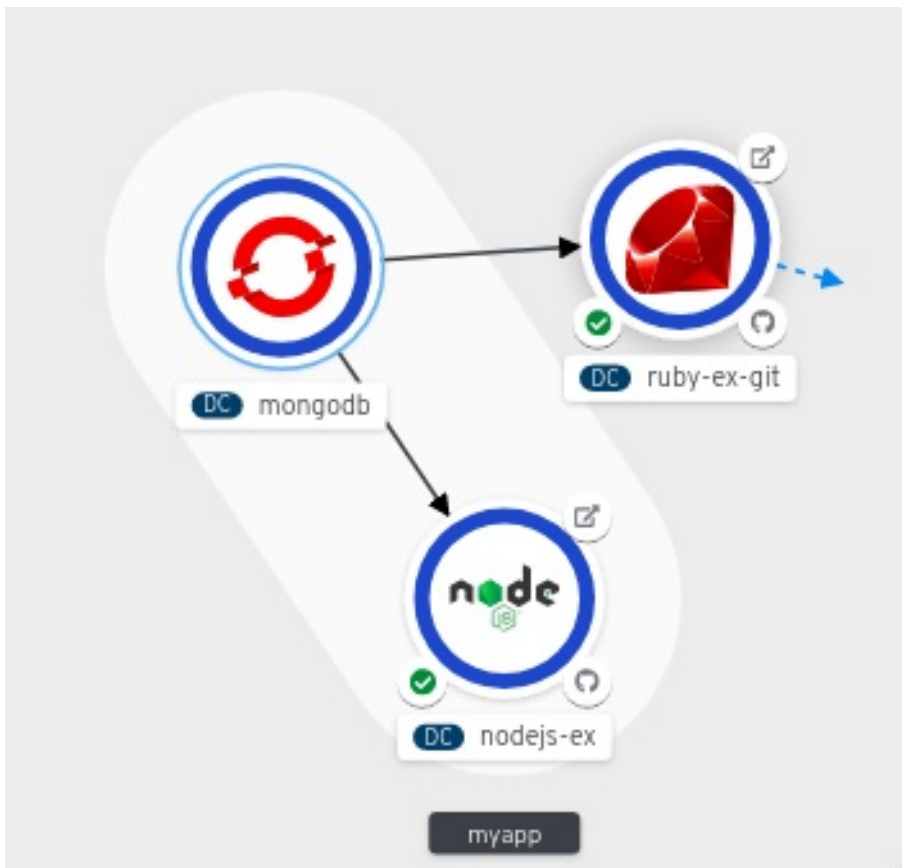
2. Click and drag the arrow towards the Node.js component to connect the MongoDB service with it.

- Click on the MongoDB service to see the **Overview** Panel. In the **Annotations** section, click the edit icon to see the **Key = app.openshift.io/connects-to** and **Value = nodejs-ex** annotation added to the service.



Similarly you can create other applications and components and establish connections between them.

Figure 2.6. Connecting multiple applications



### 2.4.6.2. Creating a binding connection between components



#### IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.



#### NOTE

Currently, a few specific Operators like the **etcd** and the **PostgreSQL Database Operator**'s service instances are bindable.

You can establish a binding connection with Operator-backed components.

This procedure walks through an example of creating a binding connection between a PostgreSQL Database service and a Node.js application. To create a binding connection with a service that is backed by the PostgreSQL Database Operator, you must first add the Red Hat-provided PostgreSQL Database Operator to the OperatorHub using a backing Operator source, and then install the Operator.

#### Prerequisite

- Ensure that you have created and deployed a Node.js application using the **Developer** perspective.
- Ensure that you have installed the **Service Binding Operator** from OperatorHub.

#### Procedure

1. Create a backing Operator source that adds the PostgreSQL Operator provided by Red Hat to the OperatorHub. A backing Operator source exposes the binding information in secrets, config maps, status, and spec attributes.
  - a. In the **Add** view, click the **YAML** option to see the **Import YAML** screen.
  - b. Add the following YAML file to apply the Operator source:

```
apiVersion: operators.coreos.com/v1
kind: OperatorSource
metadata:
  name: db-operators
  namespace: openshift-marketplace
spec:
  type: appregistry
  endpoint: https://quay.io/cnr
  registryNamespace: pmacik
```

- c. Click **Create** to create the Operator source in your cluster.
2. Install the Red Hat-provided PostgreSQL Database Operator:

- a. In the **Administrator** perspective of the console, navigate to **Operators → OperatorHub**.
  - b. In the **Database** category, select the **PostgreSQL Database** Operator and install it.
3. Create a database (DB) instance for the application:
    - a. Switch to the **Developer** perspective and ensure that you are in the appropriate project.
    - b. In the **Add** view, click the **YAML** option to see the **Import YAML** screen.
    - c. Add the service instance YAML in the editor and click **Create** to deploy the service. Following is an example of what the service YAML will look like:

```

apiVersion: postgresql.baiju.dev/v1alpha1
kind: Database
metadata:
  name: db-demo
  namespace: test-project
spec:
  image: docker.io/postgres
  imageName: postgres
  dbName: db-demo
  
```

A DB instance is now deployed in the **Topology** view.

4. In the **Topology** view, hover over the Node.js component to see a dangling arrow on the node.
5. Click and drag the arrow towards the **db-demo-postgresql** service to make a binding connection with the Node.js application. A service binding request is created and the **Service Binding Operator** controller injects the DB connection information into the application deployment as environment variables. After the request is successful, the application is redeployed and the connection is established.

**Figure 2.7. Binding connector**



### 2.4.7. Labels and annotations used for the Topology view

The **Topology** view uses the following labels and annotations:

Icon displayed in the node



Icons in the node are defined by looking for matching icons using the **app.openshift.io/runtime** label, followed by the **app.kubernetes.io/name** label. This matching is done using a predefined set of icons.

### Link to the source code editor or the source

The **app.openshift.io/vcs-uri** annotation is used to create links to the source code editor.

### Node Connector

The **app.openshift.io/connects-to** annotation is used to connect the nodes.

### App grouping

The **app.kubernetes.io/part-of=<appname>** label is used to group the applications, services, and components.

For detailed information on the labels and annotations OpenShift Container Platform applications must use, see [Guidelines for labels and annotations for OpenShift applications](#).

## 2.5. EDITING APPLICATIONS

You can edit the configuration and the source code of the application you create using the **Topology** view.

### 2.5.1. Prerequisites

- You have [logged in to the web console](#) and have switched to the **Developer perspective**.
- You have the appropriate [roles and permissions](#) in a project to create and modify applications in OpenShift Container Platform.
- You have [created and deployed an application on OpenShift Container Platform using the Developer perspective](#).

### 2.5.2. Editing the source code of an application using the Developer perspective

You can use the **Topology** view in the **Developer** perspective to edit the source code of your application.



#### Procedure

- In the **Topology** view, click the **Edit Source code** icon, displayed at the bottom-right of the deployed application, to access your source code and modify it.



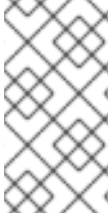
#### NOTE

This feature is available only when you create applications using the **From Git**, **From Catalog**, and the **From Dockerfile** options.

If the **Eclipse Che** Operator is installed in your cluster, a Che workspace (  ) is created and you are directed to the workspace to edit your source code. If it is not installed, you will be directed to the Git repository (  ) your source code is hosted in.

### 2.5.3. Editing the application configuration using the Developer perspective

You can use the **Topology** view in the **Developer** perspective to edit the configuration of your application.



#### NOTE

Currently, only configurations of applications created by using the **From Git, Container Image, From Catalog, or From Dockerfile** options in the **Add** workflow of the **Developer** perspective can be edited. Configurations of applications created by using the CLI or the **YAML** option from the **Add** workflow cannot be edited.

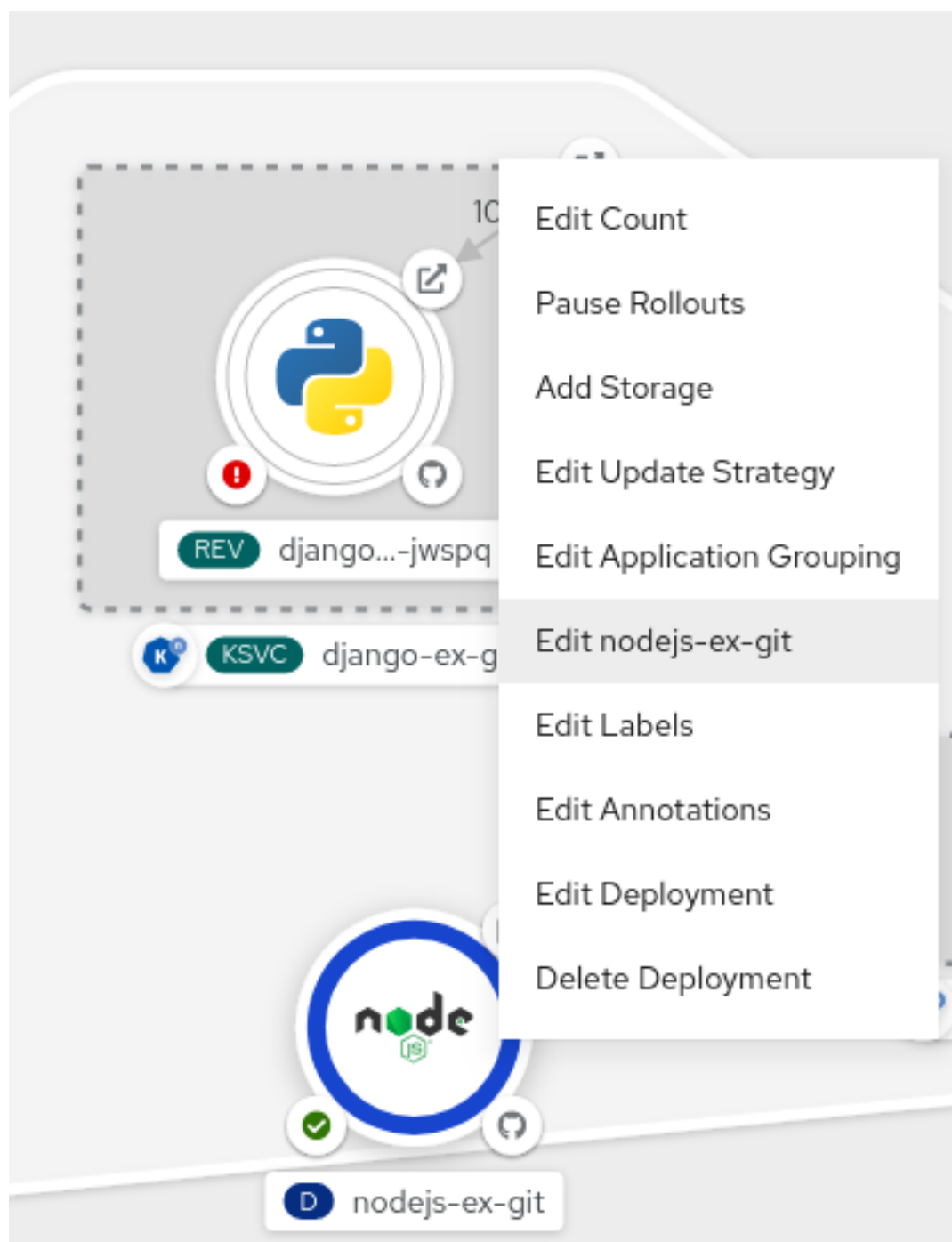
#### Prerequisites

Ensure that you have created an application using the **From Git, Container Image, From Catalog, or From Dockerfile** options in the **Add** workflow.

#### Procedure

1. After you have created an application and it is displayed in the **Topology** view, right-click the application to see the edit options available.

Figure 2.8. Edit application



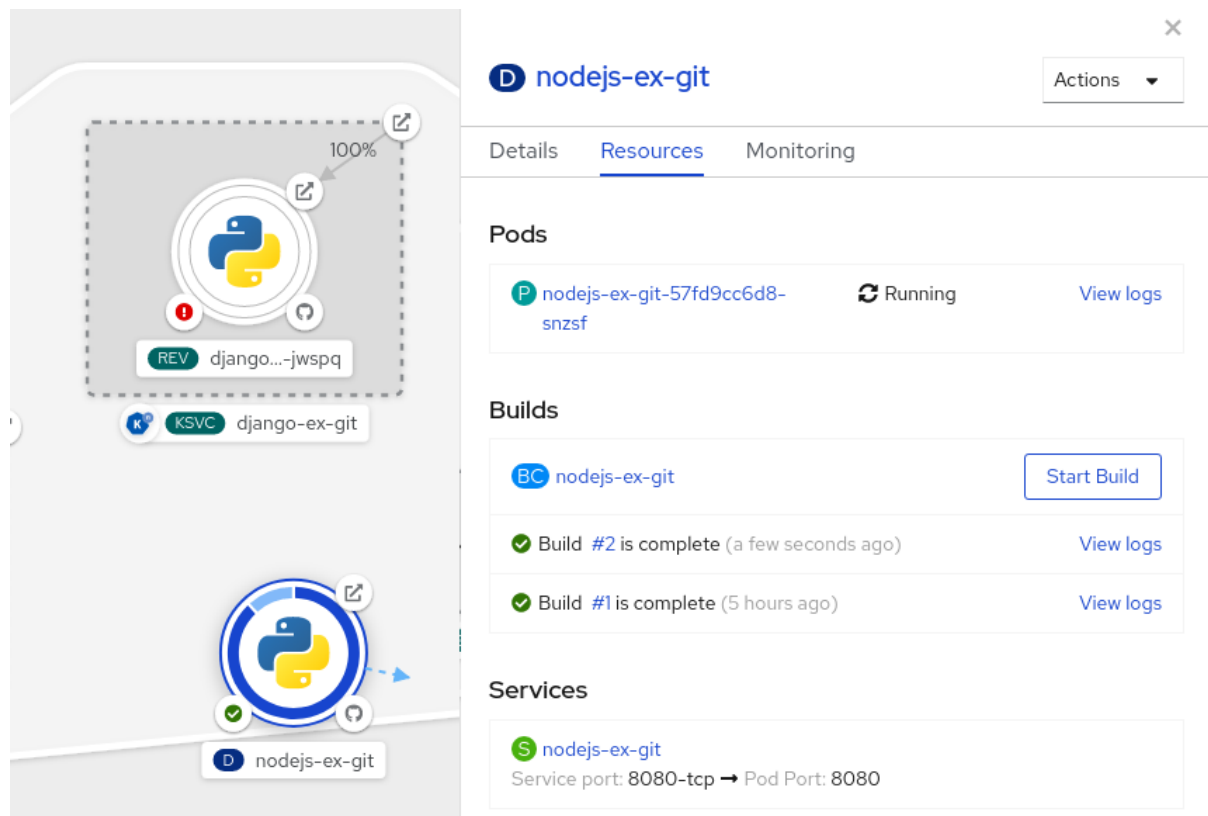
2. Click **Edit application-name** to see the **Add** workflow you used to create the application. The form is pre-populated with the values you had added while creating the application.
3. Edit the necessary values for the application.

**NOTE**

You cannot edit the **Name** field in the **General** section, the CI/CD pipelines, or the **Create a route to the application** field in the **Advanced Options** section.

4. Click **Save** to restart the build and deploy a new image.

Figure 2.9. Edit and redeploy application



## 2.6. WORKING WITH HELM CHARTS USING THE DEVELOPER PERSPECTIVE

### 2.6.1. Understanding Helm

Helm is a command-line interface (CLI) tool that simplifies deployment of applications and services to OpenShift Container Platform clusters. Helm uses a packaging format called *charts*. A Helm chart is a collection of files that describes OpenShift Container Platform resources. A running instance of the chart in a cluster is called a *release*. A new release is created every time a chart is installed on the cluster.

#### 2.6.1.1. Key features

Helm provides the ability to:

- Search through a large collection of charts stored in the chart repository.
- Modify existing charts.
- Create your own charts with OpenShift Container Platform or Kubernetes resources.
- Package and share your applications as charts.

You can use the [Developer perspective](#) in the [web console](#) to create Helm releases.

### 2.6.2. Installing Helm charts

You can use the **Developer** perspective to create Helm releases from the Helm charts provided in the **Developer Catalog**.

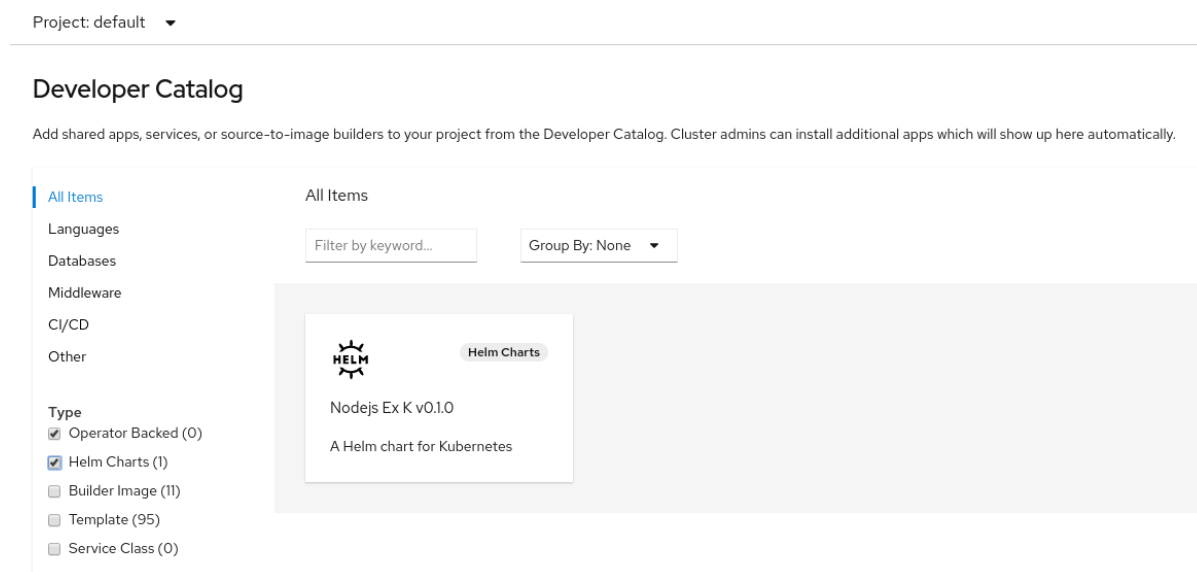
## Prerequisites

- You have logged in to the web console and have switched to the **Developer** perspective.

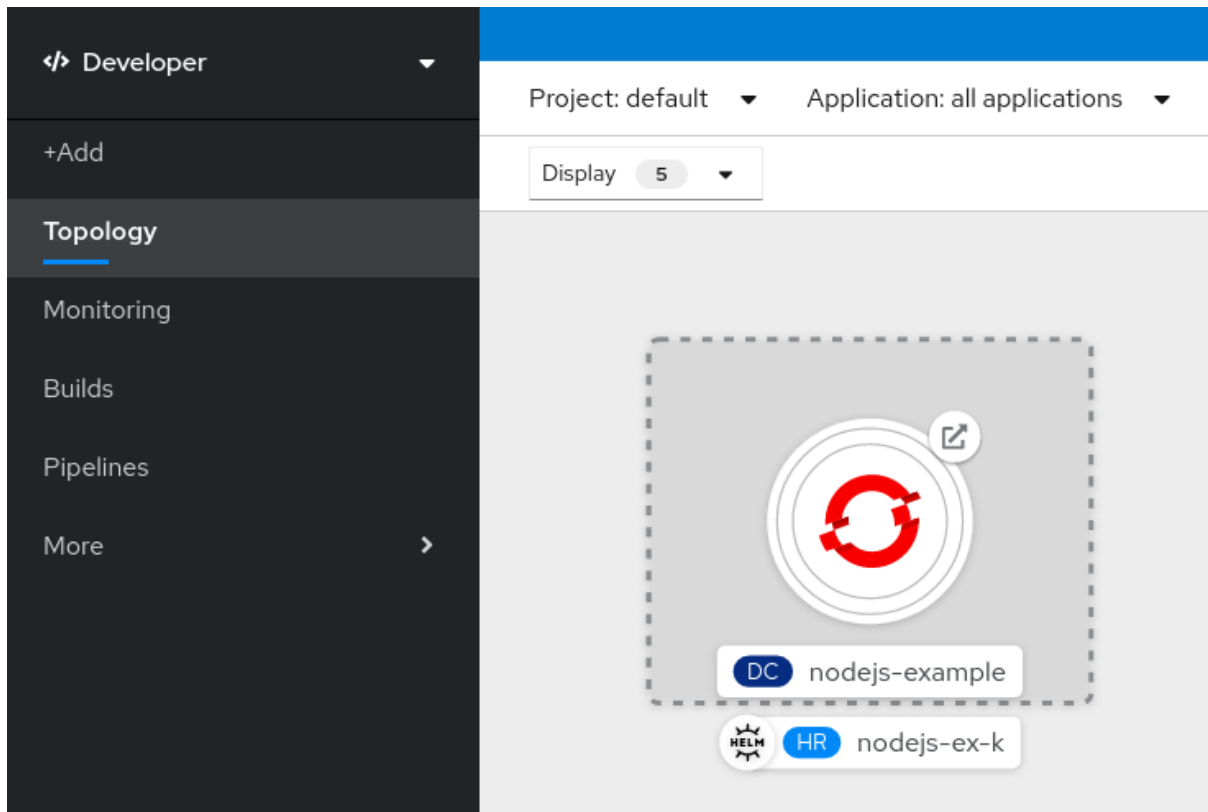
## Procedure

1. In the **Developer** perspective, navigate to the **Add** view and select the **From Catalog** option to see the **Developer Catalog**.
2. In the filters listed on the left, under **Type**, select the **Helm Charts** filter to see the available Helm charts.

**Figure 2.10. Helm charts in developer catalog**



3. Select the Node.js chart to see the description of the chart and click **Install Helm Chart**.
4. In the **Install Helm Chart** page:
  - a. Enter a unique name for the release in the **Release Name** field.
  - b. Optionally, in the YAML editor, modify the YAML file.
5. Click **Install** to create a Helm release. The release is displayed in the **Topology** view.



6. In the left panel of the **Developer** perspective, click **More > Helm** to see the list of all the Helm releases in the namespace.
7. Click on a listed Helm release to see the details and resources for that release.

## 2.7. DELETING APPLICATIONS

You can delete applications created in your project.

### 2.7.1. Deleting applications using the Developer perspective

You can delete an application and all of its associated components using the **Topology** view in the **Developer** perspective:

1. Click the application you want to delete to see the side panel with the resource details of the application.
2. Click the **Actions** drop-down menu displayed on the upper right of the panel, and select **Delete Application** to see a confirmation dialog box.
3. Enter the name of the application and click **Delete** to delete it.

You can also right-click the application you want to delete and click **Delete Application** to delete it.

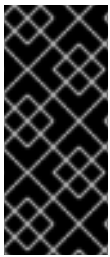
## CHAPTER 3. SERVICE BROKERS

### 3.1. INSTALLING SERVICE CATALOG



#### IMPORTANT

Service Catalog is deprecated in OpenShift Container Platform 4. Equivalent and better functionality is present in the Operator Framework and Operator Lifecycle Manager (OLM).



#### IMPORTANT

Starting in OpenShift Container Platform 4.4, the **service-catalog-controller-manager** and **service-catalog-apiserver** cluster Operators are now set to **Upgradeable=false**. This means that they will block future cluster upgrades to the next minor version, for example 4.5, if they are still installed at that time. Upgrading to z-stream releases such as 4.4.z, however, are still permitted in this state.

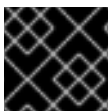
#### 3.1.1. About Service Catalog

When developing microservices-based applications to run on cloud native platforms, there are many ways to provision different resources and share their coordinates, credentials, and configuration, depending on the service provider and the platform.

To give developers a more seamless experience, OpenShift Container Platform includes *Service Catalog*, an implementation of the [Open Service Broker API](#) (OSB API) for Kubernetes. This allows users to connect any of their applications deployed in OpenShift Container Platform to a wide variety of service brokers.

Service Catalog allows cluster administrators to integrate multiple platforms using a single API specification. The OpenShift Container Platform web console displays the cluster service classes offered by service brokers in Service Catalog, allowing users to discover and instantiate those services for use with their applications.

As a result, service users benefit from ease and consistency of use across different types of services from different providers, while service providers benefit from having one integration point that gives them access to multiple platforms.



#### IMPORTANT

Service Catalog is not installed by default in OpenShift Container Platform 4.

#### 3.1.2. Installing Service Catalog

If you plan on using any of the services from the Template Service Broker, you must install Service Catalog by completing the following steps.

The custom resources for Service Catalog's API server and controller manager are created by default in OpenShift Container Platform, but initially have a **managementState** of **Removed**. To install Service Catalog, you must change the **managementState** for these resources to **Managed**.

#### Procedure

1. Enable Service Catalog's API server.

- a. Use the following command to edit Service Catalog's API server resource.

```
$ oc edit servicecatalogapiservers
```

- b. Under **spec**, set the **managementState** field to **Managed**:

```
spec:  
  logLevel: Normal  
  managementState: Managed
```

- c. Save the file to apply the changes.

The Operator installs Service Catalog's API server component. As of OpenShift Container Platform 4, this component is installed into the **openshift-service-catalog-apiserver** namespace.

2. Enable Service Catalog's controller manager.

- a. Use the following command to edit Service Catalog's controller manager resource.

```
$ oc edit servicecatalogcontrollermanagers
```

- b. Under **spec**, set the **managementState** field to **Managed**:

```
spec:  
  logLevel: Normal  
  managementState: Managed
```

- c. Save the file to apply the changes.

The Operator installs Service Catalog's controller manager component. As of OpenShift Container Platform 4, this component is installed into the **openshift-service-catalog-controller-manager** namespace.

3. Verify that the installation is completed successfully by checking that **Service Catalog** appears in the left navigation of the web console.

## 3.2. UNINSTALLING SERVICE CATALOG



### IMPORTANT

Starting in OpenShift Container Platform 4.4, the **service-catalog-controller-manager** and **service-catalog-apiserver** cluster Operators are now set to **Upgradeable=false**. This means that they will block future cluster upgrades to the next minor version, for example 4.5, if they are still installed at that time. Upgrading to z-stream releases such as 4.4.z, however, are still permitted in this state.

You can uninstall Service Catalog if you have installed it previously. Service Catalog must be removed before the cluster can upgrade to future minor versions of OpenShift Container Platform.

### 3.2.1. Uninstalling Service Catalog

If Service Catalog is installed, cluster administrators can uninstall it by using the following procedure.



**WARNING**

Uninstalling Service Catalog will impact any service brokers and provisioned services from those brokers, such as the Template Service Broker, in your cluster.

**Prerequisites**

- Service Catalog is installed.

**Procedure**

1. Using an account with **cluster-admin** privileges, edit the **servicecatalogcontrollermanagers** resource:

```
$ oc edit servicecatalogcontrollermanagers
```

2. Change the **managementState** parameter from **Managed** back to the default **Removed**, and save your changes.

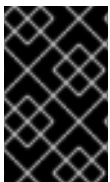
3. Edit the **servicecatalogapiservers** resource:

```
$ oc edit servicecatalogapiservers
```

4. Change the **managementState** parameter from **Managed** back to the default **Removed**, and save your changes.
5. Verify that the removal is completed successfully by checking that **Service Catalog** is removed from the left navigation of the web console.

**3.3. INSTALLING THE TEMPLATE SERVICE BROKER**

You can install the Template Service Broker to gain access to the template applications that it provides.

**IMPORTANT**

The Template Service Broker is deprecated in OpenShift Container Platform 4. Equivalent and better functionality is present in the Operator Framework and Operator Lifecycle Manager (OLM).

**NOTE**

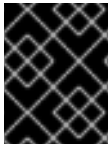
By default, the Samples Operator handles Red Hat Enterprise Linux (RHEL)-based OpenShift Container Platform imagestreams and OpenShift Container Platform templates. See [Configuring the Samples Operator](#) for details.

**3.3.1. About the Template Service Broker**

The *Template Service Broker* gives Service Catalog visibility into the default Instant App and Quickstart templates that have shipped with OpenShift Container Platform since its initial release. The Template

Service Broker can also make available as a service anything for which an OpenShift Container Platform template has been written, whether provided by Red Hat, a cluster administrator or user, or a third-party vendor.

By default, the Template Service Broker shows objects that are globally available from the **openshift** project. It can also be configured to watch any other project that a cluster administrator chooses.



### IMPORTANT

The Template Service Broker is not installed by default in OpenShift Container Platform 4.

## 3.3.2. Installing the Template Service Broker Operator

### Prerequisites

- Service Catalog is installed.

### Procedure

The following procedure installs the Template Service Broker Operator using the web console.

1. Create a namespace.
    - a. Using the **Administrator** perspective, navigate in the web console to **Administration** → **Namespaces** and click **Create Namespace**.
    - b. Enter the following:
      - **openshift-template-service-broker** in the **Name** field
  - ### NOTE

The namespace must start with **openshift-**.
  - **openshift.io/cluster-monitoring=true** in the **Labels** field
  - c. Click **Create**.
2. Navigate to the **Operators** → **OperatorHub** page. Verify that the **openshift-template-service-broker** project is selected.
  3. Select **Template Service Broker Operator**.
  4. Read the information about the Operator and click **Install**.
  5. Review the default selections and click **Subscribe**.

Next, you must start the Template Service Broker in order to access the template applications it provides.

## 3.3.3. Starting the Template Service Broker

After you have installed the Template Service Broker Operator, you can start the Template Service Broker using the following procedure.

## Prerequisites

- Service Catalog is installed.
- The Template Service Broker Operator is installed.

## Procedure

1. Using the **Administrator** perspective, navigate in the web console to **Operators** → **Installed Operators** and select the **openshift-template-service-broker** project.
2. Select the **Template Service Broker Operator**.
3. Under **Provided APIs**, click **Create Instance** for **Template Service Broker**.
4. Review the default YAML and click **Create**.
5. Verify that the Template Service Broker starts correctly by checking that the template applications are available.
  - To check from the web console, navigate to **Service Catalog** → **Broker Management** → **Service Classes** to view the list of template application service classes.
  - To check from the CLI:

```
$ oc get ClusterServiceClasses -n openshift-template-service-broker
```



### NOTE

It may take a few minutes for the Template Service Broker to start and the template applications to be available.

If you do not yet see these service classes, you can check the status of the following items:

- Template Service Broker Pod status
  - From the **Workloads** → **Pods** page for the **openshift-template-service-broker** project, verify that the Pod that starts with **apiserver-** has a status of **Running** and readiness of **Ready**.
- Cluster service broker status
  - From the **Service Catalog** → **Broker Management** → **Service Brokers** page, verify that the **template-service-broker** service broker has a status of **Ready**.
- Service Catalog controller manager Pod logs
  - From the **Workloads** → **Pods** page for the **openshift-service-catalog-controller-manager** project, review the logs for each of the Pods and verify that you see a log entry with the message **Successfully fetched catalog entries from broker**.

## 3.4. UNINSTALLING THE TEMPLATE SERVICE BROKER

You can uninstall the Template Service Broker if you no longer require access to the template applications that it provides.



### IMPORTANT

The Template Service Broker is deprecated in OpenShift Container Platform 4. Equivalent and better functionality is present in the Operator Framework and Operator Lifecycle Manager (OLM).



### NOTE

By default, the Samples Operator handles Red Hat Enterprise Linux (RHEL)-based OpenShift Container Platform imagestreams and OpenShift Container Platform templates. See [Configuring the Samples Operator](#) for details.

## 3.4.1. Uninstalling the Template Service Broker

The following procedure uninstalls the Template Service Broker and its Operator using the web console.



### WARNING

Do not uninstall the Template Service Broker if there are any provisioned services from it in your cluster, otherwise you might encounter errors when trying to manage the services.

### Prerequisites

- The Template Service Broker is installed.

### Procedure

This procedure assumes that you installed the Template Service Broker into the **openshift-template-service-broker** project.

1. Uninstall the Template Service Broker.
  - a. Navigate to **Operators** → **Installed Operators** and select the **openshift-template-service-broker** project from the drop-down menu.
  - b. Click **Template Service Broker Operator**.
  - c. Select the **Template Service Broker** tab.
  - d. Click **template-service-broker**.
  - e. From the **Actions** drop-down menu, select **Delete Template Service Broker**.
  - f. Click **Delete** from the confirmation pop-up window.  
The Template Service Broker is now uninstalled, and template applications will soon be removed from the Developer Catalog.
2. Uninstall the Template Service Broker Operator.
  - a. From the **Operators** → **Installed Operators** page, scroll or type a keyword into the **Filter by name** to find the Template Service Broker Operator, then click on it.

- b. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
- c. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the Pods, Deployments, CRDs, and CRs associated with the Operator.
- d. Select **Remove**. This Operator will stop running and no longer receive updates. The Template Service Broker Operator is no longer installed in your cluster.

After the Template Service Broker is uninstalled, users will no longer have access to the template applications provided by the Template Service Broker.

## CHAPTER 4. DEPLOYMENTS

### 4.1. UNDERSTANDING DEPLOYMENT AND DEPLOYMENTCONFIG OBJECTS

The **Deployment** and **DeploymentConfig** API objects in OpenShift Container Platform provide two similar but different methods for fine-grained management over common user applications. They are composed of the following separate API objects:

- A **DeploymentConfig** or **Deployment** object, either of which describes the desired state of a particular component of the application as a pod template.
- **DeploymentConfig** objects involve one or more *replication controllers*, which contain a point-in-time record of the state of a deployment as a pod template. Similarly, **Deployment** objects involve one or more *replica sets*, a successor of replication controllers.
- One or more pods, which represent an instance of a particular version of an application.

#### 4.1.1. Building blocks of a deployment

Deployments and deployment configs are enabled by the use of native Kubernetes API objects **ReplicaSet** and **ReplicationController**, respectively, as their building blocks.

Users do not have to manipulate replication controllers, replica sets, or pods owned by **DeploymentConfig** objects or deployments. The deployment systems ensure changes are propagated appropriately.

#### TIP

If the existing deployment strategies are not suited for your use case and you must run manual steps during the lifecycle of your deployment, then you should consider creating a custom deployment strategy.

The following sections provide further details on these objects.

##### 4.1.1.1. Replication controllers

A replication controller ensures that a specified number of replicas of a pod are running at all times. If pods exit or are deleted, the replication controller acts to instantiate more up to the defined number. Likewise, if there are more running than desired, it deletes as many as necessary to match the defined amount.

A replication controller configuration consists of:

- The number of replicas desired, which can be adjusted at run time.
- A **Pod** definition to use when creating a replicated pod.
- A selector for identifying managed pods.

A selector is a set of labels assigned to the pods that are managed by the replication controller. These labels are included in the **Pod** definition that the replication controller instantiates. The replication controller uses the selector to determine how many instances of the pod are already running in order to adjust as needed.

The replication controller does not perform auto-scaling based on load or traffic, as it does not track either. Rather, this requires its replica count to be adjusted by an external auto-scaler.

The following is an example definition of a replication controller:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 ①
  selector: ②
    name: frontend
  template: ③
    metadata:
      labels: ④
        name: frontend ⑤
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
        restartPolicy: Always
```

- ① The number of copies of the pod to run.
- ② The label selector of the pod to run.
- ③ A template for the pod the controller creates.
- ④ Labels on the pod should include those from the label selector.
- ⑤ The maximum name length after expanding any parameters is 63 characters.

#### 4.1.1.2. Replica sets

Similar to a replication controller, a **ReplicaSet** is a native Kubernetes API object that ensures a specified number of pod replicas are running at any given time. The difference between a replica set and a replication controller is that a replica set supports set-based selector requirements whereas a replication controller only supports equality-based selector requirements.



#### NOTE

Only use replica sets if you require custom update orchestration or do not require updates at all. Otherwise, use deployments. Replica sets can be used independently, but are used by deployments to orchestrate pod creation, deletion, and updates. Deployments manage their replica sets automatically, provide declarative updates to pods, and do not have to manually manage the replica sets that they create.

The following is an example **ReplicaSet** definition:

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend-1
  labels:
    tier: frontend
spec:
  replicas: 3
  selector: ①
  matchLabels: ②
    tier: frontend
  matchExpressions: ③
    - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - image: openshift/hello-openshift
          name: helloworld
          ports:
            - containerPort: 8080
              protocol: TCP
          restartPolicy: Always

```

- ① A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined.
- ② Equality-based selector to specify resources with labels that match the selector.
- ③ Set-based selector to filter keys. This selects all resources with key equal to **tier** and value equal to **frontend**.

### 4.1.2. DeploymentConfig objects

Building on replication controllers, OpenShift Container Platform adds expanded support for the software development and deployment lifecycle with the concept of **DeploymentConfig** objects. In the simplest case, a **DeploymentConfig** object creates a new replication controller and lets it start up pods.

However, OpenShift Container Platform deployments from **DeploymentConfig** objects also provide the ability to transition from an existing deployment of an image to a new one and also define hooks to be run before or after creating the replication controller.

The **DeploymentConfig** deployment system provides the following capabilities:

- A **DeploymentConfig** object, which is a template for running applications.
- Triggers that drive automated deployments in response to events.
- User-customizable deployment strategies to transition from the previous version to the new version. A strategy runs inside a pod commonly referred as the deployment process.



- A set of hooks (lifecycle hooks) for executing custom behavior in different points during the lifecycle of a deployment.
- Versioning of your application in order to support rollbacks either manually or automatically in case of deployment failure.
- Manual replication scaling and autoscaling.

When you create a **DeploymentConfig** object, a replication controller is created representing the **DeploymentConfig** object's pod template. If the deployment changes, a new replication controller is created with the latest pod template, and a deployment process runs to scale down the old replication controller and scale up the new one.

Instances of your application are automatically added and removed from both service load balancers and routers as they are created. As long as your application supports graceful shutdown when it receives the **TERM** signal, you can ensure that running user connections are given a chance to complete normally.

The OpenShift Container Platform **DeploymentConfig** object defines the following details:

1. The elements of a **ReplicationController** definition.
2. Triggers for creating a new deployment automatically.
3. The strategy for transitioning between deployments.
4. Lifecycle hooks.

Each time a deployment is triggered, whether manually or automatically, a deployer pod manages the deployment (including scaling down the old replication controller, scaling up the new one, and running hooks). The deployment pod remains for an indefinite amount of time after it completes the deployment in order to retain its logs of the deployment. When a deployment is superseded by another, the previous replication controller is retained to enable easy rollback if needed.

### Example DeploymentConfig definition

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
  - type: ConfigChange 1
  - imageChangeParams:
      automatic: true
      containerNames:
      - helloworld
    from:
      kind: ImageStreamTag
      name: hello-openshift:latest
```

```

type: ImageChange 2
strategy:
  type: Rolling 3

```

- 1 A config change trigger causes a new deployment to be created any time the replication controller template changes.
- 2 An image change trigger causes a new deployment to be created each time a new version of the backing image is available in the named image stream.
- 3 The default **Rolling** strategy makes a downtime-free transition between deployments.

### 4.1.3. Deployments

Kubernetes provides a first-class, native API object type in OpenShift Container Platform called **Deployment**. **Deployment** objects serve as a descendant of the OpenShift Container Platform-specific **DeploymentConfig** object.

Like **DeploymentConfig** objects, **Deployment** objects describe the desired state of a particular component of an application as a pod template. Deployments create replica sets, which orchestrate pod lifecycles.

For example, the following deployment definition creates a replica set to bring up one **hello-openshift** pod:

#### Deployment definition

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
        - name: hello-openshift
          image: openshift/hello-openshift:latest
          ports:
            - containerPort: 80

```

### 4.1.4. Comparing Deployment and DeploymentConfig objects

Both Kubernetes **Deployment** objects and OpenShift Container Platform-provided **DeploymentConfig** objects are supported in OpenShift Container Platform; however, it is recommended to use **Deployment** objects unless you need a specific feature or behavior provided by **DeploymentConfig** objects.

The following sections go into more detail on the differences between the two object types to further help you decide which type to use.

#### 4.1.4.1. Design

One important difference between **Deployment** and **DeploymentConfig** objects is the properties of the [CAP theorem](#) that each design has chosen for the rollout process. **DeploymentConfig** objects prefer consistency, whereas **Deployments** objects take availability over consistency.

For **DeploymentConfig** objects, if a node running a deployer pod goes down, it will not get replaced. The process waits until the node comes back online or is manually deleted. Manually deleting the node also deletes the corresponding pod. This means that you can not delete the pod to unstick the rollout, as the kubelet is responsible for deleting the associated pod.

However, deployment rollouts are driven from a controller manager. The controller manager runs in high availability mode on masters and uses leader election algorithms to value availability over consistency. During a failure it is possible for other masters to act on the same deployment at the same time, but this issue will be reconciled shortly after the failure occurs.

#### 4.1.4.2. DeploymentConfig object-specific features

##### Automatic rollbacks

Currently, deployments do not support automatically rolling back to the last successfully deployed replica set in case of a failure.

##### Triggers

Deployments have an implicit config change trigger in that every change in the pod template of a deployment automatically triggers a new rollout. If you do not want new rollouts on pod template changes, pause the deployment:

```
$ oc rollout pause deployments/<name>
```

##### Lifecycle hooks

Deployments do not yet support any lifecycle hooks.

##### Custom strategies

Deployments do not support user-specified custom deployment strategies yet.

#### 4.1.4.3. Deployment-specific features

##### Rollover

The deployment process for **Deployment** objects is driven by a controller loop, in contrast to **DeploymentConfig** objects which use deployer pods for every new rollout. This means that the **Deployment** object can have as many active replica sets as possible, and eventually the deployment controller will scale down all old replica sets and scale up the newest one.

**DeploymentConfig** objects can have at most one deployer pod running, otherwise multiple deployers end up conflicting while trying to scale up what they think should be the newest replication controller. Because of this, only two replication controllers can be active at any point in time. Ultimately, this translates to faster rapid rollouts for **Deployment** objects.

##### Proportional scaling

Because the deployment controller is the sole source of truth for the sizes of new and old replica sets owned by a deployment, it is able to scale ongoing rollouts. Additional replicas are distributed proportionally based on the size of each replica set.

Deployments cannot be scaled when a rollout is ongoing because the controller will end up having issues with the deployer process about the size of the new replication controller.

### Pausing mid-rollout

Deployments can be paused at any point in time, meaning you can also pause ongoing rollouts. On the other hand, you cannot pause deployer pods currently, so if you try to pause a deployment in the middle of a rollout, the deployer process will not be affected and will continue until it finishes.

## 4.2. MANAGING DEPLOYMENT PROCESSES

### 4.2.1. Managing DeploymentConfig objects

**DeploymentConfig** objects can be managed from the OpenShift Container Platform web console's **Workloads** page or using the **oc** CLI. The following procedures show CLI usage unless otherwise stated.

#### 4.2.1.1. Starting a deployment

You can start a rollout to begin the deployment process of your application.

##### Procedure

1. To start a new deployment process from an existing **DeploymentConfig** object, run the following command:

```
$ oc rollout latest dc/<name>
```



##### NOTE

If a deployment process is already in progress, the command displays a message and a new replication controller will not be deployed.

#### 4.2.1.2. Viewing a deployment

You can view a deployment to get basic information about all the available revisions of your application.

##### Procedure

1. To show details about all recently created replication controllers for the provided **DeploymentConfig** object, including any currently running deployment process, run the following command:

```
$ oc rollout history dc/<name>
```

2. To view details specific to a revision, add the **--revision** flag:

```
$ oc rollout history dc/<name> --revision=1
```

3. For more detailed information about a **DeploymentConfig** object and its latest revision, use the **oc describe** command:

```
$ oc describe dc <name>
```

### 4.2.1.3. Retrying a deployment

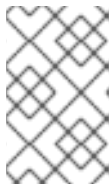
If the current revision of your **DeploymentConfig** object failed to deploy, you can restart the deployment process.

#### Procedure

1. To restart a failed deployment process:

```
$ oc rollout retry dc/<name>
```

If the latest revision of it was deployed successfully, the command displays a message and the deployment process is not retried.



#### NOTE

Retrying a deployment restarts the deployment process and does not create a new deployment revision. The restarted replication controller has the same configuration it had when it failed.

### 4.2.1.4. Rolling back a deployment

Rollbacks revert an application back to a previous revision and can be performed using the REST API, the CLI, or the web console.

#### Procedure

1. To rollback to the last successful deployed revision of your configuration:

```
$ oc rollout undo dc/<name>
```

The **DeploymentConfig** object's template is reverted to match the deployment revision specified in the undo command, and a new replication controller is started. If no revision is specified with **--to-revision**, then the last successfully deployed revision is used.

2. Image change triggers on the **DeploymentConfig** object are disabled as part of the rollback to prevent accidentally starting a new deployment process soon after the rollback is complete. To re-enable the image change triggers:

```
$ oc set triggers dc/<name> --auto
```



#### NOTE

Deployment configs also support automatically rolling back to the last successful revision of the configuration in case the latest deployment process fails. In that case, the latest template that failed to deploy stays intact by the system and it is up to users to fix their configurations.

### 4.2.1.5. Executing commands inside a container

You can add a command to a container, which modifies the container's start-up behavior by overruling the image's **ENTRYPOINT**. This is different from a lifecycle hook, which instead can be run once per deployment at a specified time.

## Procedure

1. Add the **command** parameters to the **spec** field of the **DeploymentConfig** object. You can also add an **args** field, which modifies the **command** (or the **ENTRYPOINT** if **command** does not exist).

```
spec:
  containers:
  -
    name: <container_name>
    image: 'image'
    command:
    - '<command>'
    args:
    - '<argument_1>'
    - '<argument_2>'
    - '<argument_3>'
```

For example, to execute the **java** command with the **-jar** and **/opt/app-root/springboots2idemo.jar** arguments:

```
spec:
  containers:
  -
    name: example-spring-boot
    image: 'image'
    command:
    - java
    args:
    - '-jar'
    - '/opt/app-root/springboots2idemo.jar'
```

### 4.2.1.6. Viewing deployment logs

#### Procedure

1. To stream the logs of the latest revision for a given **DeploymentConfig** object:

```
$ oc logs -f dc/<name>
```

If the latest revision is running or failed, the command returns the logs of the process that is responsible for deploying your pods. If it is successful, it returns the logs from a pod of your application.

2. You can also view logs from older failed deployment processes, if and only if these processes (old replication controllers and their deployer pods) exist and have not been pruned or deleted manually:

```
$ oc logs --version=1 dc/<name>
```

### 4.2.1.7. Deployment triggers

A **DeploymentConfig** object can contain triggers, which drive the creation of new deployment processes in response to events inside the cluster.



## WARNING

If no triggers are defined on a **DeploymentConfig** object, a config change trigger is added by default. If triggers are defined as an empty field, deployments must be started manually.

### Config change deployment triggers

The config change trigger results in a new replication controller whenever configuration changes are detected in the pod template of the **DeploymentConfig** object.



## NOTE

If a config change trigger is defined on a **DeploymentConfig** object, the first replication controller is automatically created soon after the **DeploymentConfig** object itself is created and it is not paused.

### Config change deployment trigger

```
triggers:
  - type: "ConfigChange"
```

### Image change deployment triggers

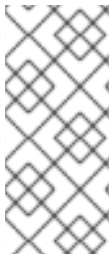
The image change trigger results in a new replication controller whenever the content of an image stream tag changes (when a new version of the image is pushed).

### Image change deployment trigger

```
triggers:
  - type: "ImageChange"
    imageChangeParams:
      automatic: true 1
      from:
        kind: "ImageStreamTag"
        name: "origin-ruby-sample:latest"
        namespace: "myproject"
      containerNames:
        - "helloworld"
```

**1** If the **imageChangeParams.automatic** field is set to **false**, the trigger is disabled.

With the above example, when the **latest** tag value of the **origin-ruby-sample** image stream changes and the new image value differs from the current image specified in the **DeploymentConfig** object's **helloworld** container, a new replication controller is created using the new image for the **helloworld** container.

**NOTE**

If an image change trigger is defined on a **DeploymentConfig** object (with a config change trigger and **automatic=false**, or with **automatic=true**) and the image stream tag pointed by the image change trigger does not exist yet, the initial deployment process will automatically start as soon as an image is imported or pushed by a build to the image stream tag.

**4.2.1.7.1. Setting deployment triggers****Procedure**

1. You can set deployment triggers for a **DeploymentConfig** object using the **oc set triggers** command. For example, to set a image change trigger, use the following command:

```
$ oc set triggers dc/<dc_name> \
  --from-image=<project>/<image>:<tag> -c <container_name>
```

**4.2.1.8. Setting deployment resources****NOTE**

This resource is available only if a cluster administrator has enabled the ephemeral storage technology preview. This feature is disabled by default.

A deployment is completed by a pod that consumes resources (memory, CPU, and ephemeral storage) on a node. By default, pods consume unbounded node resources. However, if a project specifies default container limits, then pods consume resources up to those limits.

You can also limit resource use by specifying resource limits as part of the deployment strategy. Deployment resources can be used with the recreate, rolling, or custom deployment strategies.

**Procedure**

1. In the following example, each of **resources**, **cpu**, **memory**, and **ephemeral-storage** is optional:

```
type: "Recreate"
resources:
  limits:
    cpu: "100m" 1
    memory: "256Mi" 2
    ephemeral-storage: "1Gi" 3
```

- 1 **cpu** is in CPU units: **100m** represents 0.1 CPU units ( $100 * 1e-3$ ).
- 2 **memory** is in bytes: **256Mi** represents 268435456 bytes ( $256 * 2^{20}$ ).
- 3 **ephemeral-storage** is in bytes: **1Gi** represents 1073741824 bytes ( $2^{30}$ ). This applies only if your cluster administrator enabled the ephemeral storage technology preview.

However, if a quota has been defined for your project, one of the following two items is required:



- A **resources** section set with an explicit **requests**:

```
type: "Recreate"
resources:
  requests: 1
    cpu: "100m"
    memory: "256Mi"
    ephemeral-storage: "1Gi"
```

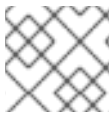
- 1 The **requests** object contains the list of resources that correspond to the list of resources in the quota.

- A limit range defined in your project, where the defaults from the **LimitRange** object apply to pods created during the deployment process.

To set deployment resources, choose one of the above options. Otherwise, deploy pod creation fails, citing a failure to satisfy quota.

#### 4.2.1.9. Scaling manually

In addition to rollbacks, you can exercise fine-grained control over the number of replicas by manually scaling them.



#### NOTE

Pods can also be auto-scaled using the **oc autoscale** command.

#### Procedure

1. To manually scale a **DeploymentConfig** object, use the **oc scale** command. For example, the following command sets the replicas in the **frontend DeploymentConfig** object to **3**.

```
$ oc scale dc frontend --replicas=3
```

The number of replicas eventually propagates to the desired and current state of the deployment configured by the **DeploymentConfig** object **frontend**.

#### 4.2.1.10. Accessing private repositories from **DeploymentConfig** objects

You can add a secret to your **DeploymentConfig** object so that it can access images from a private repository. This procedure shows the OpenShift Container Platform web console method.

#### Procedure

1. Create a new project.
2. From the **Workloads** page, create a secret that contains credentials for accessing a private image repository.
3. Create a **DeploymentConfig** object.
4. On the **DeploymentConfig** object editor page, set the **Pull Secret** and save your changes.

### 4.2.1.11. Assigning pods to specific nodes

You can use node selectors in conjunction with labeled nodes to control pod placement.

Cluster administrators can set the default node selector for a project in order to restrict pod placement to specific nodes. As a developer, you can set a node selector on a **Pod** configuration to restrict nodes even further.

#### Procedure

1. To add a node selector when creating a pod, edit the **Pod** configuration, and add the **nodeSelector** value. This can be added to a single **Pod** configuration, or in a **Pod** template:

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    disktype: ssd
  ...
```

Pods created when the node selector is in place are assigned to nodes with the specified labels. The labels specified here are used in conjunction with the labels added by a cluster administrator.

For example, if a project has the **type=user-node** and **region=east** labels added to a project by the cluster administrator, and you add the above **disktype: ssd** label to a pod, the pod is only ever scheduled on nodes that have all three labels.



#### NOTE

Labels can only be set to one value, so setting a node selector of **region=west** in a **Pod** configuration that has **region=east** as the administrator-set default, results in a pod that will never be scheduled.

### 4.2.1.12. Running a pod with a different service account

You can run a pod with a service account other than the default.

#### Procedure

1. Edit the **DeploymentConfig** object:

```
$ oc edit dc/<deployment_config>
```

2. Add the **serviceAccount** and **serviceAccountName** parameters to the **spec** field, and specify the service account you want to use:

```
spec:
  securityContext: {}
  serviceAccount: <service_account>
  serviceAccountName: <service_account>
```

## 4.3. USING DEPLOYMENT STRATEGIES

A *deployment strategy* is a way to change or upgrade an application. The aim is to make the change without downtime in a way that the user barely notices the improvements.

Because the end user usually accesses the application through a route handled by a router, the deployment strategy can focus on **DeploymentConfig** object features or routing features. Strategies that focus on the deployment impact all routes that use the application. Strategies that use router features target individual routes.

Many deployment strategies are supported through the **DeploymentConfig** object, and some additional strategies are supported through router features. Deployment strategies are discussed in this section.

### Choosing a deployment strategy

Consider the following when choosing a deployment strategy:

- Long-running connections must be handled gracefully.
- Database conversions can be complex and must be done and rolled back along with the application.
- If the application is a hybrid of microservices and traditional components, downtime might be required to complete the transition.
- You must have the infrastructure to do this.
- If you have a non-isolated test environment, you can break both new and old versions.

A deployment strategy uses readiness checks to determine if a new pod is ready for use. If a readiness check fails, the **DeploymentConfig** object retries to run the pod until it times out. The default timeout is **10m**, a value set in **TimeoutSeconds** in **dc.spec.strategy.\*params**.

#### 4.3.1. Rolling strategy

A rolling deployment slowly replaces instances of the previous version of an application with instances of the new version of the application. The rolling strategy is the default deployment strategy used if no strategy is specified on a **DeploymentConfig** object.

A rolling deployment typically waits for new pods to become **ready** via a readiness check before scaling down the old components. If a significant issue occurs, the rolling deployment can be aborted.

#### When to use a rolling deployment:

- When you want to take no downtime during an application update.
- When your application supports having old code and new code running at the same time.

A rolling deployment means you to have both old and new versions of your code running at the same time. This typically requires that your application handle N-1 compatibility.

#### Example rolling strategy definition

```
strategy:
  type: Rolling
  rollingParams:
    updatePeriodSeconds: 1 1
```

```

intervalSeconds: 1 2
timeoutSeconds: 120 3
maxSurge: "20%" 4
maxUnavailable: "10%" 5
pre: {} 6
post: {}

```

- 1 The time to wait between individual pod updates. If unspecified, this value defaults to **1**.
- 2 The time to wait between polling the deployment status after update. If unspecified, this value defaults to **1**.
- 3 The time to wait for a scaling event before giving up. Optional; the default is **600**. Here, *giving up* means automatically rolling back to the previous complete deployment.
- 4 **maxSurge** is optional and defaults to **25%** if not specified. See the information below the following procedure.
- 5 **maxUnavailable** is optional and defaults to **25%** if not specified. See the information below the following procedure.
- 6 **pre** and **post** are both lifecycle hooks.

The rolling strategy:

1. Executes any **pre** lifecycle hook.
2. Scales up the new replication controller based on the surge count.
3. Scales down the old replication controller based on the max unavailable count.
4. Repeats this scaling until the new replication controller has reached the desired replica count and the old replication controller has been scaled to zero.
5. Executes any **post** lifecycle hook.



### IMPORTANT

When scaling down, the rolling strategy waits for pods to become ready so it can decide whether further scaling would affect availability. If scaled up pods never become ready, the deployment process will eventually time out and result in a deployment failure.

The **maxUnavailable** parameter is the maximum number of pods that can be unavailable during the update. The **maxSurge** parameter is the maximum number of pods that can be scheduled above the original number of pods. Both parameters can be set to either a percentage (e.g., **10%**) or an absolute value (e.g., **2**). The default value for both is **25%**.

These parameters allow the deployment to be tuned for availability and speed. For example:

- **maxUnavailable\*=0** and **maxSurge\*=20%** ensures full capacity is maintained during the update and rapid scale up.
- **maxUnavailable\*=10%** and **maxSurge\*=0** performs an update using no extra capacity (an in-place update).

- **maxUnavailable\*=10%** and **maxSurge\*=10%** scales up and down quickly with some potential for capacity loss.

Generally, if you want fast rollouts, use **maxSurge**. If you have to take into account resource quota and can accept partial unavailability, use **maxUnavailable**.

#### 4.3.1.1. Canary deployments

All rolling deployments in OpenShift Container Platform are *canary deployments*; a new version (the canary) is tested before all of the old instances are replaced. If the readiness check never succeeds, the canary instance is removed and the **DeploymentConfig** object will be automatically rolled back.

The readiness check is part of the application code and can be as sophisticated as necessary to ensure the new instance is ready to be used. If you must implement more complex checks of the application (such as sending real user workloads to the new instance), consider implementing a custom deployment or using a blue-green deployment strategy.

#### 4.3.1.2. Creating a rolling deployment

Rolling deployments are the default type in OpenShift Container Platform. You can create a rolling deployment using the CLI.

##### Procedure

1. Create an application based on the example deployment images found in [Docker Hub](#):

```
$ oc new-app openshift/deployment-example
```

2. If you have the router installed, make the application available via a route or use the service IP directly.

```
$ oc expose svc/deployment-example
```

3. Browse to the application at **deployment-example.<project>.<router\_domain>** to verify you see the **v1** image.

4. Scale the **DeploymentConfig** object up to three replicas:

```
$ oc scale dc/deployment-example --replicas=3
```

5. Trigger a new deployment automatically by tagging a new version of the example as the **latest** tag:

```
$ oc tag deployment-example:v2 deployment-example:latest
```

6. In your browser, refresh the page until you see the **v2** image.

7. When using the CLI, the following command shows how many pods are on version 1 and how many are on version 2. In the web console, the pods are progressively added to v2 and removed from v1:

```
$ oc describe dc deployment-example
```

During the deployment process, the new replication controller is incrementally scaled up. After the new pods are marked as **ready** (by passing their readiness check), the deployment process continues.

If the pods do not become ready, the process aborts, and the deployment rolls back to its previous version.

### 4.3.1.3. Starting a rolling deployment using the Developer perspective

#### Prerequisites

- Ensure that you are in the **Developer** perspective of the web console.
- Ensure that you have created an application using the **Add** view and see it deployed in the **Topology** view.

#### Procedure

To start a rolling deployment to upgrade an application:

1. In the **Topology** view of the **Developer** perspective, click on the application node to see the **Overview** tab in the side panel. Note that the **Update Strategy** is set to the default **Rolling** strategy.
2. In the **Actions** drop-down menu, select **Start Rollout** to start a rolling update. The rolling deployment spins up the new version of the application and then terminates the old one.

Figure 4.1. Rolling update

The screenshot shows the OpenShift web console interface for the application 'nodejs-ex1'. The application is in the 'test-project' namespace. The update strategy is set to 'Rolling'. The pod selector is 'app=nodejs-ex1, deploymentconfig=nodejs-ex1'. The interface shows a visual indicator of the rolling update process, with a circle containing '0 pods' on the left and a circle containing '1 pod' on the right, connected by an arrow. The 'Overview' tab is selected, and the 'Actions' menu is visible in the top right corner.

Name	Latest Version
nodejs-ex1	2

Namespace	Message
test-project	manual change

Labels	Update Strategy
app=nodejs-ex1	Rolling
app.kubernetes.io/com... =nodejs...	
app.kubernetes.io/inst... =nodejs-...	
app.kubernetes.io/name=nodejs	
app.openshift.io/runtime=nodejs	
app.openshift.io/runtime-... =10-S...	

Min Ready Seconds	Triggers
Not Configured	ImageChange, ConfigChange

**Pod Selector**  
 app=nodejs-ex1,  
 deploymentconfig=nodejs-ex1

## Additional resources

- [Creating and deploying applications on OpenShift Container Platform using the Developer perspective](#)
- [Viewing the applications in your project, verifying their deployment status, and interacting with them in the Topology view](#)

### 4.3.2. Recreate strategy

The recreate strategy has basic rollout behavior and supports lifecycle hooks for injecting code into the deployment process.

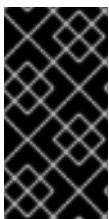
#### Example recreate strategy definition

```
strategy:
  type: Recreate
  recreateParams: 1
  pre: {} 2
  mid: {}
  post: {}
```

- 1 **recreateParams** are optional.
- 2 **pre**, **mid**, and **post** are lifecycle hooks.

The recreate strategy:

1. Executes any **pre** lifecycle hook.
2. Scales down the previous deployment to zero.
3. Executes any **mid** lifecycle hook.
4. Scales up the new deployment.
5. Executes any **post** lifecycle hook.



#### IMPORTANT

During scale up, if the replica count of the deployment is greater than one, the first replica of the deployment will be validated for readiness before fully scaling up the deployment. If the validation of the first replica fails, the deployment will be considered a failure.

#### When to use a recreate deployment:

- When you must run migrations or other data transformations before your new code starts.
- When you do not support having new and old versions of your application code running at the same time.
- When you want to use a RWO volume, which is not supported being shared between multiple replicas.

A recreate deployment incurs downtime because, for a brief period, no instances of your application are running. However, your old code and new code do not run at the same time.

### 4.3.3. Starting a recreate deployment using the Developer perspective

You can switch the deployment strategy from the default rolling update to a recreate update using the **Developer** perspective in the web console.

#### Prerequisites

- Ensure that you are in the **Developer** perspective of the web console.
- Ensure that you have created an application using the **Add** view and see it deployed in the **Topology** view.

#### Procedure

To switch to a recreate update strategy and to upgrade an application:

1. In the **Actions** drop-down menu, select **Edit Deployment Config** to see the deployment configuration details of the application.
2. In the YAML editor, change the **spec.strategy.type** to **Recreate** and click **Save**.
3. In the **Topology** view, select the node to see the **Overview** tab in the side panel. The **Update Strategy** is now set to **Recreate**.
4. Use the **Actions** drop-down menu to select **Start Rollout** to start an update using the recreate strategy. The recreate strategy first terminates pods for the older version of the application and then spins up pods for the new version.



Figure 4.2. Recreate update

The screenshot shows the OpenShift console interface for a deployment named 'nodejs-ex1'. The deployment is currently in the 'test-project' namespace and has 0 pods. The update strategy is set to 'Recreate', which was triggered by a 'manual change'. The latest version of the deployment is 3. The pod selector is 'app=nodejs-ex1, deploymentconfig=nodejs-ex1'. The console also shows various labels and triggers associated with the deployment.

Property	Value
Name	nodejs-ex1
Namespace	test-project
Labels	app=nodejs-ex1, app.kubernetes.io/com... =nodejs..., app.kubernetes.io/inst... =nodejs-..., app.kubernetes.io/name=nodejs, app.openshift.io/runtime=nodejs, app.openshift.io/runtime-... =10-S...
Pod Selector	app=nodejs-ex1, deploymentconfig=nodejs-ex1
Latest Version	3
Message	manual change
Update Strategy	Recreate
Min Ready Seconds	Not Configured
Triggers	ImageChange, ConfigChange

#### Additional resources

- [Creating and deploying applications on OpenShift Container Platform using the Developer perspective](#)
- [Viewing the applications in your project, verifying their deployment status, and interacting with them in the Topology view](#)

#### 4.3.4. Custom strategy

The custom strategy allows you to provide your own deployment behavior.

#### Example custom strategy definition

```
strategy:
  type: Custom
  customParams:
    image: organization/strategy
    command: [ "command", "arg1" ]
```

```
environment:
  - name: ENV_1
    value: VALUE_1
```

In the above example, the **organization/strategy** container image provides the deployment behavior. The optional **command** array overrides any **CMD** directive specified in the image's **Dockerfile**. The optional environment variables provided are added to the execution environment of the strategy process.

Additionally, OpenShift Container Platform provides the following environment variables to the deployment process:

Environment variable	Description
<b>OPENSIFT_DEPLOYMENT_NAME</b>	The name of the new deployment, a replication controller.
<b>OPENSIFT_DEPLOYMENT_NAMESPACE</b>	The name space of the new deployment.

The replica count of the new deployment will initially be zero. The responsibility of the strategy is to make the new deployment active using the logic that best serves the needs of the user.

Alternatively, use the **customParams** object to inject the custom deployment logic into the existing deployment strategies. Provide a custom shell script logic and call the **openshift-deploy** binary. Users do not have to supply their custom deployer container image; in this case, the default OpenShift Container Platform deployer image is used instead:

```
strategy:
  type: Rolling
  customParams:
    command:
      - /bin/sh
      - -c
      - |
        set -e
        openshift-deploy --until=50%
        echo Halfway there
        openshift-deploy
        echo Complete
```

This results in following deployment:

```
Started deployment #2
--> Scaling up custom-deployment-2 from 0 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
  Scaling custom-deployment-2 up to 1
--> Reached 50% (currently 50%)
Halfway there
--> Scaling up custom-deployment-2 from 1 to 2, scaling down custom-deployment-1 from 2 to 0
(keep 2 pods available, don't exceed 3 pods)
  Scaling custom-deployment-1 down to 1
  Scaling custom-deployment-2 up to 2
```

```
Scaling custom-deployment-1 down to 0
--> Success
Complete
```

If the custom deployment strategy process requires access to the OpenShift Container Platform API or the Kubernetes API the container that executes the strategy can use the service account token available inside the container for authentication.

### 4.3.5. Lifecycle hooks

The rolling and recreate strategies support *lifecycle hooks*, or deployment hooks, which allow behavior to be injected into the deployment process at predefined points within the strategy:

#### Example pre lifecycle hook

```
pre:
  failurePolicy: Abort
  execNewPod: {} 1
```

**1** **execNewPod** is a pod-based lifecycle hook.

Every hook has a *failure policy*, which defines the action the strategy should take when a hook failure is encountered:

<b>Abort</b>	The deployment process will be considered a failure if the hook fails.
<b>Retry</b>	The hook execution should be retried until it succeeds.
<b>Ignore</b>	Any hook failure should be ignored and the deployment should proceed.

Hooks have a type-specific field that describes how to execute the hook. Currently, pod-based hooks are the only supported hook type, specified by the **execNewPod** field.

#### Pod-based lifecycle hook

Pod-based lifecycle hooks execute hook code in a new pod derived from the template in a **DeploymentConfig** object.

The following simplified example deployment uses the rolling strategy. Triggers and some other minor details are omitted for brevity:

```
kind: DeploymentConfig
apiVersion: v1
metadata:
  name: frontend
spec:
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
        - name: helloworld
```

```

    image: openshift/origin-ruby-sample
  replicas: 5
  selector:
    name: frontend
  strategy:
    type: Rolling
    rollingParams:
      pre:
        failurePolicy: Abort
        execNewPod:
          containerName: helloworld 1
          command: [ "/usr/bin/command", "arg1", "arg2" ] 2
          env: 3
            - name: CUSTOM_VAR1
              value: custom_value1
          volumes:
            - data 4

```

- 1** The **helloworld** name refers to `spec.template.spec.containers[0].name`.
- 2** This **command** overrides any **ENTRYPOINT** defined by the **openshift/origin-ruby-sample** image.
- 3** **env** is an optional set of environment variables for the hook container.
- 4** **volumes** is an optional set of volume references for the hook container.

In this example, the **pre** hook will be executed in a new pod using the **openshift/origin-ruby-sample** image from the **helloworld** container. The hook pod has the following properties:

- The hook command is **/usr/bin/command arg1 arg2**.
- The hook container has the **CUSTOM\_VAR1=custom\_value1** environment variable.
- The hook failure policy is **Abort**, meaning the deployment process fails if the hook fails.
- The hook pod inherits the **data** volume from the **DeploymentConfig** object pod.

#### 4.3.5.1. Setting lifecycle hooks

You can set lifecycle hooks, or deployment hooks, for a deployment using the CLI.

##### Procedure

1. Use the **oc set deployment-hook** command to set the type of hook you want: **--pre**, **--mid**, or **--post**. For example, to set a pre-deployment hook:

```

$ oc set deployment-hook dc/frontend \
  --pre -c helloworld -e CUSTOM_VAR1=custom_value1 \
  -v data --failure-policy=abort -- /usr/bin/command arg1 arg2

```

## 4.4. USING ROUTE-BASED DEPLOYMENT STRATEGIES

Deployment strategies provide a way for the application to evolve. Some strategies use **DeploymentConfig** objects to make changes that are seen by users of all routes that resolve to the

application. Other advanced strategies, such as the ones described in this section, use router features in conjunction with `DeploymentConfig`` objects to impact specific routes.

The most common route-based strategy is to use a *blue-green deployment*. The new version (the blue version) is brought up for testing and evaluation, while the users still use the stable version (the green version). When ready, the users are switched to the blue version. If a problem arises, you can switch back to the green version.

A common alternative strategy is to use *A/B versions* that are both active at the same time and some users use one version, and some users use the other version. This can be used for experimenting with user interface changes and other features to get user feedback. It can also be used to verify proper operation in a production context where problems impact a limited number of users.

A canary deployment tests the new version but when a problem is detected it quickly falls back to the previous version. This can be done with both of the above strategies.

The route-based deployment strategies do not scale the number of pods in the services. To maintain desired performance characteristics the deployment configurations might have to be scaled.

#### 4.4.1. Proxy shards and traffic splitting

In production environments, you can precisely control the distribution of traffic that lands on a particular shard. When dealing with large numbers of instances, you can use the relative scale of individual shards to implement percentage based traffic. That combines well with a *proxy shard*, which forwards or splits the traffic it receives to a separate service or application running elsewhere.

In the simplest configuration, the proxy forwards requests unchanged. In more complex setups, you can duplicate the incoming requests and send to both a separate cluster as well as to a local instance of the application, and compare the result. Other patterns include keeping the caches of a DR installation warm, or sampling incoming traffic for analysis purposes.

Any TCP (or UDP) proxy could be run under the desired shard. Use the **oc scale** command to alter the relative number of instances serving requests under the proxy shard. For more complex traffic management, consider customizing the OpenShift Container Platform router with proportional balancing capabilities.

#### 4.4.2. N-1 compatibility

Applications that have new code and old code running at the same time must be careful to ensure that data written by the new code can be read and handled (or gracefully ignored) by the old version of the code. This is sometimes called *schema evolution* and is a complex problem.

This can take many forms: data stored on disk, in a database, in a temporary cache, or that is part of a user's browser session. While most web applications can support rolling deployments, it is important to test and design your application to handle it.

For some applications, the period of time that old code and new code is running side by side is short, so bugs or some failed user transactions are acceptable. For others, the failure pattern may result in the entire application becoming non-functional.

One way to validate N-1 compatibility is to use an A/B deployment: run the old code and new code at the same time in a controlled way in a test environment, and verify that traffic that flows to the new deployment does not cause failures in the old deployment.

#### 4.4.3. Graceful termination

OpenShift Container Platform and Kubernetes give application instances time to shut down before removing them from load balancing rotations. However, applications must ensure they cleanly terminate user connections as well before they exit.

On shutdown, OpenShift Container Platform sends a **TERM** signal to the processes in the container. Application code, on receiving **SIGTERM**, stop accepting new connections. This ensures that load balancers route traffic to other active instances. The application code then waits until all open connections are closed, or gracefully terminate individual connections at the next opportunity, before exiting.

After the graceful termination period expires, a process that has not exited is sent the **KILL** signal, which immediately ends the process. The **terminationGracePeriodSeconds** attribute of a pod or pod template controls the graceful termination period (default 30 seconds) and can be customized per application as necessary.

#### 4.4.4. Blue-green deployments

Blue-green deployments involve running two versions of an application at the same time and moving traffic from the in-production version (the green version) to the newer version (the blue version). You can use a rolling strategy or switch services in a route.

Because many applications depend on persistent data, you must have an application that supports *N-1 compatibility*, which means it shares data and implements live migration between the database, store, or disk by creating two copies of the data layer.

Consider the data used in testing the new version. If it is the production data, a bug in the new version can break the production version.

##### 4.4.4.1. Setting up a blue-green deployment

Blue-green deployments use two **DeploymentConfig** objects. Both are running, and the one in production depends on the service the route specifies, with each **DeploymentConfig** object exposed to a different service.



#### NOTE

Routes are intended for web (HTTP and HTTPS) traffic, so this technique is best suited for web applications.

You can create a new route to the new version and test it. When ready, change the service in the production route to point to the new service and the new (blue) version is live.

If necessary, you can roll back to the older (green) version by switching the service back to the previous version.

#### Procedure

1. Create two copies of the example application:

```
$ oc new-app openshift/deployment-example:v1 --name=example-green
$ oc new-app openshift/deployment-example:v2 --name=example-blue
```

This creates two independent application components: one running the **v1** image under the **example-green** service, and one using the **v2** image under the **example-blue** service.

2. Create a route that points to the old service:

```
$ oc expose svc/example-green --name=bluegreen-example
```

3. Browse to the application at **example-green.<project>.<router\_domain>** to verify you see the **v1** image.
4. Edit the route and change the service name to **example-blue**:

```
$ oc patch route/bluegreen-example -p '{"spec":{"to":{"name":"example-blue"}}}'
```

5. To verify that the route has changed, refresh the browser until you see the **v2** image.

### 4.4.5. A/B deployments

The A/B deployment strategy lets you try a new version of the application in a limited way in the production environment. You can specify that the production version gets most of the user requests while a limited fraction of requests go to the new version.

Because you control the portion of requests to each version, as testing progresses you can increase the fraction of requests to the new version and ultimately stop using the previous version. As you adjust the request load on each version, the number of pods in each service might have to be scaled as well to provide the expected performance.

In addition to upgrading software, you can use this feature to experiment with versions of the user interface. Since some users get the old version and some the new, you can evaluate the user's reaction to the different versions to inform design decisions.

For this to be effective, both the old and new versions must be similar enough that both can run at the same time. This is common with bug fix releases and when new features do not interfere with the old. The versions require N-1 compatibility to properly work together.

OpenShift Container Platform supports N-1 compatibility through the web console as well as the CLI.

#### 4.4.5.1. Load balancing for A/B testing

The user sets up a route with multiple services. Each service handles a version of the application.

Each service is assigned a **weight** and the portion of requests to each service is the **service\_weight** divided by the **sum\_of\_weights**. The **weight** for each service is distributed to the service's endpoints so that the sum of the endpoint **weights** is the service **weight**.

The route can have up to four services. The **weight** for the service can be between **0** and **256**. When the **weight** is **0**, the service does not participate in load-balancing but continues to serve existing persistent connections. When the service **weight** is not **0**, each endpoint has a minimum **weight** of **1**. Because of this, a service with a lot of endpoints can end up with higher **weight** than desired. In this case, reduce the number of pods to get the desired load balance **weight**.

#### Procedure

To set up the A/B environment:

1. Create the two applications and give them different names. Each creates a **DeploymentConfig** object. The applications are versions of the same program; one is usually the current production version and the other the proposed new version:

■

```
$ oc new-app openshift/deployment-example --name=ab-example-a
$ oc new-app openshift/deployment-example --name=ab-example-b
```

Both applications are deployed and services are created.

2. Make the application available externally via a route. At this point, you can expose either. It can be convenient to expose the current production version first and later modify the route to add the new version.

```
$ oc expose svc/ab-example-a
```

Browse to the application at **ab-example-`<project>`.`<router_domain>`** to verify that you see the desired version.

3. When you deploy the route, the router balances the traffic according to the **weights** specified for the services. At this point, there is a single service with default **weight=1** so all requests go to it. Adding the other service as an **alternateBackends** and adjusting the **weights** brings the A/B setup to life. This can be done by the **oc set route-backends** command or by editing the route. Setting the **oc set route-backend** to **0** means the service does not participate in load-balancing, but continues to serve existing persistent connections.



#### NOTE

Changes to the route just change the portion of traffic to the various services. You might have to scale the deployment to adjust the number of pods to handle the anticipated loads.

To edit the route, run:


```
$ oc edit route <route_name>
...
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin
spec:
  host: ab-example.my-project.my-domain
  to:
    kind: Service
    name: ab-example-a
    weight: 10
  alternateBackends:
  - kind: Service
    name: ab-example-b
    weight: 15
...
```

#### 4.4.5.1.1. Managing weights of an existing route using the web console

##### Procedure

1. Navigate to the **Networking** → **Routes** page.



2. Click the Actions menu  next to the route you want to edit and select **Edit Route**.
3. Edit the YAML file. Update the **weight** to be an integer between **0** and **256** that specifies the relative weight of the target against other target reference objects. The value **0** suppresses requests to this back end. The default is **100**. Run **oc explain routes.spec.alternateBackends** for more information about the options.
4. Click **Save**.

#### 4.4.5.1.2. Managing weights of a new route using the web console

1. Navigate to the **Networking → Routes** page.
2. Click **Create Route**.
3. Enter the route **Name**.
4. Select the **Service**.
5. Click **Add Alternate Service**.
6. Enter a value for **Weight** and **Alternate Service Weight**. Enter a number between **0** and **255** that depicts relative weight compared with other targets. The default is **100**.
7. Select the **Target Port**.
8. Click **Create**.

#### 4.4.5.1.3. Managing weights using the CLI

##### Procedure

1. To manage the services and corresponding weights load balanced by the route, use the **oc set route-backends** command:

```
$ oc set route-backends ROUTENAME \
  [--zero|--equal] [--adjust] SERVICE=WEIGHT[%] [...] [options]
```

For example, the following sets **ab-example-a** as the primary service with **weight=198** and **ab-example-b** as the first alternate service with a **weight=2**:

```
$ oc set route-backends ab-example ab-example-a=198 ab-example-b=2
```

This means 99% of traffic is sent to service **ab-example-a** and 1% to service **ab-example-b**.

This command does not scale the deployment. You might be required to do so to have enough pods to handle the request load.

2. Run the command with no flags to verify the current configuration:

```
$ oc set route-backends ab-example
NAME           KIND  TO      WEIGHT
routes/ab-example  Service  ab-example-a 198 (99%)
routes/ab-example  Service  ab-example-b  2  (1%)
```

- 3. To alter the weight of an individual service relative to itself or to the primary service, use the **--adjust** flag. Specifying a percentage adjusts the service relative to either the primary or the first alternate (if you specify the primary). If there are other backends, their weights are kept proportional to the changed.

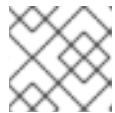
For example:

```
$ oc set route-backends ab-example --adjust ab-example-a=200 ab-example-b=10
$ oc set route-backends ab-example --adjust ab-example-b=5%
$ oc set route-backends ab-example --adjust ab-example-b=+15%
```

The **--equal** flag sets the **weight** of all services to **100**:

```
$ oc set route-backends ab-example --equal
```

The **--zero** flag sets the **weight** of all services to **0**. All requests then return with a 503 error.



#### NOTE

Not all routers may support multiple or weighted backends.

#### 4.4.5.1.4. One service, multiple DeploymentConfig objects

##### Procedure

1. Create a new application, adding a label **ab-example=true** that will be common to all shards:

```
$ oc new-app openshift/deployment-example --name=ab-example-a
```

The application is deployed and a service is created. This is the first shard.

2. Make the application available via a route, or use the service IP directly:

```
$ oc expose svc/ab-example-a --name=ab-example
```

3. Browse to the application at **ab-example-<project>.<router\_domain>** to verify you see the **v1** image.
4. Create a second shard based on the same source image and label as the first shard, but with a different tagged version and unique environment variables:

```
$ oc new-app openshift/deployment-example:v2 \
  --name=ab-example-b --labels=ab-example=true \
  SUBTITLE="shard B" COLOR="red"
```

5. At this point, both sets of pods are being served under the route. However, because both browsers (by leaving a connection open) and the router (by default, through a cookie) attempt to preserve your connection to a back-end server, you might not see both shards being returned to you.

To force your browser to one or the other shard:

- a. Use the **oc scale** command to reduce replicas of **ab-example-a** to **0**.

```
$ oc scale dc/ab-example-a --replicas=0
```

Refresh your browser to show **v2** and **shard B** (in red).

- b. Scale **ab-example-a** to **1** replica and **ab-example-b** to **0**:

```
$ oc scale dc/ab-example-a --replicas=1; oc scale dc/ab-example-b --replicas=0
```

Refresh your browser to show **v1** and **shard A** (in blue).

6. If you trigger a deployment on either shard, only the pods in that shard are affected. You can trigger a deployment by changing the **SUBTITLE** environment variable in either **DeploymentConfig** object:

```
$ oc edit dc/ab-example-a
```

or

```
$ oc edit dc/ab-example-b
```

## CHAPTER 5. QUOTAS

### 5.1. RESOURCE QUOTAS PER PROJECT

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

This guide describes how resource quotas work, how cluster administrators can set and manage resource quotas on a per project basis, and how developers and cluster administrators can view them.

#### 5.1.1. Resources managed by quotas

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



#### NOTE

A pod is in a terminal state if **status.phase in (Failed, Succeeded)** is true.

Table 5.1. Compute resources managed by quota

Resource Name	Description
<b>cpu</b>	The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. <b>cpu</b> and <b>requests.cpu</b> are the same value and can be used interchangeably.
<b>memory</b>	The sum of memory requests across all pods in a non-terminal state cannot exceed this value. <b>memory</b> and <b>requests.memory</b> are the same value and can be used interchangeably.
<b>ephemeral-storage</b>	The sum of local ephemeral storage requests across all pods in a non-terminal state cannot exceed this value. <b>ephemeral-storage</b> and <b>requests.ephemeral-storage</b> are the same value and can be used interchangeably. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default.
<b>requests.cpu</b>	The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. <b>cpu</b> and <b>requests.cpu</b> are the same value and can be used interchangeably.
<b>requests.memory</b>	The sum of memory requests across all pods in a non-terminal state cannot exceed this value. <b>memory</b> and <b>requests.memory</b> are the same value and can be used interchangeably.

Resource Name	Description
<b>requests.ephemeral-storage</b>	The sum of ephemeral storage requests across all pods in a non-terminal state cannot exceed this value. <b>ephemeral-storage</b> and <b>requests.ephemeral-storage</b> are the same value and can be used interchangeably. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default.
<b>limits.cpu</b>	The sum of CPU limits across all pods in a non-terminal state cannot exceed this value.
<b>limits.memory</b>	The sum of memory limits across all pods in a non-terminal state cannot exceed this value.
<b>limits.ephemeral-storage</b>	The sum of ephemeral storage limits across all pods in a non-terminal state cannot exceed this value. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default.

Table 5.2. Storage resources managed by quota

Resource Name	Description
<b>requests.storage</b>	The sum of storage requests across all persistent volume claims in any state cannot exceed this value.
<b>persistentvolumeclaims</b>	The total number of persistent volume claims that can exist in the project.
<b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/requests.storage</b>	The sum of storage requests across all persistent volume claims in any state that have a matching storage class, cannot exceed this value.
<b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/persistentvolumeclaims</b>	The total number of persistent volume claims with a matching storage class that can exist in the project.

Table 5.3. Object counts managed by quota

Resource Name	Description
<b>pods</b>	The total number of pods in a non-terminal state that can exist in the project.
<b>replicationcontrollers</b>	The total number of ReplicationControllers that can exist in the project.
<b>resourcequotas</b>	The total number of resource quotas that can exist in the project.

Resource Name	Description
<b>services</b>	The total number of services that can exist in the project.
<b>services.loadbalancers</b>	The total number of services of type <b>LoadBalancer</b> that can exist in the project.
<b>services.nodeports</b>	The total number of services of type <b>NodePort</b> that can exist in the project.
<b>secrets</b>	The total number of secrets that can exist in the project.
<b>configmaps</b>	The total number of <b>ConfigMap</b> objects that can exist in the project.
<b>persistentvolumeclaims</b>	The total number of persistent volume claims that can exist in the project.
<b>openshift.io/imagestreams</b>	The total number of imagestreams that can exist in the project.

### 5.1.2. Quota scopes

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

Adding a scope to a quota restricts the set of resources to which that quota can apply. Specifying a resource outside of the allowed set results in a validation error.

Scope	Description
<b>Terminating</b>	Match pods where <b>spec.activeDeadlineSeconds</b> <b>&gt;= 0</b> .
<b>NotTerminating</b>	Match pods where <b>spec.activeDeadlineSeconds</b> is <b>nil</b> .
<b>BestEffort</b>	Match pods that have best effort quality of service for either <b>cpu</b> or <b>memory</b> .
<b>NotBestEffort</b>	Match pods that do not have best effort quality of service for <b>cpu</b> and <b>memory</b> .

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

- **pods**

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

- **pods**

- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**
- **limits.cpu**
- **ephemeral-storage**
- **requests.ephemeral-storage**
- **limits.ephemeral-storage**



#### NOTE

Ephemeral storage requests and limits apply only if you enabled the ephemeral storage technology preview. This feature is disabled by default.

### 5.1.3. Quota enforcement

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

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

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

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

### 5.1.4. Requests versus limits

When allocating compute resources, each container might specify a request and a limit value each for CPU, memory, and ephemeral storage. Quotas can restrict any of these values.

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

### 5.1.5. Sample resource quota definitions

#### **core-object-counts.yaml**

```
apiVersion: v1
```

```

kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" ❶
    persistentvolumeclaims: "4" ❷
    replicationcontrollers: "20" ❸
    secrets: "10" ❹
    services: "10" ❺
    services.loadbalancers: "2" ❻

```

- ❶ The total number of **ConfigMap** objects that can exist in the project.
- ❷ The total number of persistent volume claims (PVCs) that can exist in the project.
- ❸ The total number of replication controllers that can exist in the project.
- ❹ The total number of secrets that can exist in the project.
- ❺ The total number of services that can exist in the project.
- ❻ The total number of services of type **LoadBalancer** that can exist in the project.

#### openshift-object-counts.yaml

```

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

```

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

#### compute-resources.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4" ❶
    requests.cpu: "1" ❷
    requests.memory: 1Gi ❸
    requests.ephemeral-storage: 2Gi ❹
    limits.cpu: "2" ❺
    limits.memory: 2Gi ❻
    limits.ephemeral-storage: 4Gi ❼

```



- 1 The total number of pods in a non-terminal state that can exist in the project.
- 2 Across all pods in a non-terminal state, the sum of CPU requests cannot exceed 1 core.
- 3 Across all pods in a non-terminal state, the sum of memory requests cannot exceed 1Gi.
- 4 Across all pods in a non-terminal state, the sum of ephemeral storage requests cannot exceed 2Gi.
- 5 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed 2 cores.
- 6 Across all pods in a non-terminal state, the sum of memory limits cannot exceed 2Gi.
- 7 Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed 4Gi.

### besteffort.yaml

```

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

```

- 1 The total number of pods in a non-terminal state with **BestEffort** quality of service that can exist in the project.
- 2 Restricts the quota to only matching pods that have **BestEffort** quality of service for either memory or CPU.

### compute-resources-long-running.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" 1
    limits.cpu: "4" 2
    limits.memory: "2Gi" 3
    limits.ephemeral-storage: "4Gi" 4
  scopes:
    - NotTerminating 5

```

- 1 The total number of pods in a non-terminal state.
- 2 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- 3 Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.

- 4 Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed this value.
- 5 Restricts the quota to only matching pods where **spec.activeDeadlineSeconds** is set to **nil**. Build pods will fall under **NotTerminating** unless the **RestartNever** policy is applied.

### compute-resources-time-bound.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" 1
    limits.cpu: "1" 2
    limits.memory: "1Gi" 3
    limits.ephemeral-storage: "1Gi" 4
  scopes:
    - Terminating 5

```

- 1 The total number of pods in a non-terminal state.
- 2 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- 3 Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
- 4 Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed this value.
- 5 Restricts the quota to only matching pods where **spec.activeDeadlineSeconds**  $\geq 0$ . For example, this quota would charge for build or deployer pods, but not long running pods like a web server or database.

### storage-consumption.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" 1
    requests.storage: "50Gi" 2
    gold.storageclass.storage.k8s.io/requests.storage: "10Gi" 3
    silver.storageclass.storage.k8s.io/requests.storage: "20Gi" 4
    silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" 5
    bronze.storageclass.storage.k8s.io/requests.storage: "0" 6
    bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" 7

```

- 1 The total number of persistent volume claims in a project

- 2 Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.
- 3 Across all persistent volume claims in a project, the sum of storage requested in the gold storage class cannot exceed this value.
- 4 Across all persistent volume claims in a project, the sum of storage requested in the silver storage class cannot exceed this value.
- 5 Across all persistent volume claims in a project, the total number of claims in the silver storage class cannot exceed this value.
- 6 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot request storage.
- 7 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot create claims.

### 5.1.6. Creating a quota

You can create a quota to constrain resource usage in a given project.

#### Procedure

1. Define the quota in a file.
2. Use the file to create the quota and apply it to a project:

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

For example:

```
$ oc create -f core-object-counts.yaml -n demoproject
```

#### 5.1.6.1. Creating object count quotas

You can create an object count quota for all OpenShift Container Platform standard namespaced resource types, such as **BuildConfig**, and **DeploymentConfig**. An object quota count places a defined quota on all standard namespaced resource types.

When using a resource quota, an object is charged against the quota if it exists in server storage. These types of quotas are useful to protect against exhaustion of storage resources.

#### Procedure

To configure an object count quota for a resource:

1. Run the following command:

```
$ oc create quota <name> \
  --hard=count/<resource>.<group>=<quota>,count/<resource>.<group>=<quota> 1
```

- 1 **<resource>** is the name of the resource, and **<group>** is the API group, if applicable. Use the **oc api-resources** command for a list of resources and their associated API groups.

For example:

```
$ oc create quota test \
--
hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secrets=4
resourcequota "test" created
```

This example limits the listed resources to the hard limit in each project in the cluster.

2. Verify that the quota was created:

```
$ oc describe quota test
Name:          test
Namespace:     quota
Resource       Used Hard
-----
count/deployments.extensions 0 2
count/pods                0 3
count/replicasets.extensions 0 4
count/secrets              0 4
```

### 5.1.6.2. Setting resource quota for extended resources

Overcommitment of resources is not allowed for extended resources, so you must specify **requests** and **limits** for the same extended resource in a quota. Currently, only quota items with the prefix **requests.** is allowed for extended resources. The following is an example scenario of how to set resource quota for the GPU resource **nvidia.com/gpu**.

#### Procedure

1. Determine how many GPUs are available on a node in your cluster. For example:

```
# oc describe node ip-172-31-27-209.us-west-2.compute.internal | egrep
'Capacity|Allocatable|gpu'
      openshift.com/gpu-accelerator=true
Capacity:
  nvidia.com/gpu: 2
Allocatable:
  nvidia.com/gpu: 2
  nvidia.com/gpu 0      0
```

In this example, 2 GPUs are available.

2. Set a quota in the namespace **nvidia**. In this example, the quota is **1**:

```
# cat gpu-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: gpu-quota
```

```

namespace: nvidia
spec:
  hard:
    requests.nvidia.com/gpu: 1

```

3. Create the quota:

```

# oc create -f gpu-quota.yaml
resourcequota/gpu-quota created

```

4. Verify that the namespace has the correct quota set:

```

# oc describe quota gpu-quota -n nvidia
Name:          gpu-quota
Namespace:     nvidia
Resource       Used  Hard
-----
requests.nvidia.com/gpu 0   1

```

5. Run a pod that asks for a single GPU:

```

# oc create -f gpu-pod.yaml

apiVersion: v1
kind: Pod
metadata:
  generateName: gpu-pod-
  namespace: nvidia
spec:
  restartPolicy: OnFailure
  containers:
  - name: rhel7-gpu-pod
    image: rhel7
    env:
    - name: NVIDIA_VISIBLE_DEVICES
      value: all
    - name: NVIDIA_DRIVER_CAPABILITIES
      value: "compute,utility"
    - name: NVIDIA_REQUIRE_CUDA
      value: "cuda>=5.0"
    command: ["sleep"]
    args: ["infinity"]
  resources:
    limits:
      nvidia.com/gpu: 1

```

6. Verify that the pod is running:

```

# oc get pods
NAME          READY  STATUS   RESTARTS  AGE
gpu-pod-s46h7 1/1    Running  0          1m

```

7. Verify that the quota **Used** counter is correct:

```
# oc describe quota gpu-quota -n nvidia
Name:          gpu-quota
Namespace:     nvidia
Resource       Used Hard
-----
requests.nvidia.com/gpu 1 1
```

- Attempt to create a second GPU pod in the **nvidia** namespace. This is technically available on the node because it has 2 GPUs:

```
# oc create -f gpu-pod.yaml
Error from server (Forbidden): error when creating "gpu-pod.yaml": pods "gpu-pod-f7z2w" is forbidden: exceeded quota: gpu-quota, requested: requests.nvidia.com/gpu=1, used: requests.nvidia.com/gpu=1, limited: requests.nvidia.com/gpu=1
```

This **Forbidden** error message is expected because you have a quota of 1 GPU and this pod tried to allocate a second GPU, which exceeds its quota.

### 5.1.7. Viewing a quota

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

You can also use the CLI to view quota details.

#### Procedure

- Get the list of quotas defined in the project. For example, for a project called **demoproject**:

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

- Describe the quota you are interested in, for example the **core-object-counts** quota:

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

### 5.1.8. Configuring explicit resource quotas

Configure explicit resource quotas in a project request template to apply specific resource quotas in new projects.

## Prerequisites

- Access to the cluster as a user with the cluster-admin role.
- Install the OpenShift CLI (**oc**).

## Procedure

1. Add a resource quota definition to a project request template:

- If a project request template does not exist in a cluster:
  - a. Create a bootstrap project template and output it to a file called **template.yaml**:

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

- b. Add a resource quota definition to **template.yaml**. The following example defines a resource quota named 'storage-consumption'. The definition must be added before the **parameters:** section in the template:

```
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: storage-consumption
  spec:
    hard:
      persistentvolumeclaims: "10" 1
      requests.storage: "50Gi" 2
      gold.storageclass.storage.k8s.io/requests.storage: "10Gi" 3
      silver.storageclass.storage.k8s.io/requests.storage: "20Gi" 4
      silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" 5
      bronze.storageclass.storage.k8s.io/requests.storage: "0" 6
      bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" 7
```

- 1 The total number of persistent volume claims in a project.
- 2 Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.
- 3 Across all persistent volume claims in a project, the sum of storage requested in the gold storage class cannot exceed this value.
- 4 Across all persistent volume claims in a project, the sum of storage requested in the silver storage class cannot exceed this value.
- 5 Across all persistent volume claims in a project, the total number of claims in the silver storage class cannot exceed this value.
- 6 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this value is set to **0**, the bronze storage class cannot request storage.
- 7 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this value is set to **0**, the bronze storage class cannot create claims.

- c. Create a project request template from the modified **template.yaml** file in the **openshift-config** namespace:

```
$ oc create -f template.yaml -n openshift-config
```



#### NOTE

To include the configuration as a **kubectrl.kubernetes.io/last-applied-configuration** annotation, add the **--save-config** option to the **oc create** command.

By default, the template is called **project-request**.

- If a project request template already exists within a cluster:



#### NOTE

If you declaratively or imperatively manage objects within your cluster by using configuration files, edit the existing project request template through those files instead.

- a. List templates in the **openshift-config** namespace:

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

- b. Edit an existing project request template:

```
$ oc edit template <project_request_template> -n openshift-config
```

- c. Add a resource quota definition, such as the preceding **storage-consumption** example, into the existing template. The definition must be added before the **parameters:** section in the template.

2. If you created a project request template, reference it in the cluster's project configuration resource:

- a. Access the project configuration resource for editing:

- By using the web console:
  - i. Navigate to the **Administration** → **Cluster Settings** page.
  - ii. Click **Global Configuration** to view all configuration resources.
  - iii. Find the entry for **Project** and click **Edit YAML**.

- By using the CLI:

- i. Edit the **project.config.openshift.io/cluster** resource:

```
$ oc edit project.config.openshift.io/cluster
```



- b. Update the **spec** section of the project configuration resource to include the **projectRequestTemplate** and **name** parameters. The following example references the default project request template name **project-request**:

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: project-request
```

3. Verify that the resource quota is applied when projects are created:

- a. Create a project:

```
$ oc new-project <project_name>
```

- b. List the project's resource quotas:

```
$ oc get resourcequotas
```

- c. Describe the resource quota in detail:

```
$ oc describe resourcequotas <resource_quota_name>
```

## 5.2. RESOURCE QUOTAS ACROSS MULTIPLE PROJECTS

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

This guide describes how cluster administrators can set and manage resource quotas across multiple projects.

### 5.2.1. Selecting multiple projects during quota creation

When creating quotas, you can select multiple projects based on annotation selection, label selection, or both.

#### Procedure

1. To select projects based on annotations, run the following command:

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

This creates the following **ClusterResourceQuota** object:

```
apiVersion: v1
kind: ClusterResourceQuota
```

```

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

```

- 1 The **ResourceQuotaSpec** object that will be enforced over the selected projects.
- 2 A simple key-value selector for annotations.
- 3 A label selector that can be used to select projects.
- 4 A per-namespace map that describes current quota usage in each selected project.
- 5 The aggregate usage across all selected projects.

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

2. Similarly, to select projects based on labels, run this command:

```

$ oc create clusterresourcequota for-name \ 1
  --project-label-selector=name=frontend \ 2
  --hard=pods=10 --hard=secrets=20

```

- 1 Both **clusterresourcequota** and **clusterquota** are aliases of the same command. **for-name** is the name of the **ClusterResourceQuota** object.
- 2 To select projects by label, provide a key-value pair by using the format **--project-label-selector=key=value**.

This creates the following **ClusterResourceQuota** object definition:

```

apiVersion: v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend

```

### 5.2.2. Viewing applicable cluster resource quotas

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

#### Procedure

1. To view quotas applied to a project, run:

```
$ oc describe AppliedClusterResourceQuota
```

For example:

```

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

```

### 5.2.3. Selection granularity

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

## CHAPTER 6. MONITORING PROJECT AND APPLICATION METRICS USING THE DEVELOPER PERSPECTIVE

The **Monitoring** view in the **Developer** perspective provides options to monitor your project or application metrics, such as CPU, memory, and bandwidth usage, and network related information.

### 6.1. PREREQUISITES

- You have [logged in to the web console](#) and have switched to the **Developer perspective**.
- You have [created and deployed applications on OpenShift Container Platform](#).

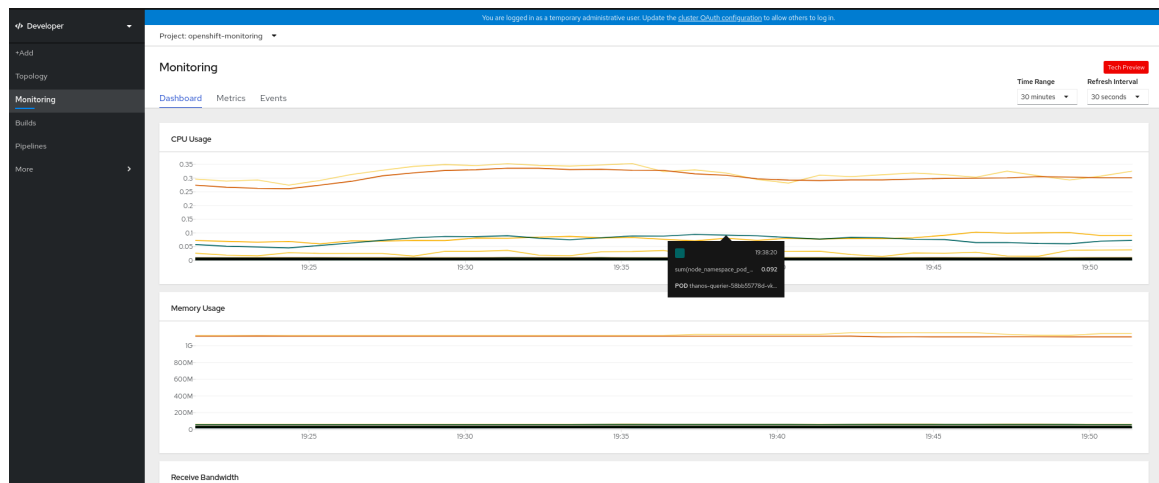
### 6.2. MONITORING YOUR PROJECT METRICS

After you create applications in your project and deploy them, you can use the **Developer** perspective in the web console to see the metrics for your project.

#### Procedure

1. On the left navigation panel of the **Developer** perspective, click **Monitoring** to see the **Dashboard**, **Metrics**, and **Events** for your project.
  - Use the **Dashboard** tab to see graphs depicting the CPU, memory, and bandwidth consumption and network related information, such as the rate of transmitted and received packets and the rate of dropped packets.

Figure 6.1. Monitoring dashboard

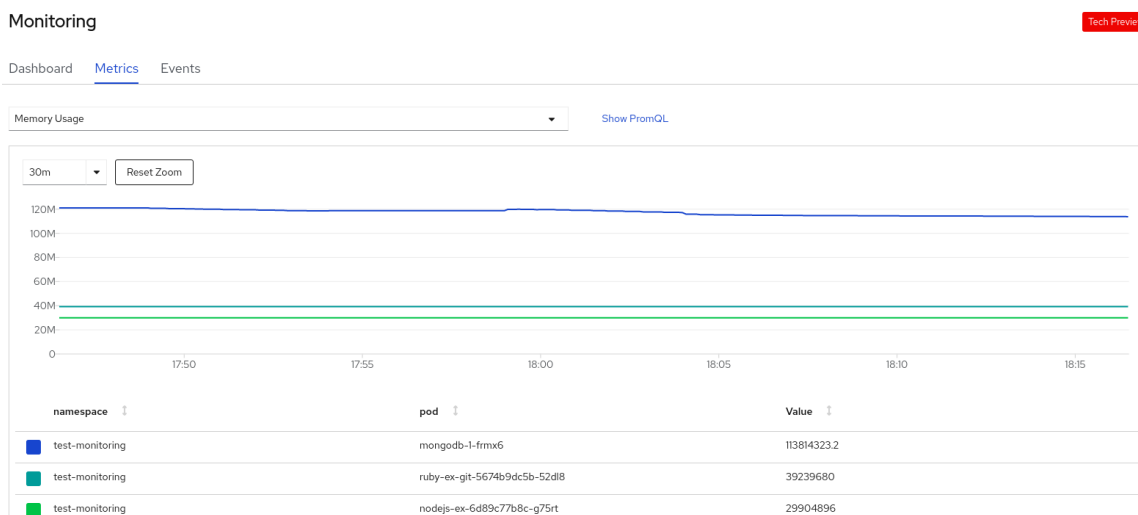


Use the following options to see further details:

- Select an option from the **Time Range** list to determine the time frame for the data being captured.
- Select an option from the **Refresh Interval** list to determine the time period after which the data is refreshed.
- Hover your cursor over the graphs to see specific details for your pod.
- Click on any of the graphs displayed to see the details for that particular metric in the **Metrics** page.

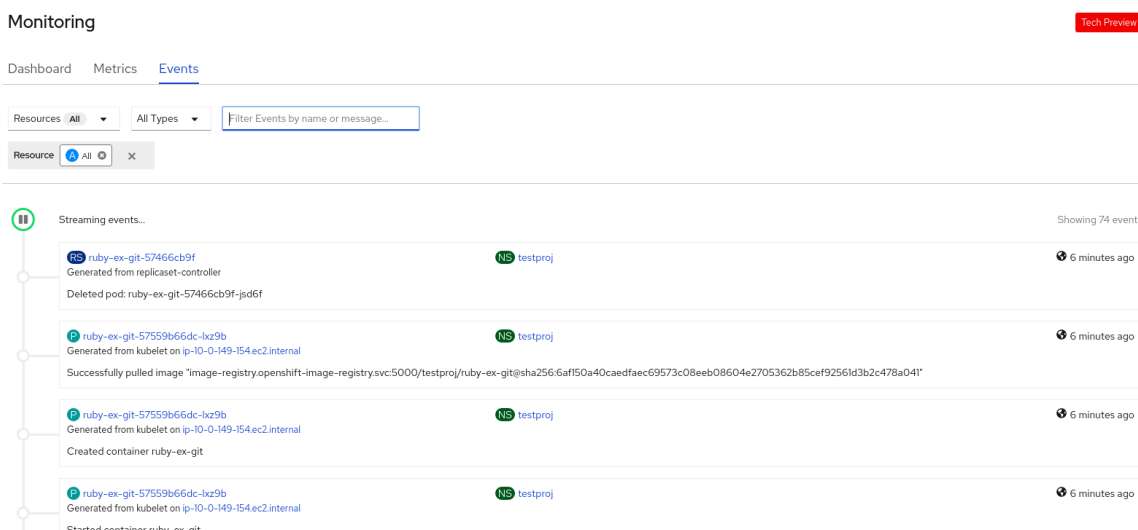
- Use the **Metrics** tab to query for the required project metric.

**Figure 6.2. Monitoring metrics**



- In the **Select Query** list, select an option to filter the required details for your project. The filtered metrics for all the application pods in your project are displayed in the graph. The pods in your project are also listed below.
  - From the list of pods, clear the colored square boxes to remove the metrics for specific pods to further filter your query result.
  - Click **Show PromQL** to see the Prometheus query. You can further modify this query with the help of prompts to customize the query and filter the metrics you want to see for that namespace.
  - Use the drop-down list to set a time range for the data being displayed. You can click **Reset Zoom** to reset it to the default time range.
  - Optionally, in the **Select Query** list, select **Custom Query** to create a custom Prometheus query and filter relevant metrics.
- Use the **Events** tab to see the events for your project.

**Figure 6.3. Monitoring events**



You can filter the displayed events using the following options:

- In the **Resources** list, select a resource to see events for that resource.
- In the **All Types** list, select a type of event to see events relevant to that type.
- Search for specific events using the **Filter events by names or messages** field.

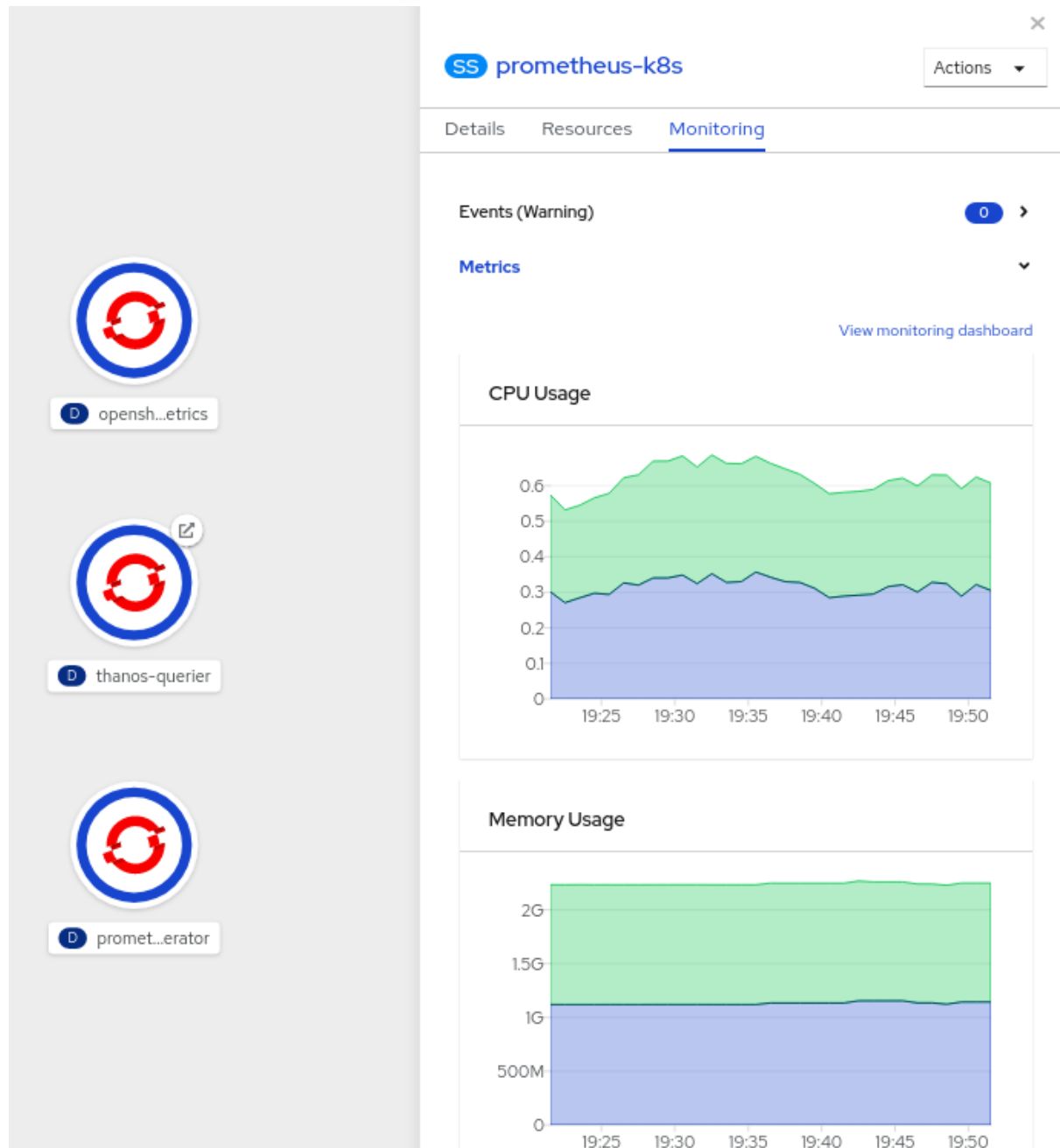
## 6.3. MONITORING YOUR APPLICATION METRICS

After you create applications in your project and deploy them, you can use the **Topology** view in the **Developer** perspective to see the metrics for your application.

### Procedure

1. In the **Topology** view, click the application to see the application details in the right panel.
2. Click the **Monitoring** tab to see the warning events for the application; graphs for CPU, memory, and bandwidth usage; and all the events for the application.

Figure 6.4. Monitoring application metrics



- Click any of the charts to go to the **Metrics** tab to see the detailed metrics for the application.
- Click **View monitoring dashboard** to see the monitoring dashboard for that application.

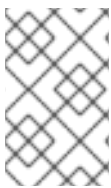
## CHAPTER 7. MONITORING APPLICATION HEALTH BY USING HEALTH CHECKS

In software systems, components can become unhealthy due to transient issues such as temporary connectivity loss, configuration errors, or problems with external dependencies. OpenShift Container Platform applications have a number of options to detect and handle unhealthy containers.

### 7.1. UNDERSTANDING HEALTH CHECKS

A health check periodically performs diagnostics on a running container using any combination of the readiness and liveness health checks.

You can include one or more probes in the specification for the pod that contains the container which you want to perform the health checks.



#### NOTE

If you want to add or edit health checks in an existing pod, you must edit the pod **DeploymentConfig** object or use the **Developer** perspective in the web console. You cannot use the CLI to add or edit health checks for an existing pod.

#### Readiness probe

A *readiness probe* determines if a container is ready to accept service requests. If the readiness probe fails for a container, the kubelet removes the pod from the list of available service endpoints. After a failure, the probe continues to examine the pod. If the pod becomes available, the kubelet adds the pod to the list of available service endpoints.

#### Liveness health check

A *liveness probe* determines if a container is still running. If the liveness probe fails due to a condition such as a deadlock, the kubelet kills the container. The pod then responds based on its restart policy. For example, a liveness probe on a pod with a **restartPolicy** of **Always** or **OnFailure** kills and restarts the container.

You can configure liveness and readiness probes with any of the following types of tests:

- **HTTP GET**: When using an HTTP **GET** test, the test determines the healthiness of the container by using a web hook. The test is successful if the HTTP response code is between **200** and **399**. You can use an HTTP **GET** test with applications that return HTTP status codes when completely initialized.
- **Container Command**: When using a container command test, the probe executes a command inside the container. The probe is successful if the test exits with a **0** status.
- **TCP socket**: When using a TCP socket test, the probe attempts to open a socket to the container. The container is only considered healthy if the probe can establish a connection. You can use a TCP socket test with applications that do not start listening until initialization is complete.

You can configure several fields to control the behavior of a probe:

- **initialDelaySeconds**: The time, in seconds, after the container starts before the probe can be scheduled. The default is 0.



- **periodSeconds:** The delay, in seconds, between performing probes. The default is **10**.
- **timeoutSeconds:** The number of seconds of inactivity after which the probe times out and the container is assumed to have failed. The default is **1**.
- **successThreshold:** The number of times that the probe must report success after a failure in order to reset the container status to successful. The value must be **1** for a liveness probe. The default is **1**.
- **failureThreshold:** The number of times that the probe is allowed to fail. The default is 3. After the specified attempts:
  - for a liveness probe, the container is restarted
  - for a readiness probe, the pod is marked **Unready**



#### NOTE

The **timeoutSeconds** parameter has no effect on the readiness and liveness probes for container command probes, as OpenShift Container Platform cannot time out on an exec call into the container. One way to implement a timeout in a container command probe is by using the **exec-timeout** command to run your liveness or readiness probes, as shown in the examples.

#### Example probes

The following are samples of different probes as they would appear in an object specification.

#### Sample readiness probe with a container command readiness probe in a pod spec

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
...
spec:
  containers:
  - name: goproxy-app 1
    args:
    image: k8s.gcr.io/goproxy:0.1 2
    readinessProbe: 3
      exec: 4
        command: 5
        - cat
        - /tmp/healthy
    ...
```

- 1** The container name.
- 2** The container image to deploy.
- 3** A readiness probe.

- 4 A container command test.
- 5 The commands to execute on the container.

### Sample liveness probe with a container command test that uses a timeout in a pod spec

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
  name: my-application
...
spec:
  containers:
  - name: goproxy-app 1
    args:
    image: k8s.gcr.io/goproxy:0.1 2
    livenessProbe: 3
      exec: 4
        command: 5
          - /bin/bash
          - '-c'
          - timeout 60 /opt/eap/bin/livenessProbe.sh
        periodSeconds: 10 6
        successThreshold: 1 7
        failureThreshold: 3 8
    ...
  ...

```

- 1 The container name.
- 2 Specify the container image to deploy.
- 3 The liveness probe.
- 4 The type of probe, here a container command probe.
- 5 The command line to execute inside the container.
- 6 How often in seconds to perform the probe.
- 7 The number of number of consecutive successes needed to show success after a failure.
- 8 The number of times to try the probe after a failure.

### Sample readiness probe and liveness probe with a TCP socket test in a deployment

```

kind: Deployment
apiVersion: apps/v1
...
spec:
...
template:

```

```

spec:
  containers:
  - resources: {}
    readinessProbe: ❶
      tcpSocket:
        port: 8080
      timeoutSeconds: 1
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
    terminationMessagePath: /dev/termination-log
    name: ruby-ex
    livenessProbe: ❷
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      timeoutSeconds: 1
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
  ...

```

❶ The readiness probe.

❷ The liveness probe.

## 7.2. CONFIGURING HEALTH CHECKS

To configure readiness and liveness probes, add one or more probes to the specification for the pod that contains the container that you want to perform the health checks on.



### NOTE

If you want to add or edit health checks in an existing pod, you must edit the pod **DeploymentConfig** object or use the **Developer** perspective in the web console. You cannot use the CLI to add or edit health checks for an existing pod.

### Procedure

To add probes for a container:

1. Create a **Pod** object to add one or more probes:

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: health-check
    name: my-application
spec:
  containers:
  - name: my-container ❶
    args:

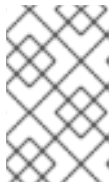
```

```

image: k8s.gcr.io/goproxy:0.1 2
livenessProbe: 3
  tcpSocket: 4
    port: 8080 5
  initialDelaySeconds: 15 6
  timeoutSeconds: 1 7
readinessProbe: 8
  httpGet: 9
    host: my-host 10
    scheme: HTTPS 11
    path: /healthz
    port: 8080 12

```

- 1 Specify the container name.
- 2 Specify the container image to deploy.
- 3 Optional: Create a Liveness probe.
- 4 Specify a test to perform, here a TCP Socket test.
- 5 Specify the port on which the container is listening.
- 6 Specify the time, in seconds, after the container starts before the probe can be scheduled.
- 7 Specify the number of seconds between probes.
- 8 Optional: Create a Readiness probe.
- 9 Specify the type of test to perform, here an HTTP test.
- 10 Specify a host IP address. When **host** is not defined, the **PodIP** is used.
- 11 Specify **HTTP** or **HTTPS**. When **scheme** is not defined, the **HTTP** scheme is used.
- 12 Specify the port on which the container is listening.



#### NOTE

If the **initialDelaySeconds** value is lower than the **periodSeconds** value, the first Readiness probe occurs at some point between the two periods due to an issue with timers.

2. Create the **Pod** object:

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

3. Verify the state of the health check pod:

```
$ oc describe pod my-application
```

#### Example output

■

```

Events:
  Type    Reason      Age    From                                     Message
  ----    -
Normal Scheduled 9s    default-scheduler                       Successfully assigned openshift-logging/liveness-exec to ip-10-0-143-40.ec2.internal
Normal Pulling  2s    kubelet, ip-10-0-143-40.ec2.internal    pulling image "k8s.gcr.io/liveness"
Normal Pulled   1s    kubelet, ip-10-0-143-40.ec2.internal    Successfully pulled image "k8s.gcr.io/liveness"
Normal Created  1s    kubelet, ip-10-0-143-40.ec2.internal    Created container
Normal Started  1s    kubelet, ip-10-0-143-40.ec2.internal    Started container

```

The following is the output of a failed probe that restarted a container:

### Sample Liveness check output with unhealthy container

```
$ oc describe pod pod1
```

### Example output

```

....

Events:
  Type    Reason      Age    From                                     Message
  ----    -
Normal Scheduled <unknown>                               Successfully assigned aaa/liveness-http to ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj
Normal AddedInterface 47s    multus                               Add eth0 [10.129.2.11/23]
Normal Pulled 46s    kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Successfully pulled image "k8s.gcr.io/liveness" in 773.406244ms
Normal Pulled 28s    kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Successfully pulled image "k8s.gcr.io/liveness" in 233.328564ms
Normal Created 10s (x3 over 46s) kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Created container liveness
Normal Started 10s (x3 over 46s) kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Started container liveness
Warning Unhealthy 10s (x6 over 34s) kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Liveness probe failed: HTTP probe failed with statuscode: 500
Normal Killing 10s (x2 over 28s) kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Container liveness failed liveness probe, will be restarted
Normal Pulling 10s (x3 over 47s) kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Pulling image "k8s.gcr.io/liveness"
Normal Pulled 10s    kubelet, ci-ln-37hz77b-f76d1-wdpjv-worker-b-snrzj    Successfully pulled image "k8s.gcr.io/liveness" in 244.116568ms

```

## CHAPTER 8. IDLING APPLICATIONS

Cluster administrators can idle applications to reduce resource consumption. This is useful when the cluster is deployed on a public cloud where cost is related to resource consumption.

If any scalable resources are not in use, OpenShift Container Platform discovers and idles them by scaling their replicas to **0**. The next time network traffic is directed to the resources, the resources are unidled by scaling up the replicas, and normal operation continues.

Applications are made of services, as well as other scalable resources, such as deployment configs. The action of idling an application involves idling all associated resources.

### 8.1. IDLING APPLICATIONS

Idling an application involves finding the scalable resources (deployment configurations, replication controllers, and others) associated with a service. Idling an application finds the service and marks it as idled, scaling down the resources to zero replicas.

You can use the **oc idle** command to idle a single service, or use the **--resource-names-file** option to idle multiple services.

#### 8.1.1. Idling a single service

##### Procedure

1. To idle a single service, run:

```
$ oc idle <service>
```

#### 8.1.2. Idling multiple services

Idling multiple services is helpful if an application spans across a set of services within a project, or when idling multiple services in conjunction with a script in order to idle multiple applications in bulk within the same project.

##### Procedure

1. Create a file containing a list of the services, each on their own line.
2. Idle the services using the **--resource-names-file** option:

```
$ oc idle --resource-names-file <filename>
```



##### NOTE

The **idle** command is limited to a single project. For idling applications across a cluster, run the **idle** command for each project individually.

### 8.2. UNIDLING APPLICATIONS

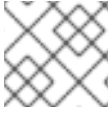
Application services become active again when they receive network traffic and are scaled back up their previous state. This includes both traffic to the services and traffic passing through routes.

Applications can also be manually unidled by scaling up the resources.

### Procedure

1. To scale up a DeploymentConfig, run:

```
$ oc scale --replicas=1 dc <dc_name>
```



### NOTE

Automatic unidling by a router is currently only supported by the default HAProxy router.

## CHAPTER 9. PRUNING OBJECTS TO RECLAIM RESOURCES

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

Cluster administrators can periodically prune older versions of objects from the cluster that are no longer required. For example, by pruning images you can delete older images and layers that are no longer in use, but are still taking up disk space.

### 9.1. BASIC PRUNING OPERATIONS

The CLI groups prune operations under a common parent command:

```
$ oc adm prune <object_type> <options>
```

This specifies:

- The **<object\_type>** to perform the action on, such as **groups, builds, deployments, or images**.
- The **<options>** supported to prune that object type.

### 9.2. PRUNING GROUPS

To prune groups records from an external provider, administrators can run the following command:

```
$ oc adm prune groups \
  --sync-config=path/to/sync/config [<options>]
```

Table 9.1. Prune groups CLI configuration options

Options	Description
<b>--confirm</b>	Indicate that pruning should occur, instead of performing a dry-run.
<b>--blacklist</b>	Path to the group blacklist file.
<b>--whitelist</b>	Path to the group whitelist file.
<b>--sync-config</b>	Path to the synchronization configuration file.

To see the groups that the prune command deletes:

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml
```

To perform the prune operation:

```
$ oc adm prune groups --sync-config=ldap-sync-config.yaml --confirm
```

### 9.3. PRUNING DEPLOYMENTCONFIG OBJECTS



In order to prune **DeploymentConfig** objects that are no longer required by the system due to age and status, administrators can run the following command:

```
$ oc adm prune deployments [<options>]
```

**Table 9.2. Prune deployments CLI configuration options**

Option	Description
<b>--confirm</b>	Indicate that pruning should occur, instead of performing a dry-run.
<b>--orphans</b>	Prune all deployments that no longer have a <b>DeploymentConfig</b> object, has status of <b>Complete</b> or <b>Failed</b> , and has a replica count of zero.
<b>--keep-complete=&lt;N&gt;</b>	Per the <b>DeploymentConfig</b> object, keep the last <b>N</b> deployments that have a status of <b>Complete</b> and replica count of zero. (default <b>5</b> )
<b>--keep-failed=&lt;N&gt;</b>	Per the <b>DeploymentConfig</b> object, keep the last <b>N</b> deployments that have a status of <b>Failed</b> and replica count of zero. (default <b>1</b> )
<b>--keep-younger-than=&lt;duration&gt;</b>	Do not prune any object that is younger than <b>&lt;duration&gt;</b> relative to the current time. (default <b>60m</b> ) Valid units of measurement include nanoseconds ( <b>ns</b> ), microseconds ( <b>us</b> ), milliseconds ( <b>ms</b> ), seconds ( <b>s</b> ), minutes ( <b>m</b> ), and hours ( <b>h</b> ).

To see what a pruning operation would delete:

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

To actually perform the prune operation:

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

## 9.4. PRUNING BUILDS

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

```
$ oc adm prune builds [<options>]
```

**Table 9.3. Prune builds CLI configuration options**

Option	Description
<b>--confirm</b>	Indicate that pruning should occur, instead of performing a dry-run.

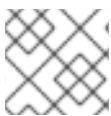
Option	Description
<b>--orphans</b>	Prune all builds whose build configuration no longer exists, status is complete, failed, error, or canceled.
<b>--keep-complete=&lt;N&gt;</b>	Per build configuration, keep the last <b>N</b> builds whose status is complete (default <b>5</b> ).
<b>--keep-failed=&lt;N&gt;</b>	Per build configuration, keep the last <b>N</b> builds whose status is failed, error, or canceled (default <b>1</b> ).
<b>--keep-younger-than=&lt;duration&gt;</b>	Do not prune any object that is younger than <b>&lt;duration&gt;</b> relative to the current time (default <b>60m</b> ).

To see what a pruning operation would delete:

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

To actually perform the prune operation:

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



#### NOTE

Developers can enable automatic build pruning by modifying their build configuration.

#### Additional resources

- [Performing advanced builds → Pruning builds](#)

## 9.5. AUTOMATICALLY PRUNING IMAGES

Images that are no longer required by the system due to age, status, or exceed limits are automatically pruned. Cluster administrators can configure the Pruning Custom Resource, or suspend it.

#### Prerequisites

- Cluster administrator permissions.
- Install the **oc** CLI.

#### Procedure

- Verify that the object named **imagepruners.imageregistry.operator.openshift.io/cluster** contains the following **spec** and **status** fields:

```

spec:
  schedule: 0 0 * * * 1
  suspend: false 2
  keepTagRevisions: 3 3
  keepYoungerThan: 3600000000000 4
  resources: {} 5
  affinity: {} 6
  nodeSelector: {} 7
  tolerations: [] 8
  successfulJobsHistoryLimit: 3 9
  failedJobsHistoryLimit: 3 10
status:
  observedGeneration: 2 11
  conditions: 12
  - type: Available
    status: "True"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Ready
    message: "Periodic image pruner has been created."
  - type: Scheduled
    status: "True"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Scheduled
    message: "Image pruner job has been scheduled."
  - type: Failed
    status: "False"
    lastTransitionTime: 2019-10-09T03:13:45
    reason: Succeeded
    message: "Most recent image pruning job succeeded."

```

- 1 **schedule:** **CronJob** formatted schedule. This is an optional field, default is daily at midnight.
- 2 **suspend:** If set to **true**, the **CronJob** running pruning is suspended. This is an optional field, default is **false**. The initial value on new clusters is **true**.
- 3 **keepTagRevisions:** The number of revisions per tag to keep. This is an optional field, default is **3**. The initial value is **3**.
- 4 **keepYoungerThan:** Retain images younger than this duration in nanoseconds. This is an optional field, default is **3600000000000** (60 minutes).
- 5 **resources:** Standard **Pod** resource requests and limits. This is an optional field.
- 6 **affinity:** Standard pod affinity. This is an optional field.
- 7 **nodeSelector:** Standard pod node selector for the image pruner pod. This is an optional field.
- 8 **tolerations:** Standard pod tolerations. This is an optional field.
- 9 **successfulJobsHistoryLimit:** The maximum number of successful jobs to retain. Must be **>= 1** to ensure metrics are reported. This is an optional field, default is **3**. The initial value is **3**.
- 10 **failedJobsHistoryLimit:** The maximum number of failed jobs to retain. Must be **>= 1** to ensure metrics are reported. This is an optional field, default is **3**. The initial value is **3**.

**11** **observedGeneration:** The generation observed by the Operator.

**12** **conditions:** The standard condition objects with the following types:

- **Available:** Indicates if the pruning job has been created. Reasons can be Ready or Error.
- **Scheduled:** Indicates if the next pruning job has been scheduled. Reasons can be Scheduled, Suspended, or Error.
- **Failed:** Indicates if the most recent pruning job failed.

### IMPORTANT

The Image Registry Operator's behavior for managing the pruner is orthogonal to the **managementState** specified on the Image Registry Operator's **ClusterOperator** object. If the Image Registry Operator is not in the **Managed** state, the image pruner can still be configured and managed by the Pruning Custom Resource.

However, the **managementState** of the Image Registry Operator alters the behavior of the deployed image pruner job:

- **Managed:** the **--prune-registry** flag for the image pruner is set to **true**.
- **Removed:** the **--prune-registry** flag for the image pruner is set to **false**, meaning it only prunes image metadata in etcd.
- **Unmanaged:** the **--prune-registry** flag for the image pruner is set to **false**.

## 9.6. MANUALLY PRUNING IMAGES

The pruning custom resource enables automatic image pruning. However, administrators can manually prune images that are no longer required by the system due to age, status, or exceed limits. There are two methods to manually prune images:

- Running image pruning as a **Job** or **CronJob** on the cluster.
- Running the **oc adm prune images** command.

### Prerequisites

- To prune images, you must first log in to the CLI as a user with an access token. The user must also have the **system:image-pruner** cluster role or greater (for example, **cluster-admin**).
- Expose the image registry.

### Procedure

To manually prune images that are no longer required by the system due to age, status, or exceed limits, use one of the following methods:

- Run image pruning as a **Job** or **CronJob** on the cluster by creating a YAML file for the **pruner** service account, for example:

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

## Example output

```

kind: List
apiVersion: v1
items:
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: pruner
    namespace: openshift-image-registry
- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: openshift-image-registry-pruner
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: system:image-pruner
  subjects:
  - kind: ServiceAccount
    name: pruner
    namespace: openshift-image-registry
- apiVersion: batch/v1beta1
  kind: CronJob
  metadata:
    name: image-pruner
    namespace: openshift-image-registry
  spec:
    schedule: "0 0 * * *"
    concurrencyPolicy: Forbid
    successfulJobsHistoryLimit: 1
    failedJobsHistoryLimit: 3
    jobTemplate:
      spec:
        template:
          spec:
            restartPolicy: OnFailure
            containers:
            - image: "quay.io/openshift/origin-cli:4.1"
              resources:
                requests:
                  cpu: 1
                  memory: 1Gi
            terminationMessagePolicy: FallbackToLogsOnError
            command:
            - oc
            args:
            - adm
            - prune
            - images
            - --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
            - --keep-tag-revisions=5
            - --keep-younger-than=96h
            - --confirm=true
            name: image-pruner
            serviceAccountName: pruner

```

- Run the **oc adm prune images [<options>]** command:

```
$ oc adm prune images [<options>]
```

Pruning images removes data from the integrated registry unless **--prune-registry=false** is used.

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

By default, the integrated registry caches metadata of blobs to reduce the number of requests to storage, and to increase the request-processing speed. Pruning does not update the integrated registry cache. Images that still contain pruned layers after pruning will be broken because the pruned layers that have metadata in the cache will not be pushed. Therefore, you must redeploy the registry to clear the cache after pruning:

```
$ oc rollout restart deployment/image-registry -n openshift-image-registry
```

If the integrated registry uses a Redis cache, you must clean the database manually.

If redeploying the registry after pruning is not an option, then you must permanently disable the cache.

**oc adm prune images** operations require a route for your registry. Registry routes are not created by default.

The **Prune images CLI configuration options** table describes the options you can use with the **oc adm prune images <options>** command.

**Table 9.4. Prune images CLI configuration options**

Option	Description
<b>--all</b>	Include images that were not pushed to the registry, but have been mirrored by pullthrough. This is on by default. To limit the pruning to images that were pushed to the integrated registry, pass <b>--all=false</b> .
<b>--certificate-authority</b>	The path to a certificate authority file to use when communicating with the OpenShift Container Platform-managed registries. Defaults to the certificate authority data from the current user's configuration file. If provided, a secure connection is initiated.
<b>--confirm</b>	Indicate that pruning should occur, instead of performing a test-run. This requires a valid route to the integrated container image registry. If this command is run outside of the cluster network, the route must be provided using <b>--registry-url</b> .
<b>--force-insecure</b>	Use caution with this option. Allow an insecure connection to the container registry that is hosted via HTTP or has an invalid HTTPS certificate.

Option	Description
<b>--keep-tag-revisions=&lt;N&gt;</b>	For each imagestream, keep up to at most <b>N</b> image revisions per tag (default <b>3</b> ).
<b>--keep-younger-than=&lt;duration&gt;</b>	Do not prune any image that is younger than <b>&lt;duration&gt;</b> relative to the current time. Alternately, do not prune any image that is referenced by any other object that is younger than <b>&lt;duration&gt;</b> relative to the current time (default <b>60m</b> ).
<b>--prune-over-size-limit</b>	Prune each image that exceeds the smallest limit defined in the same project. This flag cannot be combined with <b>--keep-tag-revisions</b> nor <b>--keep-younger-than</b> .
<b>--registry-url</b>	The address to use when contacting the registry. The command attempts to use a cluster-internal URL determined from managed images and image streams. In case it fails (the registry cannot be resolved or reached), an alternative route that works needs to be provided using this flag. The registry host name can be prefixed by <b>https://</b> or <b>http://</b> , which enforces particular connection protocol.
<b>--prune-registry</b>	<p>In conjunction with the conditions stipulated by the other options, this option controls whether the data in the registry corresponding to the OpenShift Container Platform image API object is pruned. By default, image pruning processes both the image API objects and corresponding data in the registry.</p> <p>This option is useful when you are only concerned with removing etcd content, to reduce the number of image objects but are not concerned with cleaning up registry storage, or if you intend to do that separately by hard pruning the registry during an appropriate maintenance window for the registry.</p>

### 9.6.1. Image prune conditions

You can apply conditions to your manually pruned images.

- To remove any image managed by OpenShift Container Platform, or images with the annotation **openshift.io/image.managed**:
  - Created at least **--keep-younger-than** minutes ago and are not currently referenced by any:
    - Pods created less than **--keep-younger-than** minutes ago
    - Image streams created less than **--keep-younger-than** minutes ago
    - Running pods
    - Pending pods
    - Replication controllers
    - Deployments

- Deployment configs
- Replica sets
- Build configurations
- Builds
- **--keep-tag-revisions** most recent items in **stream.status.tags[].items**
- That are exceeding the smallest limit defined in the same project and are not currently referenced by any:
  - Running pods
  - Pending pods
  - Replication controllers
  - Deployments
  - Deployment configs
  - Replica sets
  - Build configurations
  - Builds
- There is no support for pruning from external registries.
- When an image is pruned, all references to the image are removed from all image streams that have a reference to the image in **status.tags**.
- Image layers that are no longer referenced by any images are removed.



## NOTE

The **--prune-over-size-limit** flag cannot be combined with the **--keep-tag-revisions** flag nor the **--keep-younger-than** flags. Doing so returns information that this operation is not allowed.

Separating the removal of OpenShift Container Platform image API objects and image data from the registry by using **--prune-registry=false**, followed by hard pruning the registry, can narrow timing windows and is safer when compared to trying to prune both through one command. However, timing windows are not completely removed.

For example, you can still create a Pod referencing an image as pruning identifies that image for pruning. You should still keep track of an API object created during the pruning operations that might reference images so that you can mitigate any references to deleted content.

Re-doing the pruning without the **--prune-registry** option or with **--prune-registry=true** does not lead to pruning the associated storage in the image registry for images previously pruned by **--prune-registry=false**. Any images that were pruned with **--prune-registry=false** can only be deleted from registry storage by hard pruning the registry.



## 9.6.2. Running the image prune operation

### Procedure

1. To see what a pruning operation would delete:
  - a. Keeping up to three tag revisions, and keeping resources (images, image streams, and pods) younger than 60 minutes:

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

- b. Pruning every image that exceeds defined limits:

```
$ oc adm prune images --prune-over-size-limit
```

2. To perform the prune operation with the options from the previous step:

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

```
$ oc adm prune images --prune-over-size-limit --confirm
```

## 9.6.3. Using secure or insecure connections

The secure connection is the preferred and recommended approach. It is done over HTTPS protocol with a mandatory certificate verification. The **prune** command always attempts to use it if possible. If it is not possible, in some cases it can fall-back to insecure connection, which is dangerous. In this case, either certificate verification is skipped or plain HTTP protocol is used.

The fall-back to insecure connection is allowed in the following cases unless **--certificate-authority** is specified:

1. The **prune** command is run with the **--force-insecure** option.
2. The provided **registry-url** is prefixed with the **http://** scheme.
3. The provided **registry-url** is a local-link address or **localhost**.
4. The configuration of the current user allows for an insecure connection. This can be caused by the user either logging in using **--insecure-skip-tls-verify** or choosing the insecure connection when prompted.



### IMPORTANT

If the registry is secured by a certificate authority different from the one used by OpenShift Container Platform, it must be specified using the **--certificate-authority** flag. Otherwise, the **prune** command fails with an error.

## 9.6.4. Image pruning problems

### Images not being pruned

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

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

```
$ image_name="sha:abz"
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range $ti, $tag := $is.status.tags}}\
  '{{range $ii, $item := $tag.items}}{{if eq $item.image ""${image_name}"\
  $"}}{{$.metadata.name}}:{{$.tag.tag}} at position {{$.ii}} out of {{len $.tag.items}}\n\
  '{{end}}'{{end}}'
myapp:v2 at position 4 out of 5
myapp:v2.1 at position 2 out of 2
myapp:v2.1-may-2016 at position 0 out of 1
```

When default options are used, the image is never pruned because it occurs at position **0** in a history of **myapp:v2.1-may-2016** tag. For an image to be considered for pruning, the administrator must either:

- Specify **--keep-tag-revisions=0** with the **oc adm prune images** command.



#### WARNING

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

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

Tags having a date or time of a particular image's build in their names should be avoided, unless the image must be preserved for an undefined amount of time. Such tags tend to have just one image in their history, which prevents them from ever being pruned.

#### Using a secure connection against insecure registry

If you see a message similar to the following in the output of the **oc adm prune images** command, then your registry is not secured and the **oc adm prune images** client attempts to use a secure connection:

```
error: error communicating with registry: Get https://172.30.30.30:5000/healthz: http: server gave HTTP response to HTTPS client
```

- The recommended solution is to secure the registry. Otherwise, you can force the client to use an insecure connection by appending **--force-insecure** to the command; however, this is not recommended.

#### Using an insecure connection against a secured registry

If you see one of the following errors in the output of the **oc adm prune images** command, it means that your registry is secured using a certificate signed by a certificate authority other than the one used by **oc adm prune images** client for connection verification:

```
error: error communicating with registry: Get http://172.30.30.30:5000/healthz: malformed HTTP
response "\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get https://172.30.30.30:5000/healthz: x509: certificate
signed by unknown authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"]
```

By default, the certificate authority data stored in the user's configuration files is used; the same is true for communication with the master API.

Use the **--certificate-authority** option to provide the right certificate authority for the container image registry server.

### Using the wrong certificate authority

The following error means that the certificate authority used to sign the certificate of the secured container image registry is different from the authority used by the client:

```
error: error communicating with registry: Get https://172.30.30.30:5000/: x509: certificate signed by
unknown authority
```

Make sure to provide the right one with the flag **--certificate-authority**.

As a workaround, the **--force-insecure** flag can be added instead. However, this is not recommended.

### Additional resources

- [Accessing the registry](#)
- [Exposing the registry](#)
- See [Image Registry Operator in OpenShift Container Platform](#) for information on how to create a registry route.

## 9.7. HARD PRUNING THE REGISTRY

The OpenShift Container Registry can accumulate blobs that are not referenced by the OpenShift Container Platform cluster's etcd. The basic pruning images procedure, therefore, is unable to operate on them. These are called *orphaned blobs*.

Orphaned blobs can occur from the following scenarios:

- Manually deleting an image with **oc delete image <sha256:image-id>** command, which only removes the image from etcd, but not from the registry's storage.
- Pushing to the registry initiated by daemon failures, which causes some blobs to get uploaded, but the image manifest (which is uploaded as the very last component) does not. All unique image blobs become orphans.
- OpenShift Container Platform refusing an image because of quota restrictions.
- The standard image pruner deleting an image manifest, but is interrupted before it deletes the related blobs.
- A bug in the registry pruner, which fails to remove the intended blobs, causing the image objects referencing them to be removed and the blobs becoming orphans.

*Hard pruning* the registry, a separate procedure from basic image pruning, allows cluster administrators to remove orphaned blobs. You should hard prune if you are running out of storage space in your OpenShift Container Registry and believe you have orphaned blobs.

This should be an infrequent operation and is necessary only when you have evidence that significant numbers of new orphans have been created. Otherwise, you can perform standard image pruning at regular intervals, for example, once a day (depending on the number of images being created).

## Procedure

To hard prune orphaned blobs from the registry:

### 1. Log in.

Log in to the cluster with the CLI as **kubeadmin** or another privileged user that has access to the **openshift-image-registry** namespace.

### 2. Run a basic image prune

Basic image pruning removes additional images that are no longer needed. The hard prune does not remove images on its own. It only removes blobs stored in the registry storage. Therefore, you should run this just before the hard prune.

### 3. Switch the registry to read-only mode.

If the registry is not running in read-only mode, any pushes happening at the same time as the prune will either:

- fail and cause new orphans, or
- succeed although the images cannot be pulled (because some of the referenced blobs were deleted).

Pushes will not succeed until the registry is switched back to read-write mode. Therefore, the hard prune must be carefully scheduled.

To switch the registry to read-only mode:

- In **configs.imageregistry.operator.openshift.io/cluster**, set **spec.readOnly** to **true**:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec": {"readOnly":true}}' --type=merge
```

### 4. Add the **system:image-pruner** role.

The service account used to run the registry instances requires additional permissions in order to list some resources.

- Get the service account name:

```
$ service_account=$(oc get -n openshift-image-registry \
-o jsonpath='{.spec.template.spec.serviceAccountName}' deploy/image-registry)
```

- Add the **system:image-pruner** cluster role to the service account:

```
$ oc adm policy add-cluster-role-to-user \
system:image-pruner -z \
${service_account} -n openshift-image-registry
```

### 5. (Optional) Run the pruner in dry-run mode.

To see how many blobs would be removed, run the hard pruner in dry-run mode. No changes are actually made:

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry \
  /usr/bin/dockerregistry -prune=check
```

Alternatively, to get the exact paths for the prune candidates, increase the logging level:

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry env REGISTRY_LOG_LEVEL=info \
  /usr/bin/dockerregistry -prune=check
```

### Truncated sample output

```
$ oc exec image-registry-3-vhndw \
  -- /bin/sh -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -prune=check'

time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune (dry-run mode)"
distribution_version="v2.4.1+unknown" kubernetes_version=v1.6.1+${Format:%h$}
openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663fa5"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete blob:
sha256:0022d49612807cb348cab562c072ef34d756adfe0100a61952cbcb87ee6578a"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a37c"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7663"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeec207e4fad2a5bafd28afbef57e"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5e06"
go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data
```

## 6. Run the hard prune.

Execute the following command inside one running instance of a **image-registry** pod to run the hard prune:

```
$ oc -n openshift-image-registry \
  rsh deploy/image-registry \
  /usr/bin/dockerregistry -prune=delete
```

### Sample output

■

```
$ oc exec image-registry-3-vhndw \  
  -- /usr/bin/dockerregistry -prune=delete
```

```
Deleted 13374 blobs  
Freed up 2.835 GiB of disk space
```

#### 7. Switch the registry back to read-write mode.

After the prune is finished, the registry can be switched back to read-write mode. In **configs.imageregistry.operator.openshift.io/cluster**, set **spec.readOnly** to **false**:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster -p '{"spec":{"readOnly":false}}' -  
  -type=merge
```

## 9.8. PRUNING CRON JOBS

Cron jobs can perform pruning of successful jobs, but might not properly handle failed jobs. Therefore, the cluster administrator should perform regular cleanup of jobs manually. They should also restrict the access to cron jobs to a small group of trusted users and set appropriate quota to prevent the cron job from creating too many jobs and pods.

### Additional resources

- [Running tasks in pods using jobs](#)
- [Resource quotas across multiple projects](#)
- [Using RBAC to define and apply permissions](#)

## CHAPTER 10. USING THE RED HAT MARKETPLACE

The [Red Hat Marketplace](#) is an open cloud marketplace that makes it easy to discover and access certified software for container-based environments that run on public clouds and on-premises.

### 10.1. RED HAT MARKETPLACE FEATURES

Cluster administrators can use [the Red Hat Marketplace](#) to manage software on OpenShift Container Platform, give developers self-service access to deploy application instances, and correlate application usage against a quota.

#### 10.1.1. Connect OpenShift Container Platform clusters to the Marketplace

Cluster administrators can install a common set of applications on OpenShift Container Platform clusters that connect to the Marketplace. They can also use the Marketplace to track cluster usage against subscriptions or quotas. Users that they add by using the Marketplace have their product usage tracked and billed to their organization.

During the [cluster connection process](#), a Marketplace Operator is installed that updates the image registry secret, manages the catalog, and reports application usage.

#### 10.1.2. Install applications

Cluster administrators can [install Marketplace applications](#) from within OperatorHub in OpenShift Container Platform, or from the [Marketplace web application](#).

You can access installed applications from the web console by clicking **Operators > Installed Operators**

#### 10.1.3. Deploy applications from different perspectives

You can deploy Marketplace applications from the web console's Administrator and Developer perspectives.

##### The Developer perspective

Developers can access newly installed capabilities by using the Developer perspective.

For example, after a database Operator is installed, a developer can create an instance from the catalog within their project. Database usage is aggregated and reported to the cluster administrator.

This perspective does not include Operator installation and application usage tracking.

##### The Administrator perspective

Cluster administrators can access Operator installation and application usage information from the Administrator perspective.

They can also launch application instances by browsing custom resource definitions (CRDs) in the **Installed Operators** list.