Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster (DO280)

# Chapter 1.  Declarative Resource Management

[Resource Manifests](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01)

[Guided Exercise: Resource Manifests](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01s02)

[Kustomize Overlays](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01s03)

[Guided Exercise: Kustomize Overlays](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01s04)

[Lab: Declarative Resource Management](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01s05)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch01s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Deploy and update applications from resource manifests that are parameterized for different target environments. |
| **Objectives** | * Deploy and update applications from resource manifests that are stored as YAML files. * Deploy and update applications from resource manifests that are augmented by Kustomize. |
| **Sections** | * Resource Manifests (and Guided Exercise) * Kustomize Overlays (and Guided Exercise) |
| **Lab** | * Declarative Resource Management |
|  |  |

## **Resource Manifests**

### **Objectives**

* Deploy and update applications from resource manifests that are stored as YAML files.

An application in a Kubernetes cluster often consists of multiple resources that work together. Each resource has a definition and a configuration. Many of the resource configurations share common attributes that must match to operate correctly. Imperative commands configure each resource, one at time. However, using imperative commands has some issues:

* Impaired reproducibility
* Lacking version control
* Lacking support for GitOps

Rather than imperative commands, declarative commands are instead the preferred way to manage resources, by using resource manifests. A resource manifest is a file, in JSON or YAML format, with resource definition and configuration information. Resource manifests simplify the management of Kubernetes resources, by encapsulating all the attributes of an application in a file or a set of related files. Kubernetes uses declarative commands to read the resource manifests and to apply changes to the cluster to meet the state that the resource manifest defines.

The resource manifests are in YAML or JSON format, and thus can be version-controlled. Version control of resource manifests enables tracing of configuration changes. As such, adverse changes can be rolled back to an earlier version to support recoverability.

Resource manifests ensure that applications can be precisely reproduced, typically with a single command to deploy many resources. The reproducibility from resource manifests supports the automation of the GitOps practices of continuous integration and continuous delivery (CI/CD).

#### **Imperative Versus Declarative Workflows**

The Kubernetes CLI uses both imperative and declarative commands. Imperative commands perform an action that is based on a command, and use command names that closely reflect the action. In contrast, declarative commands use a resource manifest file to declare the intended state of a resource.

A Kubernetes manifest is a YAML- or JSON-formatted file with declaration statements for Kubernetes resources such as deployments, pods, or services. Instead of using imperative commands to create Kubernetes resources, manifest files provide all the details for the resource in a single file. Working with manifest files enables the use of more reproducible processes. Instead of reproducing sequences of imperative commands, manifest files contain the entire definition of resources and can be applied in a single step. Using manifest files is also useful for tracking system configuration changes in a source code management system.

Given a new or updated resource manifest, Kubernetes provides commands that compare the intended state that is specified in the resource manifest to the current state of the resource. These commands then apply transformations to the current state to match the intended state.

#### **Imperative workflow**

An imperative workflow is useful for developing and testing. The following example uses the kubectl create deployment imperative command, to create a deployment for a MYSQL database.

[user@host ~]$ **kubectl create deployment db-pod --port 3306 \**

**--image registry.ocp4.example.com:8443/rhel8/mysql-80**

Warning: would violate PodSecurity "restricted:v1.24": ...output omitted...

deployment.apps/db-pod created

### NOTE

A warning indicates that the pod would violate several policies. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

In addition to using verbs that reflect the action of the command, imperative commands use options to provide the details. The example command uses the --port and the --image options to provide the required details to create the deployment.

The use of imperative commands affects applying changes to live resources. For example, the pod from the previous deployment would fail to start due to missing environment variables. The following kubectl set env deployment imperative command resolves the problem by adding the required environment variables to the deployment:

[user@host ~]$ **kubectl set env deployment/db-pod \**

**MYSQL\_USER='user1' \**

**MYSQL\_PASSWORD='mypa55w0rd' \**

**MYSQL\_DATABASE='items'**

Warning: would violate PodSecurity "restricted:v1.24": ...output omitted...

deployment.apps/db-pod updated

Executing this kubectl set env deployment command changes the deployment resource named db-pod, and provides the extra needed variables to start the container. A developer can continue building out the application, by using imperative commands to add components, such as services, routes, volume mounts, and persistent volume claims. With the addition of each component, the developer can run tests to ensure that the component correctly executes the intended function.

Imperative commands are useful for developing and experimenting. With imperative commands, a developer can build up an application one component at a time. When a component is added, the Kubernetes cluster provides error messages that are specific to the component. The process is analogous to using a debugger to step through code execution one line at a time. Using imperative commands usually provides clearer error messages, because an error occurs after adding a specific component.

However, long command lines and a fragmented application deployment are not ideal for deploying an application in production. With imperative commands, changes are a sequence of commands that must be maintained to reflect the intended state of the resources. The sequence of commands must be tracked and kept up to date.

#### **Using Declarative Commands**

Instead of tracking a sequence of commands, a manifest file captures the intended state of the sequence. In contrast to using imperative commands, declarative commands use a manifest file, or a set of manifest files, to combine all the details for creating those components into YAML files that can be applied in a single command. Future changes to the manifest files require only reapplying the manifests. Instead of tracking a sequence of complex commands, version control systems can track changes to the manifest file.

Although manifest files can also use the JSON syntax, YAML is generally preferred and is more popular. To continue the debugging analogy, debugging an application that is deployed from manifests is similar to trying to debug a full, completed running application. It can take more effort to find the source of the error, especially when the error is not a result of manifest errors.

#### **Creating Kubernetes Manifests**

Creating manifest files from scratch can take time. You can use the following techniques to provide a starting point for your manifest files:

* Use the YAML view of a resource from the web console.
* Use imperative commands with the --dry-run=client option to generate manifests that correspond to the imperative command.

The kubectl explain command provides the details for any field in the manifest. For example, use the kubectl explain deployment.spec.template.spec command to view field descriptions that specify a pod object within a deployment manifest.

To create a starter deployment manifest, use the kubectl create deployment command to generate a manifest by using the --dry-run=client option:

[user@host ~]$ **kubectl create deployment hello-openshift -o yaml \**

**--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 \**

**--save-config \**

**--dry-run=client \**

**> ~/my-app/example-deployment.yaml**

|  |  |
| --- | --- |
|  | The --save-config option adds configuration attributes that declarative commands use. For deployments resources, this option saves the resource configuration in an kubectl.kubernetes.io/last-applied-configuration annotation. |
|  | The --dry-run=client option prevents the command from creating resources in the cluster. |

The following example shows a minimal deployment manifest file, not production-ready, for the hello-openshift deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

annotations:

...output omitted...

creationTimestamp: null

labels:

app: hello-openshift

name: hello-openshift

spec:

replicas: 1

selector:

matchLabels:

app: hello-openshift

strategy: {}

template:

metadata:

creationTimestamp: null

labels:

app: hello-openshift

spec:

containers:

- image: quay.io/redhattraining/hello-world-nginx:v1.0

name: hello-world-nginx

resources: {}

status: {}

When using imperative commands to create manifests, the resulting manifests might contain fields that are not necessary for creating a resource. For example, the following example changes the manifest by removing the empty and null fields. Removing unnecessary fields can significantly reduce the length of the manifests, and in turn reduce the overhead to work with them.

Additionally, you might need to further customize the manifests. For example, in a deployment, you might customize the number of replicas, or declare the ports that the deployment provides. The following notes explain the additional changes:

apiVersion: apps/v1

kind: Deployment

metadata:

namespace: **resource-manifests**

labels:

app: hello-openshift

name: hello-openshift

spec:

**replicas: 2**

selector:

matchLabels:

app: hello-openshift

template:

metadata:

labels:

app: hello-openshift

spec:

containers:

- image: quay.io/redhattraining/hello-world-nginx:v1.0

name: hello-world-nginx

ports:

- **containerPort: 8080**

protocol: TCP

|  |  |
| --- | --- |
|  | Add a namespace attribute to prevent deployment to the wrong project. |
|  | Requires two replicas instead of one. |
|  | Specifies the container port for the service to use. |

You can create a manifest file for each resource that you manage. Alternatively, add each of the manifests to a single multi-part YAML file, and use a --- line to separate the manifests.

---

apiVersion: apps/v1

kind: Deployment

metadata:

namespace: resource-manifests

annotations:

...output omitted...

---

apiVersion: v1

kind: Service

metadata:

namespace: resource-manifests

labels:

app: hello-openshift

name: hello-openshift

spec:

...output omitted...

Using a single file with multiple manifests versus using manifests that are defined in multiple manifest files is a matter of organizational preference. The single file approach has the advantage of keeping together related manifests. With the single file approach, it can be more convenient to change a resource that must be reflected across multiple manifests. In contrast, keeping manifests in multiple files can be more convenient for sharing resource definitions with others.

After creating manifests, you can test them in a non-production environment, or proceed to deploy the manifests. Validate the resource manifests before deploying applications in the production environment.

#### **Declarative Workflows**

Declarative commands use a resource manifest instead of adding the details to many options on the command line. To create a resource, use the kubectl create -f *resource.yaml* command. Instead of a file name, you can pass a directory to the command to process all the resource files in a directory. Add the --recursive=true or -R option to recursively process resource files that are provided in multiple subdirectories.

The following example creates the resources from the manifests in the my-app directory. In this example, the my-app directory contains the example-deployment.yaml and service/example-service.yaml files from previously.

[user@host ~]$ **tree my-app**

my-app

├── example\_deployment.yaml

└── service

└── example\_service.yaml

[user@host ~]$ **kubectl create -R -f ~/my-app**

deployment.apps/hello-openshift created

service/hello-openshift created

The command also accepts a URL:

[user@host ~]$ **kubectl create -f \**

**https://example.com/example-apps/deployment.yaml**

...output omitted...

deployment.apps/hello-openshift created

#### **Updating Resources**

The kubectl apply command can also create resources with the same -f option that is illustrated with the kubectl create command. However, the kubectl apply command can also update a resource.

Updating resources is more complex than creating resources. The kubectl apply command implements several techniques to apply the updates without causing issues.

The kubectl apply command writes the contents of the configuration file to the kubectl.kubernetes.io/last-applied-configuration annotation. The kubectl create command can also generate this annotation by using the --save-config option. The kubectl apply command uses the last-applied-configuration annotation to identify fields that are removed from the configuration file and that must be cleared from the live configuration.

Although the kubectl create -f command can create resources from a manifest, the command is imperative and thus does not account for the current state of a live resource. Executing kubectl create -f against a manifest for a live resource gives an error. In contrast, the kubectl apply -f command is declarative, and considers the difference between the current resource state in the cluster and the intended resource state that is expressed in the manifest.

For example, to update the container's image from version v1.0 to latest, first update the YAML resource manifest to specify the new tag on the image. Then, use the kubectl apply command to instruct Kubernetes to create a version of the deployment resource by using the updated image version that is specified in the manifest.

#### **YAML Validation**

Before applying the changes to the resource, use the --dry-run=server and the --validate=true flags to inspect the file for errors.

* The --dry-run=server option submits a server-side request without persisting the resource.
* The --validate=true option uses a schema to validate the input and fails the request if it is invalid.

Any syntax errors in the YAML are included in the output. Most importantly, the --dry-run=server option prevents applying any changes to the Kubernetes runtime.

[user@host ~]$ **kubectl apply -f ~/my-app/example-deployment.yaml \**

**--dry-run=server --validate=true**

...output omitted...

deployment.apps/hello-openshift created (server dry-run)

|  |  |
| --- | --- |
|  | The output line that ends in (server dry-run) provides the action that the resource file would perform if applied. |

### NOTE

The --dry-run=client option prints only the object that would be sent to the server. The cluster resource controllers can refuse a manifest even if the syntax is valid YAML. In contrast, the --dry-run=server option sends the request to the server to confirm that the manifest conforms to current server policies, without creating resources on the server.

#### **Comparing Resources**

Use the kubectl diff command to review differences between live objects and manifests. When updating resource manifests, you can track differences in the changed files. However, many manifest changes, when applied, do not change the state of the cluster resources. A text-based diff tool would show all such differences, and result in a noisy output.

In contrast, using the kubectl diff command might be more convenient to preview changes. The kubectl diff command emphasizes the significant changes for the Kubernetes cluster. Review the differences to validate that manifest changes have the intended effect.

[user@host ~]$ **kubectl diff -f example-deployment.yaml**

...output omitted...

diff -u -N /tmp/LIVE-2647853521/apps.v1.Deployment.resource...

--- /tmp/LIVE-2647853521/apps.v1.Deployment.resource-manife...

+++ /tmp/MERGED-2640652736/apps.v1.Deployment.resource-mani...

@@ -6,7 +6,7 @@

kubectl.kubernetes.io/last-applied-configuration: |

...output omitted...

creationTimestamp: "2023-04-27T16:07:47Z"

- generation: 1

+ **generation: 2**

labels:

app: hello-openshift

name: hello-openshift

@@ -32,7 +32,7 @@

app: hello-openshift

spec:

containers:

- - image: registry.ocp4.example.com:8443/.../hello-world-nginx:v1.0

+ - **image: registry.ocp4.example.com:8443/.../hello-world-nginx:latest**

imagePullPolicy: IfNotPresent

name: hello-openshift

ports:

|  |  |
| --- | --- |
|  | The line that starts with the - character shows that the current deployment is on generation 1. The following line, which starts with the + character, shows that the generation changes to 2 when the manifest file is applied. |
|  | The image line, which starts with the - character, shows that the current image uses the v1.0 version. The following line, which starts with the + character, shows a version change to latest when the manifest file is applied. |

Kubernetes resource controllers automatically add annotations and attributes to the live resource that make the output of other text-based diff tools misleading, by reporting many differences that have no impact on the resource configuration. Extracting manifests from live resources and making comparisons with tools such as the diff command reports many differences of no value. Using the kubectl diff command confirms that a live resource matches a resource configuration that a manifest provides. GitOps tools depend on the kubectl diff command to determine whether anyone changed resources outside the GitOps workflow. Because the tools themselves cannot know all details about how any controllers might change a resource, the tools defer to the cluster to determine whether a change is meaningful.

#### **Update Considerations**

When using the oc diff command, recognize when applying a manifest change does not generate new pods. For example, if an updated manifest changes only values in secret or a configuration map, then applying the updated manifest does not generate new pods that use those values. Because pods read secret and configuration maps at startup, in this case applying the updated manifest leaves the pods in a vulnerable state, with stale values that are not synchronized with the updated secret or with the configuration map.

As a solution, use the oc rollout restart deployment *deployment-name* command to force a restart of the pods that are associated with the deployment. The forced restart generates pods that use the new values from the updated secret or configuration map.

In deployments with a single replica, you can also resolve the problem by deleting the pod. Kubernetes responds by automatically creating a pod to replace the deleted pod. However, for multiple replicas, using the oc rollout command to restart the pods is preferred, because the pods are stopped and replaced in a smart manner that minimizes downtime.

This course covers other resource management mechanisms that can automate or eliminate some of these challenges.

#### **Applying Changes**

The kubectl create command attempts to create the specified resources in the manifest file. Using the kubectl create command generates an error if the targeted resources are already live in the cluster. In contrast, the kubectl apply command compares three sources to determine how to process the request and to apply changes.

1. The manifest file
2. The live configuration of the resource in the cluster
3. The configuration that is stored in the last-applied-configuration annotation

If the specified resource in the manifest file does not exist, then the kubectl apply command creates the resource. If any fields in the last-applied-configuration annotation of the live resource are not present in the manifest, then the command removes those fields from the live configuration. After applying changes to the live resource, the kubectl apply command updates the last-applied-configuration annotation of the live resource to account for the change.

When creating a resource, the --save-config option of the kubectl create command produces the required annotations for future kubectl apply commands to operate.

### REFERENCES

For more information, refer to the OpenShift CLI Developer Command Reference section in the OpenShift CLI (oc) chapter in the Red Hat OpenShift Container Platform 4.12 CLI Tools documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.12/html-single/cli_tools/index#cli-developer-commands>

For more information, refer to the Using Deployment Strategies section in the Deployments chapter in the Red Hat OpenShift Container Platform 4.12 Building Applications documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.12/html-single/building_applications/index#deployment-strategies>

[Kubernetes Documentation - Replicaset](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/)

[Kubernetes Documentation - Deployment Strategy](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#strategy)

[Kubernetes Documentation - Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)

## **Guided Exercise: Kustomize Overlays**

Deploy and update an application by applying different Kustomize overlays that are stored in a Git server.

**Outcomes**

* Deploy an application by using Kustomize from provided files.
* Apply an application update that changes a deployment.
* Deploy an overlay of the application that increases the number of replicas.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable.

[student@workstation ~]$ **lab start declarative-kustomize**

**Instructions**

1. Clone the v1.1.0 version of the application. Because this repository uses Git branches to represent application versions, you must use the v1.1.0 branch.

Clone the repository from the following URL:

https://git.ocp4.example.com/developer/declarative-kustomize.git

* 1. Change to the ~/DO280/labs/declarative-kustomize directory.

[student@workstation ~]$ **cd DO280/labs/declarative-kustomize**

[student@workstation declarative-kustomize]$

* 1. Clone the initial version of the application.

[student@workstation declarative-kustomize]$ **git clone \**

**https://git.ocp4.example.com/developer/declarative-kustomize.git --branch v1.1.0**

Cloning into 'declarative-kustomize'...

...output omitted...

* 1. Change to the repository directory.

[student@workstation declarative-kustomize]$ **cd declarative-kustomize**

1. Examine the first version of the application.
   1. Use the tree command to review the structure of the repository.

[student@workstation declarative-kustomize]$ **tree**

.

├── base

│   ├── database

│   │   ├── configmap.yaml

│   │   ├── deployment.yaml

│   │   ├── kustomization.yaml

│   │   └── service.yaml

│   ├── exoplanets

│   │   ├── deployment.yaml

│   │   ├── kustomization.yaml

│   │   ├── route.yaml

│   │   └── service.yaml

│   └── kustomization.yaml

└── README.md

3 directories, 10 files

|  |  |
| --- | --- |
|  | The database base defines resources to deploy a database. |
|  | The exoplanets base defines resources to deploy an application that uses the database. |
|  | The repository has a kustomization.yaml file at the root, which uses two other bases. |

* 1. Examine the base/kustomization.yaml file.

[student@workstation declarative-kustomize]$ **cat base/kustomization.yaml**

kind: Kustomization

resources:

- database

- exoplanets

secretGenerator:

- name: db-secrets

literals:

- DB\_ADMIN\_PASSWORD=postgres

- DB\_NAME=database

- DB\_PASSWORD=password

- DB\_USER=user

configMapGenerator:

- name: db-config

literals:

- DB\_HOST=database

- DB\_PORT=5432

|  |  |
| --- | --- |
|  | The base/kustomization.yaml file uses the other two bases. |
|  | The base also uses generators to provide configuration for the two deployments in the application. |

1. Deploy the base directory of the repository to a new declarative-kustomize project. Verify that the v1.1.0 version of the application is available at http://exoplanets-declarative-kustomize.apps.ocp4.example.com.
   1. Log in to the OpenShift cluster as the developer user with the developer password.

[student@workstation declarative-kustomize]$ **oc login -u developer -p developer \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Create the declarative-kustomize project.

[student@workstation declarative-kustomize]$ **oc new-project declarative-kustomize**

...output omitted...

* 1. Use the oc apply -k command to deploy the application with Kustomize.

[student@workstation declarative-kustomize]$ **oc apply -k base**

configmap/database created

configmap/db-config-2d7thbcgkc created

secret/db-secrets-55cbgc8c6m created

service/database created

service/exoplanets created

deployment.apps/database created

deployment.apps/exoplanets created

route.route.openshift.io/exoplanets created

* 1. Use the watch command to wait until the workloads are running.

[student@workstation declarative-kustomize]$ **watch oc get all**

NAME READY STATUS RESTARTS AGE

pod/database-55d6c77787-47649 1/1 Running 0 57s

pod/exoplanets-d6f57869d-jhkhc 1/1 Running 2 (54s ago) 57s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/database ClusterIP 172.30.236.123 <none> 5432/TCP 57s

service/exoplanets ClusterIP 172.30.248.130 <none> 8080/TCP 57s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/database 1/1 1 1 57s

deployment.apps/exoplanets 1/1 1 1 57s

NAME DESIRED CURRENT READY AGE

replicaset.apps/database-55d6c77787 1 1 1 57s

replicaset.apps/exoplanets-d6f57869d 1 1 1 57s

NAME

HOST/PORT ...

route.route.openshift.io/exoplanets

exoplanets-declarative-kustomize.apps.ocp4.example.com ...

Press **Ctrl**+**C** to exit the watch command.

* 1. Open a web browser and navigate to http://exoplanets-declarative-kustomize.apps.ocp4.example.com.

|  |
| --- |
|  |

* 1. The browser displays the v1.1.0 version of the application.

1. Change to the v1.1.1 version of the application and examine the changes.
   1. Change to the v1.1.1 branch.

[student@workstation declarative-kustomize]$ **git checkout v1.1.1**

branch 'v1.1.1' set up to track 'origin/v1.1.1'.

Switched to a new branch 'v1.1.1'

* 1. Use the git show command to display the last commit.

[student@workstation declarative-kustomize]$ **git show**

...output omitted...

diff --git a/base/exoplanets/deployment.yaml b/base/exoplanets/deployment.yaml

index 8bc4cf9..8389b69 100644

--- a/base/exoplanets/deployment.yaml

+++ b/base/exoplanets/deployment.yaml

@@ -23,7 +23,7 @@ spec:

name: exoplanets

- secretRef:

name: exoplanets

**- image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.0**

**+ image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.1**

imagePullPolicy: Always

livenessProbe:

httpGet:

The v1.1.1 version updates the application to the v1.1.1 image.

1. Deploy the updated application and verify that the URL now displays the v1.1.1 version.
   1. Use the oc apply -k command to execute the changes.

[student@workstation declarative-kustomize]$ **oc apply -k base**

...output omitted...

* 1. Use the watch command to wait until the application redeploys.

[student@workstation declarative-kustomize]$ **watch oc get all**

NAME READY STATUS RESTARTS AGE

pod/database-55d6c77787-47649 1/1 Running 0 57s

pod/exoplanets-d6f57869d-jhkhc 1/1 Running 2 (54s ago) 57s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/database ClusterIP 172.30.236.123 <none> 5432/TCP 57s

service/exoplanets ClusterIP 172.30.248.130 <none> 8080/TCP 57s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/database 1/1 1 1 57s

deployment.apps/exoplanets 1/1 1 1 57s

NAME DESIRED CURRENT READY AGE

replicaset.apps/database-55d6c77787 1 1 1 57s

replicaset.apps/exoplanets-d6f57869d 1 1 1 57s

NAME

HOST/PORT ...

route.route.openshift.io/exoplanets

exoplanets-declarative-kustomize.apps.ocp4.example.com ...

Press **Ctrl**+**C** to exit the watch command.

* 1. Open a web browser and navigate to http://exoplanets-declarative-kustomize.apps.ocp4.example.com.

|  |
| --- |
|  |

* 1. The browser displays the v1.1.1 version of the application.

1. Change to the v1.1.2 version of the application and examine the changes.
   1. Change to the v1.1.2 branch.

[student@workstation declarative-kustomize]$ **git checkout v1.1.2**

branch 'v1.1.2' set up to track 'origin/v1.1.2'.

Switched to a new branch 'v1.1.2'

* 1. Use the git show command to display the last commit.

[student@workstation declarative-kustomize]$ **git show**

...output omitted...

diff --git a/base/kustomization.yaml b/base/kustomization.yaml

index fdf129a..8de16e8 100644

--- a/base/kustomization.yaml

+++ b/base/kustomization.yaml

@@ -7,7 +7,7 @@ secretGenerator:

literals:

- DB\_ADMIN\_PASSWORD=postgres

- DB\_NAME=database

**- - DB\_PASSWORD=password**

**+ - DB\_PASSWORD=newpassword**

- DB\_USER=user

configMapGenerator:

- name: db-config

The v1.1.2 version updates the base kustomization. This update changes the password that the database uses. This change is possible because the sample application re-creates the database on startup.

* 1. List the secrets in the namespace.

[student@workstation declarative-kustomize]$ **oc get secret**

NAME TYPE DATA AGE

builder-dockercfg-qwn4v kubernetes.io/dockercfg 1 4m31s

builder-token-z754n kubernetes.io/service-account-token 4 4m31s

**db-secrets-55cbgc8c6m Opaque 4 4m28s**

default-dockercfg-w4v89 kubernetes.io/dockercfg 1 4m31s

default-token-zw89c kubernetes.io/service-account-token 4 4m31s

deployer-dockercfg-l8sct kubernetes.io/dockercfg 1 4m31s

deployer-token-knvhb kubernetes.io/service-account-token 4 4m31s

When creating a secret, Kustomize appends a hash to the secret name.

* 1. Extract the contents of the secret. The name of the secret can change in your environment. Use the output from a previous step to learn the name of the secret.

[student@workstation declarative-kustomize]$ **oc extract \**

**secret/db-secrets-*55cbgc8c6m* --to=-**

# DB\_PASSWORD

password

# DB\_USER

user

# DB\_ADMIN\_PASSWORD

postgres

# DB\_NAME

database

1. Deploy the updated application.
   1. Use the oc apply -k command to execute the changes.

[student@workstation declarative-kustomize]$ **oc apply -k base**

configmap/database unchanged

configmap/db-config-2d7thbcgkc unchanged

**secret/db-secrets-6h668tk789 created**

service/database unchanged

service/exoplanets unchanged

**deployment.apps/database configured**

**deployment.apps/exoplanets configured**

route.route.openshift.io/exoplanets configured

Because the password is different, Kustomize creates another secret. Kustomize also updates the two deployments that use the secret to use the new secret.

* 1. Use the watch command to wait until the application redeploys.

[student@workstation declarative-kustomize]$ **watch oc get all**

NAME READY STATUS RESTARTS AGE

pod/database-55d6c77787-47649 1/1 Running 0 57s

pod/exoplanets-d6f57869d-jhkhc 1/1 Running 2 (54s ago) 57s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/database ClusterIP 172.30.236.123 <none> 5432/TCP 57s

service/exoplanets ClusterIP 172.30.248.130 <none> 8080/TCP 57s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/database 1/1 1 1 57s

deployment.apps/exoplanets 1/1 1 1 57s

NAME DESIRED CURRENT READY AGE

replicaset.apps/database-55d6c77787 1 1 1 57s

replicaset.apps/exoplanets-d6f57869d 1 1 1 57s

NAME

HOST/PORT ...

route.route.openshift.io/exoplanets

exoplanets-declarative-kustomize.apps.ocp4.example.com ...

Press **Ctrl**+**C** to exit the watch command.

* 1. Open a web browser and navigate to http://exoplanets-declarative-kustomize.apps.ocp4.example.com.

|  |
| --- |
|  |

* 1. The browser continues showing the v1.1.1 version of the application.
  2. Examine the deployment.

[student@workstation declarative-kustomize]$ **oc get deployment exoplanets \**

**-o jsonpath='{.spec.template.spec.containers[0].envFrom}{"\n"}'**

[{"configMapRef":{"name":"db-config-2d7thbcgkc"}},{"secretRef":{"name":"db-secrets-**6h668tk789**"}}]

The deployment uses the new secret.

* 1. Examine the secret. Use the name of the secret from a previous step.

[student@workstation declarative-kustomize]$ **oc extract \**

**secret/db-secrets-*6h668tk789* --to=-**

# DB\_ADMIN\_PASSWORD

postgres

# DB\_NAME

database

# DB\_PASSWORD

**newpassword**

# DB\_USER

user

The deployment uses the changed password.

1. Change to the v1.1.3 version of the application and examine the changes.
   1. Change to the v1.1.3 branch.

[student@workstation declarative-kustomize]$ **git checkout v1.1.3**

branch 'v1.1.3' set up to track 'origin/v1.1.3'.

Switched to a new branch 'v1.1.3'

Use the git show command to display the last commit.

[student@workstation declarative-kustomize]$ **git show**

...output omitted...

diff --git a/overlays/production/kustomization.yaml b/overlays/production/kustomization.yaml

new file mode 100644

index 0000000..73bb7fe

--- /dev/null

+++ b/overlays/production/kustomization.yaml

@@ -0,0 +1,8 @@

+kind: Kustomization

+resources:

+- ../../base/

+patches:

+- path: patch-replicas.yaml

+ target:

+ kind: Deployment

+ name: exoplanets

diff --git a/overlays/production/patch-replicas.yaml b/overlays/production/patch-replicas.yaml

new file mode 100644

index 0000000..a025aa0

--- /dev/null

+++ b/overlays/production/patch-replicas.yaml

@@ -0,0 +1,6 @@

+apiVersion: apps/v1

+kind: Deployment

+metadata:

+ name: exoplanets

+spec:

+ replicas: 2

The v1.1.3 version adds a production overlay that increases the number of replicas.

1. Deploy the updated application and verify the number of replicas.
   1. Use the oc apply -k command to execute the changes.
   2. [student@workstation declarative-kustomize]$ **oc apply -k overlays/production**

...output omitted...

* 1. Use the watch command to wait until the application redeploys.

[student@workstation declarative-kustomize]$ **watch oc get all**

NAME READY STATUS RESTARTS AGE

pod/database-7dfb559cf7-rvxhx 1/1 Running 0 11m

pod/exoplanets-957bb5b48-5xl2d 1/1 Running 2 (11m ago) 11m

pod/exoplanets-957bb5b48-mgbrx 1/1 Running 0 19s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/database ClusterIP 172.30.87.214 <none> 5432/TCP 19m

service/exoplanets ClusterIP 172.30.25.65 <none> 8080/TCP 19m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/database 1/1 1 1 19m

deployment.apps/exoplanets 2/2 2 2 19m

NAME DESIRED CURRENT READY AGE

replicaset.apps/database-7dfb559cf7 1 1 1 11m

replicaset.apps/database-d4cd8dcc 0 0 0 19m

replicaset.apps/exoplanets-6c7b4bb44c 0 0 0 19m

replicaset.apps/exoplanets-7ccb754c8b 0 0 0 18m

replicaset.apps/exoplanets-957bb5b48 2 2 2 11m

NAME

HOST/PORT ...

route.route.openshift.io/exoplanets

exoplanets-declarative-kustomize.apps.ocp4.example.com ...

Press **Ctrl**+**C** to exit the watch command. After you run the command, the application has two replicas.

1. Delete the application.
   1. Use the oc delete -k command to delete the resources that Kustomize manages.

[student@workstation declarative-kustomize]$ **oc delete -k base**

# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize edit fix' to update your Kustomization automatically.

configmap "database" deleted

configmap "db-config-2d7thbcgkc" deleted

secret "db-secrets-h9hdmt2g79" deleted

service "database" deleted

service "exoplanets" deleted

deployment.apps "database" deleted

deployment.apps "exoplanets" deleted

route.route.openshift.io "exoplanets" deleted

* 1. Change to the home directory.
  2. [student@workstation declarative-kustomize]$ **cd**

[student@workstation ~]$

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish declarative-kustomize**

# Chapter 2.  Deploy Packaged Applications

[OpenShift Templates](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02)

[Guided Exercise: OpenShift Templates](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02s02)

[Helm Charts](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02s03)

[Guided Exercise: Helm Charts](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02s04)

[Lab: Deploy Packaged Applications](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02s05)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch02s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Deploy and update applications from resource manifests that are packaged for sharing and distribution. |
| **Objectives** | * Deploy an application and its dependencies from resource manifests that are stored in an OpenShift template. * Deploy and update applications from resource manifests that are packaged as Helm charts. |
| **Sections** | * OpenShift Templates (and Guided Exercise) * Helm Charts (and Guided Exercise) |
| **Lab** | * Deploy Packaged Applications |

## **OpenShift Templates**

### **Objectives**

* Deploy and update applications from resource manifests that are packaged as OpenShift templates.

### **OpenShift Templates**

A template is a Kubernetes custom resource that describes a set of Kubernetes resource configurations. Templates can have parameters. You can create a set of related Kubernetes resources from a template by processing the template, and providing values for the parameters. Templates have varied use cases, and can create any Kubernetes resource. You can create a list of resources from a template by using the CLI or, if a template is uploaded to your project or to the global template library, by using the web console.

The template resource is a Kubernetes extension that Red Hat for OpenShift provides. The Cluster Samples Operator populates templates (and image streams) in the openshift namespace. You can opt out of adding templates during installation, and you can restrict the list of templates that the operator populates.

You can also create templates from scratch, or copy and customize a template to suit the needs of your project.

#### **Discovering Templates**

The templates that the Cluster Samples Operator provides are in the openshift namespace. Use the following oc get command to view a list of these templates:

[user@host ~]$ **oc get templates -n openshift**

NAME DESCRIPTION PARAMETERS OBJECTS

cache-service Red Hat Data Grid... 8 (1 blank) 4

cakephp-mysql-example An example CakePHP... 21 (4 blank) 8

cakephp-mysql-persistent An example CakePHP... 22 (4 blank) 9

...output omitted...

To evaluate any template, use the oc describe template *template-name* -n openshift command to view more details about the template, including the description, the labels that the template uses, the template parameters, and the resources that the template generates.

The following example shows the details of the cache-service template:

[user@host ~]$ **oc describe template cache-service -n openshift**

Name: cache-service

Namespace: openshift

Created: 2 months ago

Labels: samples.operator.openshift.io/managed=true

template=cache-service

Description: Red Hat Data Grid is an in-memory, distributed key/value store.

Annotations: iconClass=icon-datagrid

...output omitted...

Parameters:

Name: APPLICATION\_NAME

Display Name: Application Name

Description: Specifies a name for the application.

Required: true

Value: cache-service

...output omitted...

Name: APPLICATION\_PASSWORD

Display Name: Client Password

Description: Sets a password to authenticate client applications.

Required: false

Generated: expression

From: [a-zA-Z0-9]{16}

Object Labels: template=cache-service

Message: <none>

Objects:

Secret ${APPLICATION\_NAME}

Service ${APPLICATION\_NAME}-ping

Service ${APPLICATION\_NAME}

StatefulSet.apps ${APPLICATION\_NAME}

|  |  |
| --- | --- |
|  | Use the description to determine the purpose of the template. |
|  | The parameters provide deployment flexibility. |
|  | The value field provides a default value that you can override. |
|  | The Generated and From fields also generate default values. |
|  | The object labels are applied to all resources that the template creates. |
|  | The objects section lists the resources that the template creates. |

In addition to using the oc describe command to view information about a template, the oc process command provides a --parameters option to view only the parameters that a template uses. For example, use the following command to view the parameters that the cache-service template uses:

[user@host ~]$ **oc process --parameters cache-service -n openshift**

NAME ... GENERATOR VALUE

APPLICATION\_NAME ... cache-service

IMAGE ... registry.redhat.io/jboss-datagrid-7/...

NUMBER\_OF\_INSTANCES ... 1

REPLICATION\_FACTOR ... 1

EVICTION\_POLICY ... evict

TOTAL\_CONTAINER\_MEM ... 512

APPLICATION\_USER ...

APPLICATION\_PASSWORD ... expression [a-zA-Z0-9]{16}

Use the -f option to view the parameters of a template that are defined in a file:

[user@host ~]$ **oc process --parameters -f my-cache-service.yaml**

Use the oc get template *template-name* -o yaml -n *namespace* command to view the manifest for the template. The following example retrieves the template manifest for the cache-service template:

[user@host ~]$ **oc get template cache-service -o yaml -n openshift**

apiVersion: template.openshift.io/v1

kind: **Template**

labels:

template: cache-service

metadata:

...output omitted...

- apiVersion: v1

kind: **Secret**

metadata:

...output omitted...

- apiVersion: v1

kind: **Service**

metadata:

...output omitted...

- apiVersion: v1

kind: **Service**

metadata:

...output omitted...

- apiVersion: apps/v1

kind: **StatefulSet**

metadata:

...output omitted...

**parameters**:

- description: Specifies a name for the application.

displayName: Application Name

name: APPLICATION\_NAME

required: true

value: cache-service

- description: Sets an image to bootstrap the service.

name: IMAGE

...output omitted...

In the template manifest, examine how the template creates resources. The manifest is also a good resource for learning how to create your own templates.

#### **Using Templates**

The oc new-app command has a --template option that can deploy the template resources directly from the openshift project. The following example deploys the resources that are defined in the cache-service template from the openshift project:

[user@host ~]$ **oc new-app --template=cache-service -p APPLICATION\_USER=my-user**

Using the oc new-app command to deploy the template resources is convenient for development and testing. However, for production usage, consume templates in a manner that helps resource and configuration tracking. For example, the oc new-app command can only create new resources, not update existing resources.

You can use the oc process command to apply parameters to a template, to produce manifests to deploy the templates with a set of parameters. The oc process command can process both templates that are stored in files locally, and templates that are stored in the cluster. However, to process templates in a namespace, you must have write permissions on the template namespace. For example, to run oc process on the templates in the openshift namespace, you must have write permissions on this namespace.

### NOTE

Unprivileged users can read the templates in the openshift namespace by default. Those users can extract the template from the openshift namespace and create a copy in a project where they have wider permissions. By copying a template to a project, they can use the oc process command on the template.

#### **Deploying Applications from Templates**

The oc process command uses parameter values to transform a template into a set of related Kubernetes resource manifests. For example, the following command creates a set of resource manifests for the my-cache-service template. When you use the -o yaml option, the resulting manifests are in the YAML format. The example writes the manifests to a my-cache-service-manifest.yaml file:

[user@host ~]$ **oc process my-cache-service \**

**-p APPLICATION\_USER=user1 -o yaml > my-cache-service-manifest.yaml**

The previous example uses the -p option to provide a parameter value to the only required parameter without a default value.

Use the -f option with the oc process command to process a template that is defined in a file:

[user@host ~]$ **oc process -f my-cache-service.yaml \**

**-p APPLICATION\_USER=user1 -o yaml > my-cache-service-manifest.yaml**

Use the -p option with *key*=*value* pairs with the oc process command to use parameter values that override the default values. The following example passes three parameter values to the my-cache-service template, and overrides the default values of the specified parameters:

[user@host ~]$ **oc process my-cache-service -o yaml \**

**-p TOTAL\_CONTAINER\_MEM=1024 \**

**-p APPLICATION\_USER='cache-user' \**

**-p APPLICATION\_PASSWORD='my-secret-password' \**

**> my-cache-service-manifest.yaml**

Instead of specifying parameters on the command line, place the parameters in a file. This option cleans up the command line when many parameter values are required. Save the parameters file in a version control system to keep records of the parameters that are used in production deployments.

For example, instead of using the command-line options in the previous examples, place the key-value pairs in a my-cache-service-params.env file. Add the key-value pairs to the file, with each pair on a separate line:

TOTAL\_CONTAINER\_MEM=1024

APPLICATION\_USER='cache-user'

APPLICATION\_PASSWORD='my-secret-password'

The corresponding oc process command uses the --param-file option to pass the parameters as follows:

[user@host ~]$ **oc process my-cache-service -o yaml \**

**--param-file=my-cache-service-params.env > my-cache-service-manifest.yaml**

Generating a manifest file is not required to use templates. Instead, pipe the output of the oc process command directly to the input for the oc apply -f - command. The oc apply command creates live resources on the Kubernetes cluster.

[user@host ~]$ **oc process my-cache-service \**

**--param-file=my-cache-service-params.env | oc apply -f -**

Because templates are flexible, you can use the same template to create different resources by changing the input parameters.

#### **Updating Apps from Templates**

Because you use the oc apply command, after deploying a set of manifests from a template, you can process the template again and use oc apply for updates. This procedure can make simple changes to deployed templates, such as changing a parameter. However, many workload updates are not possible with this mechanism. To manage more complex applications, consider using other mechanisms such as Helm charts, which are described elsewhere in this course.

To compare the results of applying a different parameters file to a template against the live resources, pipe the manifest to the oc diff -f - command. For example, given a second parameter file named my-cache-service-params-2.env, use the following command:

[user@host ~]$ **oc process my-cache-service -o yaml \**

**--param-file=my-cache-service-params-2.env | oc diff -f -**

...output omitted...

- generation: 1

+ generation: 2

labels:

application: cache-service

template: cache-service

@@ -86,10 +86,10 @@

timeoutSeconds: 10

resources:

limits:

- memory: 1Gi

+ memory: 2Gi

requests:

cpu: 500m

- memory: 1Gi

+ memory: 2Gi

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

In this case, the configuration change increases the memory usage of the application. The output shows that the second generation uses 2Gi of memory instead of 1Gi.

After verifying that the changes are what you intend, you can pipe the output of the oc process to the oc apply -f - command.

#### **Managing Templates**

For production usage, make a customized copy of the template, to change the default values of the template to suitable values for the target project. To copy a template into your project, use the oc get template command with the -o yaml option to copy the template YAML to a file.

The following example copies the cache-service template from the openshift project to a YAML file named my-cache-service.yaml:

[user@host ~]$ **oc get template cache-service -o yaml \**

**-n openshift > my-cache-service.yaml**

After creating a YAML file for a template, consider making the following changes to the template:

* Give the template a new name that is specific to the target use of the template resources.
* Apply appropriate changes to the parameter default values at the end of the file.
* Remove the namespace field of the template resource.

You can process templates in other namespaces, if you can create the processed template resource in those namespaces. Processing the template in a different project without changing the template namespace to match the target namespace gives an error. Optionally, you can also delete the namespace field from the metadata field of the template resource.

After you have a YAML file for a template, use the oc create -f command to upload the template to the current project. In this case, the oc create command is not creating the resources that the template defines. Instead, the command is creating a template resource in the project. Using a template that is uploaded to a project clarifies which template provides the resource definitions of a project. After uploading, the template is available to anyone with access to the project.

The following example uploads a customized template that is defined in the my-cache-service.yaml file to the current project:

[user@host ~]$ **oc create -f my-cache-service.yaml**

Use the -n *namespace* option to upload the template to a different project. The following example uploads the template that is defined in the my-cache-service.yaml file to the shared-templates project:

[user@host ~]$ **oc create -f my-cache-service.yaml -n shared-templates**

Use the oc get templates command to view a list of available templates in the project:

[user@host ~]$ **oc get templates -n shared-templates**

NAME DESCRIPTION PARAMETERS OBJECTS

my-cache-service Red Hat Data Grid... 8 (1 blank) 4

### REFERENCES

For more information, refer to the Understanding Templates section in the Using Templates chapter in the Red Hat OpenShift Container Platform 4.12 Images documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.12/html-single/images/index#templates-overview\_using-templates](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.12/html-single/images/index" \l "templates-overview_using-templates" \t "_top)

For more information, refer to the OpenShift CLI Developer Command Reference section in the OpenShift CLI (oc) chapter in the Red Hat OpenShift Container Platform 4.12 CLI Tools documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.12/html-single/cli\_tools/index#cli-developer-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.12/html-single/cli_tools/index" \l "cli-developer-commands" \t "_top)

[Kubernetes Documentation - kubectl Commands](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands" \t "_top)

## **Guided Exercise: OpenShift Templates**

Deploy and update an application from a template that is stored in another project.

**Outcomes**

* Deploy and update an application from a template.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise.

[student@workstation ~]$ **lab start packaged-templates**

**Instructions**

1. Log in to the OpenShift cluster as the developer user with the developer password.
   1. Log in to the OpenShift cluster.

[student@workstation ~]$ **oc login -u developer -p developer \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

1. Examine the available templates in the cluster, in the openshift project. Identify an appropriate template to deploy a MySQL database.
   1. Use the get command to retrieve a list of templates in the cluster, in the openshift project.

[student@workstation ~]$ **oc get templates -n openshift**

NAME DESCRIPTION PARAMETERS OBJECTS

...output omitted...

mysql-ephemeral MySQL database... 8 (3 generated) 3

**mysql-persistent** MySQL database... 9 (3 generated) 4

...output omitted...

* 1. Use the oc process --parameters command to view the parameters of the mysql-persistent template.

[student@workstation ~]$ **oc process --parameters mysql-persistent \**

**-n openshift**

NAME DESCRIPTION GENERATOR VALUE

MEMORY\_LIMIT ... 512Mi

NAMESPACE ... openshift

DATABASE\_SERVICE\_NAME ... mysql

MYSQL\_USER ... expression user[A-Z0-9]{3}

MYSQL\_PASSWORD ... expression [a-zA-Z0-9]{16}

MYSQL\_ROOT\_PASSWORD ... expression [a-zA-Z0-9]{16}

MYSQL\_DATABASE ... sampledb

VOLUME\_CAPACITY ... 1Gi

MYSQL\_VERSION ... 8.0-el8

All the required parameters have either default values or generated values.

1. Use the mysql-persistent template to deploy a database by processing the template.
   1. Create the packaged-templates project.
   2. [student@workstation ~]$ **oc new-project packaged-templates**
   3. Now using project "packaged-templates" on server ...

...output omitted...

* 1. Use the oc new-app command to deploy the application.

[student@workstation ~]$ **oc new-app --template=mysql-persistent \**

**-p MYSQL\_USER=user1 \**

**-p MYSQL\_PASSWORD=mypasswd**

--> Deploying template "packaged-templates/mysql-persistent" to project packaged-templates

...output omitted...

--> Success

Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:

'oc expose service/mysql'

Run 'oc status' to view your app.

* 1. Use the watch command to verify that the pods are running. Wait for the mysql-1-deploy pod to show a Completed status. Press **Ctrl**+**C** to exit the watch command.

[student@workstation ~]$ **watch oc get pods**

NAME READY STATUS RESTARTS AGE

mysql-1-5t8h8 1/1 Running 0 83s

mysql-1-deploy 0/1 **Completed** 0 84s

* 1. Connect to the database to verify that it is working.

[student@workstation ~]$ **oc run query-db -it --rm \**

**--image registry.ocp4.example.com:8443/rhel8/mysql-80 \**

**--restart Never --command -- \**

**/bin/bash -c \**

**"mysql -uuser1 -pmypasswd --protocol tcp \**

**-h mysql -P3306 sampledb -e 'SHOW DATABASES;'"**

mysql: [Warning] Using a password on the command line interface can be insecure.

+--------------------+

| Database |

+--------------------+

| information\_schema |

| performance\_schema |

| sampledb |

+--------------------+

pod "query-db" deleted

1. Deploy the application from the custom template, in the ~/DO280/labs/packaged-templates/custom-template/roster-template.yaml file, to the project. The application initializes and uses the database that the mysql-persistent template deployed.
   1. Upload the custom template to the project.

[student@workstation ~]$ **oc create -f \**

**~/DO280/labs/packaged-templates/custom-template/roster-template.yaml**

template.template.openshift.io/roster-template created

* 1. Use oc get templates to view the available templates in the packaged-templates project.

[student@workstation ~]$ **oc get templates**

NAME DESCRIPTION PARAMETERS OBJECTS

roster-template Example application for DO280... 8 (2 blank) 4

* 1. Use the oc process --parameters command to view the parameter of the roster-template template.

[student@workstation ~]$ **oc process --parameters roster-template**

NAME DESCRIPTION GENERATOR VALUE

IMAGE ... registry.../do280-roster:v1

APPNAME ... do280-roster

NAMESPACE ... packaged-templates

DATABASE\_SERVICE\_NAME ... mysql

MYSQL\_USER ...

MYSQL\_PASSWORD ...

MYSQL\_DATABASE ... sampledb

INIT\_DB ... False

* 1. Use the oc process command to generate the manifests for the roster-template application resources, and use the oc apply command to create the resources in the Kubernetes cluster.

You must use the same database credentials that you used in an earlier step to configure the database, so that the application can access the database.

[student@workstation ~]$ **oc process roster-template \**

**-p MYSQL\_USER=user1 -p MYSQL\_PASSWORD=mypasswd -p INIT\_DB=true | oc apply -f -**

secret/mysql configured

deployment.apps/do280-roster created

service/do280-roster created

route.route.openshift.io/do280-roster created

* 1. Use the oc get pods command to confirm that the application is running.

[student@workstation ~]$ **oc get pods**

NAME READY STATUS RESTARTS AGE

do280-roster-c7f596dd8-pqvlv 1/1 **Running** 0 60s

mysql-1-bl97v 1/1 Running 0 33m

mysql-1-deploy 0/1 Completed 0 33m

* 1. Use the oc get routes command to view the routes.

[student@workstation ~]$ **oc get routes**

NAME HOST/PORT ...

do280-roster **do280-roster-packaged-templates.apps.ocp4.example.com** ...

* 1. Open the application URL in the web browser. The header confirms the use of **version 1** of the application.

http://do280-roster-packaged-templates.apps.ocp4.example.com

* 1. Enter your information in the form and save it to the database.

1. Deploy an updated version of the do280/roster application from the custom template in the roster-template template. Use version 2 of the application and do not overwrite the data in the database.
   1. Create a text file named roster-parameters.env with the following content:
   2. MYSQL\_USER=user1
   3. MYSQL\_PASSWORD=mypasswd

IMAGE=registry.ocp4.example.com:8443/redhattraining/do280-roster:v2

The option of using a parameter file helps version control software to track changes.

* 1. Use the oc process command and the oc diff command to view the changes in the new manifests when compared to the live application.

[student@workstation ~]$ **oc process roster-template \**

**--param-file=roster-parameters.env | oc diff -f -**

diff -u -N ...output omitted...

--- /tmp/LIVE-1948327112/apps.v1.Deployment.packaged-templates...

+++ /tmp/MERGED-2797490080/apps.v1.Deployment.packaged-templates...

...output omitted...

key: database-service

name: mysql

- name: INIT\_DB

- value: "true"

- image: registry.ocp4.example.com:8443/redhattraining/do280-roster:v1

+ **value: "False"**

+ **image: registry.ocp4.example.com:8443/redhattraining/do280-roster:v2**

imagePullPolicy: IfNotPresent

name: do280-roster-image

ports:

|  |  |
| --- | --- |
|  | The INIT\_DB environment variable determines whether the application initializes the database. The default False value is used when the parameter is omitted. In the first deployment, the INIT\_DB variable was set to the True value, so the database was initialized. In this second deployment, the deployment does not have to initialize the database again. |
|  | The IMAGE parameter changes the image that the template uses. |

* 1. Use the oc process command to generate the manifests for the roster-template application objects, and use the oc apply command to create the application objects. With the changes from a previous step, you use the IMAGE variable to use a different image for the update and omit the INIT\_DB variable.

[student@workstation ~]$ **oc process roster-template \**

**--param-file=roster-parameters.env | oc apply -f -**

secret/mysql configured

deployment.apps/do280-roster configured

service/do280-roster unchanged

route.route.openshift.io/do280-roster unchanged

* 1. Use watch to verify that the pods are running. Wait for the mysql-1-deploy pod to show a Completed status. Press **Ctrl**+**C** to exit the watch command.

[student@workstation ~]$ **watch oc get pods**

NAME READY STATUS RESTARTS AGE

do280-roster-c7f596dd8-ktlvl 1/1 **Running** 0 60s

mysql-1-bl97v 1/1 Running 0 53m

mysql-1-deploy 0/1 Completed 0 53m

* 1. Open the application URL in the web browser. The route is unchanged, so you can refresh the previous browser page if the page is still open. The header confirms the use of **version 2** of the application. The data that is pulled from the database is unchanged.

http://do280-roster-packaged-templates.apps.ocp4.example.com

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish packaged-templates**

## **Helm Charts**

### **Objectives**

* Deploy and update applications from resource manifests that are packaged as Helm charts.

### **Helm**

Helm is an open source application that helps to manage the lifecycle of Kubernetes applications.

Helm introduces the concept of charts. A chart is a package that describes a set of Kubernetes resources that you can deploy. Helm charts define values that you can customize when deploying an application. Helm includes functions to distribute charts and updates.

Many organizations distribute Helm charts to deploy applications. Often, Helm is the supported mechanism to deploy a specific application.

However, Helm does not cover all needs to manage certain kinds of applications. Operators have a more complete model that can handle the lifecycle of more complex applications. For more details about operators, refer to [the section called “ Kubernetes Operators and the Operator Lifecycle Manager ”](https://rol.redhat.com/rol/app/courses/do280-4.12/pages/ch07).

### **Helm Charts**

A Helm chart defines Kubernetes resources that you can deploy. A chart is a collection of files with a defined structure. These files include chart metadata (such as the chart name or version), resource definitions, and supporting material.

Chart authors can use the template feature of the Go language for the resource definitions. For example, instead of specifying the image for a deployment, charts can use user-provided values for the image. By using values to choose an image, cluster administrators can replace a default public image with an image from a private repository.

The following diagram shows the structure of a minimal Helm chart:

sample/

├── Chart.yaml

├── templates

| |── example.yaml

└── values.yaml

|  |  |
| --- | --- |
|  | The Chart.yaml file contains chart metadata, such as the name and version of the chart. |
|  | The templates directory contains files that define application resources such as deployments. |
|  | The values.yaml file contains default values for the chart. |

Helm charts can contain hooks that Helm executes at different points during installations and upgrades. Hooks can automate tasks for installations and upgrades. With hooks, Helm charts can manage more complex applications than purely manifest-based processes. Review the chart documentation to learn about the chart hooks and their implications.

### **Using Helm Charts**

Helm is a command-line application. The helm command interacts with the following entities:

**Charts**

Charts are the packaged applications that the helm command deploys.

**Releases**

A release is the result of deploying a chart. You can deploy a chart many times to the same cluster. Each deployment is a different release.

**Versions**

A Helm chart can have many versions. Chart authors can release updates to charts, to adapt to later application versions, introduce new features, or fix issues.

You can use and refer to charts in various ways. For example, if your local file system contains a chart, then you can refer to that chart by using the path to the chart directory. You can also use a path or a URL that contains a chart that is packaged in a tar archive with gzip compression.

#### **Inspecting Helm Charts**

Use the helm show command to display information about a chart. The show chart subcommand displays general information, such as the maintainers, or the source URL.

[user@host ~]$ **helm show chart *chart-reference***

apiVersion: v1

description: A Helm chart for Kubernetes

name: examplechart

version: 0.1.0

maintainers:

- email: dev@example.com

name: Developer

sources:

- https://git.example.com/examplechart

The show values subcommand displays the default values for the chart. The output is in YAML format and comes from the values.yaml file in the chart.

[user@host ~]$ **helm show values *chart-reference***

image:

repository: "sample"

tag: "1.8.10"

pullPolicy: IfNotPresent

...output omitted...

Chart resources use the values from the values.yaml file by default. You can override these default values. You can use the output of the show values command to discover customizable values.

#### **Installing Helm Charts**

After inspecting the chart, you can deploy the resources in the chart by using the helm install command. In Helm, install refers to deploying the resources in a chart to create a release.

Always refer to the documentation of the chart before installation to learn about prerequisites, extra installation steps, and other information.

To install a chart, you must decide on the following parameters:

* The deployment target namespace
* The values to override
* The release name

Helm charts can contain Kubernetes resources of any kind. These resources can be namespaced or non-namespaced. Like normal resource definitions, namespaced resources in charts can define or omit a namespace declaration.

Most Helm charts that deploy applications do not create a namespace, and namespaced resources in the chart omit a namespace declaration. Typically, when deploying a chart that follows this structure, you create a namespace for the deployment, and Helm creates namespaced resources in this namespace.

After deciding the target namespace, you can design the values to use. Inspect the documentation and the output of the helm show values command to decide which values to override.

You can define values by writing a YAML file that contains them. This file can follow the structure from the output of the helm show values command, which contains the default values. Specify only the values to override.

Consider the following output from the helm show values command for an example chart:

image:

repository: "sample"

tag: "1.8.10"

pullPolicy: IfNotPresent

Create a values.yaml file without the image key if you do not want to override any image parameters. Omit the pullPolicy key to override the tag key but not the pull policy. For example, the following YAML file would override only the image tag:

image:

tag: "1.8.10-patched"

Besides the YAML file, you can override specific values by using command-line arguments.

The final element to prepare a chart deployment is choosing a release name. You can deploy a chart many times to a cluster. Each chart deployment must have a unique release name for identification purposes. Many Helm charts use the release name to construct the name of the created resources.

With the namespace, values, and release name, you can start the deployment process. The helm install command creates a release in a namespace, with a set of values.

#### **Rendering Manifests from a Chart**

You can use the --dry-run option to preview the effects of installing a chart.

[user@host ~]$ **helm install *release-name* *chart-reference* --dry-run \**

**--values values.yaml**

NAME: release-name

LAST DEPLOYED: Tue May 30 13:14:57 2023

NAMESPACE: current-namespace

STATUS: pending-install

REVISION: 1

TEST SUITE: None

HOOKS:

MANIFEST:

---

# Source: chart/templates/serviceaccount.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: my-release-sa

labels:

...output omitted...

NOTES:

The application can be accessed via port 1234.

...output omitted...

|  |  |
| --- | --- |
|  | General information about the new release |
|  | A list of the resources that the helm install command would create |
|  | Additional information |

### NOTE

You define values to use for the installation with the --values values.yaml option. In this file, you override the default values from the chart that are defined in the values.yaml file that the chart contains.

Often, chart resource names include the release name. In the example output of the helm install command, the service account is a combination of the release name and the -sa text.

Chart authors can provide installation notes that use the chart values. In the same example, the port number in the notes reflects a value from the values.yaml file.

If the preview looks correct, then you can run the same command without the --dry-run option to deploy the resources and create the release.

### **Releases**

When the helm install command runs successfully, besides creating the resources, Helm creates a release. Helm stores information about the release as a secret of the helm.sh/release.v1 type.

#### **Inspecting Releases**

Use the helm list command to inspect releases on a cluster.

[user@host ~]$ **helm list**

NAME NAMESPACE REVISION ... STATUS CHART APP VERSION

my-release example 1 ... deployed example-4.12.1 1.8.10

Similarly to kubectl commands, many helm commands have the --all-namespaces and --namespace options. The helm list command without options lists releases in the current namespace. If you use the --all-namespaces option, then it lists releases in all namespaces. If you use the --namespace option, then it lists releases in a single namespace.

### WARNING

Do not manipulate the release secret. If you remove the secret, then Helm cannot operate with the release.

#### **Upgrading Releases**

The helm upgrade command can apply changes to existing releases, such as updating values or the chart version.

### IMPORTANT

By default, this command automatically updates releases to use the latest version of the chart.

The helm upgrade command uses similar arguments and options to the helm install command. However, the helm upgrade command interacts with existing resources in the cluster instead of creating resources from a blank state. Therefore, the helm upgrade command can have more complex effects, such as conflicting changes. Always review the chart documentation when using a later version of a chart, and when changing values. You can use the --dry-run option to preview the manifests that the helm upgrade command uses, and compare them to the running resources.

#### **Rolling Back Helm Upgrades**

Helm keeps a log of release upgrades, to review changes and roll back to previous releases.

You can review this log by using the helm history command:

[user@host ~]$ **helm history *release\_name***

REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION

1 Wed May 31... superseded chart-0.0.6 latest Install complete

2 Wed May 31... deployed chart-0.0.7 latest Upgrade complete

You can use the helm rollback command to revert to an earlier revision:

[user@host ~]$ **helm rollback *release\_name* *revision***

Rollback was a success! Happy Helming!

Rolling back can have greater implications than upgrading, because upgrades might not be reversible. If you keep a test environment with the same upgrades as a production environment, then you can test rollbacks before performing them in the production environment to find potential issues.

### **Helm Repositories**

Charts can be distributed as files, archives, or container images, or by using chart repositories.

The helm repo command provides the following subcommands to work with chart repositories.

| **Subcommand** | **Description** |
| --- | --- |
| add *NAME* *REPOSITORY\_URL* | Add a Helm chart repository. |
| list | List Helm chart repositories. |
| update | Update Helm chart repositories. |
| remove *REPOSITORY1\_NAME* *REPOSITORY2\_NAME* …​ | Remove Helm chart repositories. |

The following command adds a repository:

[user@host ~]$ **helm repo add \**

**openshift-helm-charts https://charts.openshift.io/**

"openshift-helm-charts" has been added to your repositories

This command and other repository commands change only local configuration, and do not affect any cluster resources. The helm repo add command updates the ~/.config/helm/repositories.yaml configuration file, which keeps the list of configured repositories.

When repositories are configured, other commands can use the list of repositories to perform actions. For example, the helm search repo command lists all available charts in the configured repositories:

[user@host ~]$ **helm search repo**

NAME CHART VERSION APP VERSION DESCRIPTION

repo/chart 0.0.7 latest A sample chart

...output omitted...

By default, the helm search repo command shows only the latest version of a chart. Use the --versions option to list all available versions. By default, the install and upgrade commands use the latest version of the chart in the repository. You can use the --version option to install specific versions.

### REFERENCES

[Using Helm](https://helm.sh/docs/intro/using_helm/" \t "_top)

[Helm Charts](https://helm.sh/docs/topics/charts/" \t "_top)

[Helm Chart Repository Guide](https://helm.sh/docs/topics/chart_repository/" \t "_top)

## **Guided Exercise: Helm Charts**

Deploy and update an application from a chart that is stored in a catalog.

**Outcomes**

* Deploy an application and its dependencies from a Helm chart.
* Customize the deployment, including scaling and using a custom image.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable.

[student@workstation ~]$ **lab start packaged-charts**

**Instructions**

1. Add the classroom Helm repository at the following URL and examine its contents.

http://helm.ocp4.example.com/charts

* 1. Use the helm repo list command to list the repositories that are configured for the student user.
  2. [student@workstation ~]$ **helm repo list**

Error: no repositories to show

If the do280-repo repository is present, then continue to the next step. Otherwise, add the repository.

[student@workstation ~]$ **helm repo add do280-repo \**

**http://helm.ocp4.example.com/charts**

"do280-repo" has been added to your repositories

* 1. Use the helm search command to list all the chart versions in the repository.
  2. [student@workstation ~]$ **helm search repo --versions**
  3. NAME CHART VERSION APP VERSION ...
  4. do280-repo/etherpad 0.0.7 latest ...
  5. do280-repo/etherpad 0.0.6 latest ...

...output omitted...

The etherpad chart has the 0.0.7 and 0.0.6 versions. This chart is a copy of a chart from the [https://github.com/redhat-cop/helm-charts](https://github.com/redhat-cop/helm-charts" \t "_top) repository.

1. Install the 0.0.6 version of the etherpad chart to a new packaged-charts-development project, with the example-app release name.

Use the registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18 image in the offline classroom registry. Expose the application at the https://development-etherpad.apps.ocp4.example.com URL.

* 1. Examine the values of the chart.
  2. [student@workstation ~]$ **helm show values do280-repo/etherpad --version 0.0.6**
  3. # Default values for etherpad.
  4. replicaCount: 1
  5. defaultTitle: "Labs Etherpad"
  6. defaultText: "Assign yourself a user and share your ideas!"
  7. image:
  8. repository: etherpad
  9. name:
  10. tag:
  11. pullPolicy: IfNotPresent
  12. ...output omitted...
  13. route:
  14. enabled: true
  15. host: null
  16. targetPort: http
  17. ...output omitted...
  18. resources: {}

...output omitted...

You can configure the image, the replica count, and other values. By default, the chart creates a route. You can customize the route with the route.host key.

With the default configuration, the chart uses the docker.io/etherpad/etherpad:latest image. The classroom environment is designed for offline use. Use the registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18 image from the local registry instead.

* 1. Create a values.yaml file with the following content:
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad
  5. tag: 1.8.18
  6. route:

host: development-etherpad.apps.ocp4.example.com

* 1. Log in to the cluster as the developer user with the developer password.
  2. [student@workstation ~]$ **oc login -u developer -p developer \**
  3. **https://api.ocp4.example.com:6443**

...output omitted...

* 1. Create a packaged-charts-development project.
  2. [student@workstation ~]$ **oc new-project packaged-charts-development**
  3. Now using project "packaged-charts-development" on server ...

...output omitted...

* 1. Install the etherpad chart to the packaged-charts-development project. Use the values.yaml file that you created in a previous step. Use example-app as the release name.
  2. [student@workstation ~]$ **helm install example-app do280-repo/etherpad \**
  3. **-f values.yaml --version 0.0.6**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. NAME: example-app
  6. LAST DEPLOYED: Mon Jun 5 06:31:26 2023
  7. NAMESPACE: packaged-charts-development
  8. STATUS: deployed
  9. REVISION: 1

TEST SUITE: None

### NOTE

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

* 1. Get the route to verify that you customized the route correctly.
  2. [student@workstation ~]$ **oc get route**
  3. NAME HOST/PORT ...

example-app-etherpad development-etherpad.apps.ocp4.example.com ...

* 1. Open a web browser and navigate to https://development-etherpad.apps.ocp4.example.com. The application welcome page appears.

1. Upgrade a Helm chart by installing the 0.0.7 version of the chart.
   1. Use the helm list command to verify the installed version.
   2. [student@workstation ~]$ **helm list**
   3. NAME NAMESPACE REVISION ... STATUS CHART

example-app packaged-charts-development 1 ... deployed **etherpad-0.0.6**

* 1. Use the helm search command to verify that the repository contains a later version.
  2. [student@workstation ~]$ **helm search repo --versions**
  3. NAME CHART VERSION APP VERSION ...
  4. do280-repo/etherpad **0.0.7** latest ...
  5. do280-repo/etherpad 0.0.6 latest ...

...output omitted...

* 1. Use the helm upgrade command to upgrade to the latest version of the chart.
  2. [student@workstation ~]$ **helm upgrade example-app do280-repo/etherpad \**
  3. **-f values.yaml --version 0.0.7**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. Release "example-app" has been upgraded. Happy Helming!
  6. NAME: example-app
  7. LAST DEPLOYED: Mon Jun 5 06:41:00 2023
  8. NAMESPACE: packaged-charts-development
  9. STATUS: deployed
  10. REVISION: 2

TEST SUITE: None

* 1. Use the helm list command to verify the installed version.
  2. [student@workstation ~]$ **helm list**
  3. NAME NAMESPACE REVISION ... STATUS CHART

example-app packaged-charts-development 2 ... deployed **etherpad-0.0.7**

* 1. Reload the application welcome page in the web browser.

The updates in the new version of the chart do not affect the deployment in this exercise. When you reload the application, the browser displays the same application welcome page.

1. Create a second deployment of the chart to a new packaged-charts-production project, with the example-app release name.

Expose the application at the https://etherpad.apps.ocp4.example.com URL, by customizing the route.host key.

* 1. Create a packaged-charts-production project.
  2. [student@workstation ~]$ **oc new-project packaged-charts-production**
  3. Now using project "packaged-charts-production" on server ...

...output omitted...

* 1. Edit the values.yaml file to configure the host route to etherpad.apps.ocp4.example.com.
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad
  5. tag: 1.8.18
  6. **route:**

**host: etherpad.apps.ocp4.example.com**

* 1. Install the 0.0.7 version of the etherpad chart to the packaged-review-production project.

Use the values.yaml file that you edited in a previous step. Use production as the release name.

[student@workstation ~]$ **helm install example-app do280-repo/etherpad \**

**-f values.yaml --version 0.0.7**

...output omitted...

* 1. Verify the deployment by opening a web browser and navigating to the application URL. https://etherpad.apps.ocp4.example.com

This URL corresponds to the host that you specified in the values.yaml file. The application welcome page appears in the production URL.

1. Reconfigure the production deployment to sustain heavier use. Change the number of replicas to 3.
   1. Verify that the application has a single pod.
   2. [student@workstation ~]$ **oc get pods**
   3. NAME READY STATUS RESTARTS AGE

example-app-etherpad-6b85b94975-qfpqm 1/1 Running 0 12s

* 1. Edit the values.yaml file. Add a replicaCount key with the 3 value.
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad
  5. tag: 1.8.18
  6. route:
  7. host: etherpad.apps.ocp4.example.com

**replicaCount: 3**

* 1. Use the helm upgrade command to update the parameters.
  2. [student@workstation ~]$ **helm upgrade example-app do280-repo/etherpad \**
  3. **-f values.yaml**

...output omitted...

* 1. Verify that the application has three pods.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE
  4. example-app-etherpad-6b85b94975-h9qgz 1/1 Running 0 13s
  5. example-app-etherpad-6b85b94975-lbr8h 1/1 Running 0 13s

example-app-etherpad-6b85b94975-qfpqm 1/1 Running 0 94s

* 1. Reload the application welcome page in the web browser.

The deployment continues working after adding replicas.

1. Remove the values.yaml file.

[student@workstation ~]$ **rm values.yaml**

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish packaged-charts**

## **Lab: Deploy Packaged Applications**

Deploy and update applications from resource manifests that are packaged for sharing and distribution.

**Outcomes**

* Deploy an application and its dependencies from resource manifests that are packaged as a Helm chart.
* Update the application to a later version by using the Helm chart.
* Use a container image in a private container registry instead of a public registry.
* Customize the deployment to add resource requests and limits.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable.

[student@workstation ~]$ **lab start packaged-review**

**Instructions**

1. Log in to the cluster as the developer user with the developer password. Create the packaged-review and packaged-review-prod projects.
   1. Log in to the cluster as the developer user with the developer password.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**

**https://api.ocp4.example.com:6443**

* 1. Create the packaged-review project.
  2. [student@workstation ~]$ **oc new-project packaged-review**
  3. Now using project "packaged-review" on server ...

...output omitted...

* 1. Create the packaged-review-prod project.
  2. [student@workstation ~]$ **oc new-project packaged-review-prod**
  3. Now using project "packaged-review-prod" on server ...

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Add the classroom Helm repository at the http://helm.ocp4.example.com/charts URL and examine its contents.
   1. Use the helm repo list command to list the repositories that are configured for the student user.
   2. [student@workstation ~]$ **helm repo list**
   3. NAME URL

do280-repo http://helm.ocp4.example.com/charts

If the do280-repo repository is present, then continue to the next step. Otherwise, add the repository.

[student@workstation ~]$ **helm repo add \**

**do280-repo http://helm.ocp4.example.com/charts**

"do280-repo" has been added to your repositories

* 1. Use the helm search command to list all the chart versions in the repository.

The etherpad chart has versions 0.0.6 and 0.0.7. This chart is a copy of a chart from the https://github.com/redhat-cop/helm-charts repository.

[student@workstation ~]$ **helm search repo --versions**

NAME CHART VERSION APP VERSION DESCRIPTION

do280-repo/etherpad 0.0.7 latest ...

do280-repo/etherpad 0.0.6 latest ...

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Install the 0.0.6 version of the etherpad chart on the packaged-review namespace, with the test release name. Use the registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17 image in the offline classroom registry.

Create a values-test.yaml file with the image repository, name, and tag.

| **Field** | **Value** |
| --- | --- |
| image.repository | registry.ocp4.example.com:8443/etherpad |
| image.name | etherpad |
| image.tag | 1.8.17 |

* 1. Switch to the packaged-review project.
  2. [student@workstation ~]$ **oc project packaged-review**

Now using project "packaged-review" on server ...

* 1. Examine the values of the chart.

You can configure the image, the deployment resources, and other values. By default, the chart creates a route.

[student@workstation ~]$ **helm show values do280-repo/etherpad --version 0.0.6**

# Default values for etherpad.

replicaCount: 1

defaultTitle: "Labs Etherpad"

defaultText: "Assign yourself a user and share your ideas!"

image:

**repository**: etherpad

**name**:

**tag**:

pullPolicy: IfNotPresent

...output omitted...

route:

enabled: true

**host**: null

targetPort: http

...output omitted...

**resources**: {}

...output omitted...

|  |  |
| --- | --- |
|  | The registry with the container image |
|  | Container image name |
|  | Container image tag |
|  | Hostname for the OpenShift route resource |
|  | The resource requests and limits for this workload. This value is set by default to {}, which indicates that it is an empty map. |

* 1. With the default configuration, the chart uses the docker.io/etherpad/etherpad:latest container image.

This image is not suitable for the classroom environment. Use the registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17 container image instead.

Create a values-test.yaml file with the following content:

image:

repository: registry.ocp4.example.com:8443/etherpad

name: etherpad

tag: 1.8.17

* 1. Install the etherpad chart in the packaged-review namespace.
     + Use the values-test.yaml file that you created in the previous step.
     + Use test as the release name.
  2. [student@workstation ~]$ **helm install test do280-repo/etherpad \**
  3. **-f values-test.yaml --version 0.0.6**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. NAME: test
  6. LAST DEPLOYED: Fri Jun 30 01:03:42 2023
  7. NAMESPACE: packaged-review
  8. STATUS: deployed
  9. REVISION: 1

TEST SUITE: None

### NOTE

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

* 1. Use the helm list command to verify the installed version of the etherpad chart.
  2. [student@workstation ~]$ **helm list**
  3. NAME NAMESPACE REVISION ... STATUS CHART APP VERSION

test packaged-review 1 ... deployed **etherpad-0.0.6** latest

* 1. Verify that the pod is running and that the deployment is ready.
  2. [student@workstation ~]$ **oc get deployments,pods**
  3. NAME READY UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/test-etherpad **1/1** 1 1 27s
  5. NAME READY STATUS RESTARTS AGE

pod/test-etherpad-c6657b556-4jh8z 1/1 **Running** 0 27s

* 1. Verify that the pod executes the specified container image.
  2. [student@workstation ~]$ **oc describe pods -n packaged-review | \**
  3. **egrep '^Name:|Image:'**
  4. Name: test-etherpad-c6657b556-4jh8z

Image: registry.ocp4.example.com:8443/etherpad/etherpad:**1.8.17**

* 1. Get the route to obtain the application URL.
  2. [student@workstation ~]$ **oc get routes**
  3. NAME HOST/PORT ...

test-etherpad **test-etherpad-packaged-review.apps.ocp4.example.com** ...

* 1. Open a web browser and navigate to the following URL to view the application page.

https://test-etherpad-packaged-review.apps.ocp4.example.com

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Upgrade the etherpad application in the packaged-review namespace to the 0.0.7 version of the chart. Set the image tag for the deployment in the values-test.yaml file.

| **Field** | **Value** |
| --- | --- |
| image.tag | 1.8.18 |

* 1. Edit the values-test.yaml file and update the image tag value:
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad

**tag: 1.8.18**

* 1. Use the helm search command to verify that the repository contains a more recent version of the etherpad chart.
  2. [student@workstation ~]$ **helm search repo --versions etherpad**
  3. NAME CHART VERSION APP VERSION DESCRIPTION
  4. do280-repo/etherpad **0.0.7** latest ...

do280-repo/etherpad 0.0.6 latest ...

* 1. Use the helm upgrade command to upgrade to the latest version of the chart.
  2. [student@workstation ~]$ **helm upgrade test do280-repo/etherpad \**
  3. **-f values-test.yaml --version 0.0.7**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. Release "test" has been upgraded. Happy Helming!
  6. NAME: test
  7. LAST DEPLOYED: Fri Jun 30 01:05:07 2023
  8. NAMESPACE: packaged-review
  9. STATUS: deployed
  10. REVISION: 2

TEST SUITE: None

* 1. Use the helm list command to verify the installed version of the etherpad chart.
  2. [student@workstation ~]$ **helm list**
  3. NAME NAMESPACE REVISION ... STATUS CHART APP VERSION

test packaged-review 2 ... deployed **etherpad-0.0.7** latest

* 1. Verify that the pod is running and that the deployment is ready.
  2. [student@workstation ~]$ **oc get deployments,pods**
  3. NAME READY UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/test-etherpad **1/1** 1 1 3m31s
  5. NAME READY STATUS RESTARTS AGE

pod/test-etherpad-59d775b78f-ftmsz 1/1 **Running** 0 64s

* 1. Verify that the pod executes the updated container image.
  2. [student@workstation ~]$ **oc describe pods -n packaged-review | \**
  3. **egrep '^Name:|Image:'**
  4. Name: test-etherpad-59d775b78f-ftmsz

Image: registry.ocp4.example.com:8443/etherpad/etherpad:**1.8.18**

* 1. Reload the test-etherpad application welcome page in the web browser.

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Create a second deployment of the chart in the packaged-review-prod namespace, with the prod release name. Copy the values-test.yaml file to the values-prod.yaml file, and set the route host.

| **Field** | **Value** |
| --- | --- |
| route.host | etherpad.apps.ocp4.example.com |

1. Access the application in the route URL to verify that it is working correctly.
2. https://etherpad.apps.ocp4.example.com
   1. Switch to the packaged-review-prod project.
   2. [student@workstation ~]$ **oc project packaged-review-prod**

Now using project "packaged-review-prod" on server ...

* 1. Copy the values-test.yaml file to values-prod.yaml.

[student@workstation ~]$ **cp values-test.yaml values-prod.yaml**

* 1. Set the route host in the values-prod.yaml file.
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad
  5. tag: 1.8.18
  6. **route:**

**host: etherpad.apps.ocp4.example.com**

* 1. Install the 0.0.6 version of the etherpad chart on the packaged-review-prod namespace.
     + Use the values-prod.yaml file that you edited in the previous step.
     + Use prod as the release name.
  2. [student@workstation ~]$ **helm install prod do280-repo/etherpad \**
  3. **-f values-prod.yaml --version 0.0.6**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. NAME: prod
  6. LAST DEPLOYED: Fri Jun 30 01:07:29 2023
  7. NAMESPACE: packaged-review-prod
  8. STATUS: deployed
  9. REVISION: 1

TEST SUITE: None

### NOTE

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

* 1. Use the helm list command to verify the installed version of the etherpad chart.
  2. [student@workstation ~]$ **helm list**
  3. NAME NAMESPACE REVISION ... STATUS CHART APP VERSION

prod packaged-review-prod 1 ... deployed **etherpad-0.0.6** latest

* 1. Verify that the pod is running and that the deployment is ready.
  2. [student@workstation ~]$ **oc get deployments,pods**
  3. NAME READY UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/prod-etherpad **1/1** 1 1 65s
  5. NAME READY STATUS RESTARTS AGE

pod/prod-etherpad-5947dfb987-9dclr 1/1 **Running** 0 65s

* 1. Verify that the pod executes the specified container image.
  2. [student@workstation ~]$ **oc describe pods -n packaged-review-prod | \**
  3. **egrep '^Name:|Image:'**
  4. Name: pod/prod-etherpad-5947dfb987-9dclr

Image: registry.ocp4.example.com:8443/etherpad/etherpad:**1.8.18**

* 1. Verify the deployment by opening a web browser and navigating to the application URL. This URL corresponds to the host that you specified in the values-prod.yaml file. The application welcome page appears in the production URL.

https://etherpad.apps.ocp4.example.com

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Add limits to the etherpad instance in the packaged-review-prod namespace. The chart values example contains comments that show the required format for this change. Set limits and requests for the deployment in the values-prod.yaml file.

| **Field** | **Value** |
| --- | --- |
| resources.limits.memory | 128Mi |
| resources.requests.memory | 128Mi |

* 1. Edit the values-prod.yaml file. Configure the deployment to request 128 MiB of RAM, and limit RAM usage to 128 MiB.
  2. image:
  3. repository: registry.ocp4.example.com:8443/etherpad
  4. name: etherpad
  5. tag: 1.8.18
  6. route:
  7. host: etherpad.apps.ocp4.example.com
  8. **resources:**
  9. **limits:**
  10. **memory: 128Mi**
  11. **requests:**

**memory: 128Mi**

* 1. Use the helm upgrade command to upgrade to the latest version of the chart.
  2. [student@workstation ~]$ **helm upgrade prod do280-repo/etherpad \**
  3. **-f values-prod.yaml --version 0.0.7**
  4. ... would violate PodSecurity "restricted:v1.24": ...output omitted...
  5. Release "prod" has been upgraded. Happy Helming!
  6. NAME: prod
  7. LAST DEPLOYED: Fri Jun 30 01:09:04 2023
  8. NAMESPACE: packaged-review-prod
  9. STATUS: deployed
  10. REVISION: 2

TEST SUITE: None

* 1. Verify that the pod is running and that the deployment is ready.
  2. [student@workstation ~]$ **oc get deployments,pods**
  3. NAME READY UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/prod-etherpad **1/1** 1 1 3m14s
  5. NAME READY STATUS RESTARTS AGE

pod/prod-etherpad-6b7d9dffbc-f7cng 1/1 **Running** 0 36s

* 1. Examine the application pod from the production instance of the application to verify the configuration change.
  2. [student@workstation ~]$ **oc describe pods -n packaged-review-prod | \**
  3. **egrep -A1 '^Name:|Limits|Requests'**
  4. Name: prod-etherpad-6b7d9dffbc-f7cng
  5. Namespace: packaged-review-prod
  6. ​--
  7. Limits:
  8. memory: **128Mi**
  9. Requests:

memory: **128Mi**

* 1. Examine the pod of the test instance of the application in the packaged-review namespace. This deployment uses the values from the values-test.yaml file that did not specify resource limits or requests. The pod in the packaged-review namespace does not have a custom resource allocation.
  2. [student@workstation ~]$ **oc describe pods -n packaged-review | \**
  3. **egrep -A1 '^Name:|Limits|Requests'**
  4. Name: test-etherpad-59d775b78f-ftmsz

Namespace: packaged-review

* 1. Use the helm list command to verify the installed version of the etherpad chart.
  2. [student@workstation ~]$ **helm list**
  3. NAME NAMESPACE REVISION ... STATUS CHART APP VERSION

prod packaged-review-prod 2 ... deployed **etherpad-0.0.7** latest

* 1. Reload the application welcome page in the web browser. The deployment continues working after you add the limits.
  2. Remove the values-test.yaml and values-prod.yaml files.

[student@workstation ~]$ **rm values-test.yaml values-prod.yaml**

1. [Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade packaged-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish packaged-review**

# Chapter 3.  Authentication and Authorization

[Configure Identity Providers](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03)

[Guided Exercise: Configure Identity Providers](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03s02)

[Define and Apply Permissions with RBAC](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03s03)

[Guided Exercise: Define and Apply Permissions with RBAC](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03s04)

[Lab: Authentication and Authorization](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03s05)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch03s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Configure authentication with the HTPasswd identity provider and assign roles to users and groups. |
| **Objectives** | * Configure the HTPasswd identity provider for OpenShift authentication. * Define role-based access controls and apply permissions to users. |
| **Sections** | * Configure Identity Providers (and Guided Exercise) * Define and Apply Permissions with RBAC (and Guided Exercise) |
| **Lab** | * Authentication and Authorization |

## **Configure Identity Providers**

### **Objectives**

* Configure the HTPasswd identity provider for OpenShift authentication.

### **OpenShift Users and Groups**

Several OpenShift resources relate to authentication and authorization. The following list shows the primary resource types and their definitions:

**User**

In the OpenShift Container Platform architecture, users are entities that interact with the API server. The user resource represents an actor within the system. Assign permissions by adding roles to the user directly or to the groups that the user is a member of.

**Identity**

The identity resource keeps a record of successful authentication attempts from a specific user and identity provider. Any data about the source of the authentication is stored on the identity.

**Service Account**

In OpenShift, applications can communicate with the API independently when user credentials cannot be acquired. To preserve the integrity of the credentials for a regular user, credentials are not shared and service accounts are used instead. With service accounts, you can control API access without the need to borrow a regular user's credentials.

**Group**

Groups represent a specific set of users. Users are assigned to groups. Authorization policies use groups to assign permissions to multiple users at the same time. For example, to grant 20 users access to objects within a project, it is better to use a group instead of granting access to each user individually. OpenShift Container Platform also provides system groups or virtual groups that are provisioned automatically by the cluster.

**Role**

A role defines the API operations that a user has permissions to perform on specified resource types. You grant permissions to users, groups, and service accounts by assigning roles to them.

User and identity resources are usually not created in advance. OpenShift usually creates these resources automatically after a successful interactive login with OAuth.

### **Authenticating API Requests**

The authentication and authorization security layers enable user interaction with the cluster. When a user makes a request to the API, the API associates the user with the request. The authentication layer authenticates the user. On successful authentication, the authorization layer either accepts or rejects the API request. The authorization layer uses role-based access control (RBAC) policies to determine user privileges.

The OpenShift API has two methods for authenticating requests:

* OAuth access tokens
* X.509 client certificates

If the request does not present an access token or certificate, then the authentication layer assigns it the system:anonymous virtual user and the system:unauthenticated virtual group.

#### **The Authentication Operator**

The OpenShift Container Platform provides the Authentication operator, which runs an OAuth server. The OAuth server provides OAuth access tokens to users when they attempt to authenticate to the API. An identity provider must be configured and available to the OAuth server. The OAuth server uses an identity provider to validate the identity of the requester. The server reconciles the user with the identity and creates the OAuth access token for the user. OpenShift automatically creates identity and user resources after a successful login.

#### **Identity Providers**

The OpenShift OAuth server can be configured to use many identity providers. The following lists includes the more common identity providers:

**HTPasswd**

Validates usernames and passwords against a secret that stores credentials that are generated by using the htpasswd command.

**Keystone**

Enables shared authentication with an OpenStack Keystone v3 server.

**LDAP**

Configures the LDAP identity provider to validate usernames and passwords against an LDAPv3 server, by using simple bind authentication.

**GitHub or GitHub Enterprise**

Configures a GitHub identity provider to validate usernames and passwords against GitHub or the GitHub Enterprise OAuth authentication server.

**OpenID Connect**

Integrates with an OpenID Connect identity provider by using an Authorization Code Flow.

The OAuth custom resource must be updated with your chosen identity provider. You can define multiple identity providers, of the same or different kinds, on the same OAuth custom resource.

### **Authenticating as a Cluster Administrator**

Before you can configure an identity provider and manage users, you must access your OpenShift cluster as a cluster administrator. A newly installed OpenShift cluster provides two ways to authenticate API requests with cluster administrator privileges. One way is to use the kubeconfig file, which embeds an X.509 client certificate that never expires. Another way is to authenticate as the kubeadmin virtual user. Successful authentication grants an OAuth access token.

To create additional users and grant them different access levels, you must configure an identity provider and assign roles to your users.

#### **Authenticating with the X.509 Certificate**

During installation, the OpenShift installer creates a unique kubeconfig file in the auth directory. The kubeconfig file contains specific details and parameters for the CLI to connect a client to the correct API server, including an X.509 certificate.

The installation logs provide the location of the kubeconfig file:

INFO Run 'export KUBECONFIG=root/auth/kubeconfig' to manage the cluster with 'oc'.

### NOTE

In the classroom environment, the utility machine stores the kubeconfig file at /home/lab/ocp4/auth/kubeconfig.

To use the kubeconfig file to authenticate oc commands, you must copy the file to your workstation and set the absolute or relative path to the KUBECONFIG environment variable. Then, you can run any oc command that requires cluster administrator privileges without logging in to OpenShift.

[user@host ~]$ **export KUBECONFIG=/home/user/auth/kubeconfig**

[user@host ~]$ **oc get nodes**

As an alternative, you can use the --kubeconfig option of the oc command.

[user@host ~]$ **oc --kubeconfig /home/user/auth/kubeconfig get nodes**

#### **Authenticating with the kubeadmin Virtual User**

After installation completes, OpenShift creates the kubeadmin virtual user. The kubeadmin secret in the kube-system namespace contains the hashed password for the kubeadmin user. The kubeadmin user has cluster administrator privileges.

The OpenShift installer dynamically generates a unique kubeadmin password for the cluster. The installation logs provide the kubeadmin credentials to log in to the cluster. The cluster installation logs also provide the login, password, and the URL for console access.

...output omitted...

INFO The cluster is ready when '**oc login -u kubeadmin -p shdU\_trbi\_6ucX\_edbu\_aqop'**

...output omitted...

INFO Access the OpenShift web-console here:

https://console-openshift-console.apps.ocp4.example.com

INFO Login to the console with user: kubeadmin, password: shdU\_trbi\_6ucX\_edbu\_aqop

### NOTE

In the classroom environment, the utility machine stores the password for the kubeadmin user in the /home/lab/ocp4/auth/kubeadmin-password file.

#### **Deleting the Virtual User**

After you define an identity provider, create a user, and assign that user the cluster-admin role, you can remove the kubeadmin user credentials to improve cluster security.

[user@host ~]$ **oc delete secret kubeadmin -n kube-system**

### WARNING

If you delete the kubeadmin secret before you configure another user with cluster admin privileges, then you can administer your cluster only by using the kubeconfig file. If you do not have a copy of this file in a safe location, then you cannot recover administrative access to your cluster. The only alternative is to destroy and reinstall your cluster.

### WARNING

**Do not** delete the kubeadmin user at any time during this course. The kubeadmin user is essential to the course lab architecture. If you deleted this user, you would have to delete the lab environment and re-create it.

### **Configuring the HTPasswd Identity Provider**

The HTPasswd identity provider validates users against a secret that contains usernames and passwords that are generated with the htpasswd command from the Apache HTTP Server project. Only a cluster administrator can change the data inside the HTPasswd secret. Regular users cannot change their own passwords.

Managing users with the HTPasswd identity provider might suffice for a proof-of-concept environment with a small set of users. However, most production environments require a more powerful identity provider that integrates with the organization's identity management system.

#### **Configuring the OAuth Custom Resource**

To use the HTPasswd identity provider, the OAuth custom resource must be edited to add an entry to the .spec.identityProviders array:

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

**spec**:

**identityProviders**:

- name: **my\_htpasswd\_provider**

mappingMethod: **claim**

type: HTPasswd

htpasswd:

fileData:

name: **htpasswd-secret**

|  |  |
| --- | --- |
|  | This provider name is prefixed to provider user names to form an identity name. |
|  | Controls how mappings are established between provider identities and user objects. With the default claim value, you cannot log in with different identity providers. |
|  | An existing secret that contains data that is generated by using the htpasswd command. |

#### **Updating the OAuth Custom Resource**

To update the OAuth custom resource, use the oc get command to export the existing OAuth cluster resource to a file in YAML format.

[user@host ~]$ **oc get oauth cluster -o yaml > oauth.yaml**

Then, open the resulting file in a text editor and make the needed changes to the embedded identity provider settings.

After completing modifications and saving the file, you must apply the new custom resource by using the oc replace command.

[user@host ~]$ **oc replace -f oauth.yaml**

### **Managing Users with the HTPasswd Identity Provider**

Managing user credentials with the HTPasswd Identity Provider requires creating a temporary htpasswd file, changing the file, and applying these changes to the secret.

#### **Creating an HTPasswd File**

The httpd-tools package provides the htpasswd utility, which must be installed and available on your system.

Create the htpasswd file.

[user@host ~]$ **htpasswd -c -B -b /tmp/htpasswd student redhat123**

### IMPORTANT

Use the -c option only when creating a file. The -c option replaces all file content if the file already exists.

Add or update credentials.

[user@host ~]$ **htpasswd -b /tmp/htpasswd student redhat1234**

Delete credentials.

[user@host ~]$ **htpasswd -D /tmp/htpasswd student**

#### **Creating the HTPasswd Secret**

To use the HTPasswd provider, you must create a secret that contains the htpasswd file data. The following example uses a secret named htpasswd-secret.

[user@host ~]$ **oc create secret generic htpasswd-secret \**

**--from-file htpasswd=/tmp/htpasswd -n openshift-config**

### IMPORTANT

A secret that the HTPasswd identity provider uses requires adding the htpasswd= prefix before specifying the path to the file.

#### **Extracting Secret Data**

When adding or removing users, use the oc extract command to retrieve the secret. Extracting the secret ensures that you work on the current set of users.

By default, the oc extract command saves each key within a configuration map or secret as a separate file. Alternatively, you can then redirect all data to a file or display it as standard output. To extract data from the htpasswd-secret secret to the /tmp/ directory, use the following command. The --confirm option replaces the file if it exists.

[user@host ~]$ **oc extract secret/htpasswd-secret -n openshift-config \**

**--to /tmp/ --confirm**

/tmp/htpasswd

#### **Updating the HTPasswd Secret**

The secret must be updated after adding, changing, or deleting users. Use the oc set data secret command to update a secret. Unless the file name is htpasswd, you must specify htpasswd= to update the htpasswd key within the secret.

The following command updates the htpasswd-secret secret in the openshift-config namespace by using the content of the /tmp/htpasswd file.

[user@host ~]$ **oc set data secret/htpasswd-secret \**

**--from-file htpasswd=/tmp/htpasswd -n openshift-config**

After updating the secret, the OAuth operator redeploys pods in the openshift-authentication namespace. Monitor the redeployment of the new OAuth pods by running the following command:

[user@host ~]$ **watch oc get pods -n openshift-authentication**

Test additions, changes, or deletions to the secret after the new pods finish deploying.

### **Deleting Users and Identities**

When a scenario occurs that requires you to delete a user, it is not sufficient to delete the user from the identity provider. The user and identity resources must also be deleted.

You must remove the password from the htpasswd secret, remove the user from the local htpasswd file, and then update the secret.

To delete the user from htpasswd, run the following command:

[user@host ~]$ **htpasswd -D /tmp/htpasswd manager**

Update the secret to remove all remnants of the user's password.

[user@host ~]$ **oc set data secret/htpasswd-secret \**

**--from-file htpasswd=/tmp/htpasswd -n openshift-config**

Remove the user resource with the following command:

[user@host ~]$ **oc delete user manager**

user.user.openshift.io "manager" deleted

Identity resources include the name of the identity provider. To delete the identity resource for the manager user, find the resource and then delete it.

[user@host ~]$ **oc get identities | grep manager**

my\_htpasswd\_provider:manager my\_htpasswd\_provider manager manager ...

[user@host ~]$ **oc delete identity my\_htpasswd\_provider:manager**

identity.user.openshift.io "my\_htpasswd\_provider:manager" deleted

### **Assigning Administrative Privileges**

The cluster-wide cluster-admin role grants cluster administration privileges to users and groups. With this role, the user can perform any action on any resources within the cluster. The following example assigns the cluster-admin role to the student user.

[user@host ~]$ **oc adm policy add-cluster-role-to-user cluster-admin student**

### REFERENCES

For more information about identity providers, refer to the Understanding Identity Provider Configuration chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/authentication\_and\_authorization/index#understanding-identity-provider](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index" \l "understanding-identity-provider" \t "_top)

## **Guided Exercise: Configure Identity Providers**

Configure the HTPasswd identity provider and create users for cluster administrators.

**Outcomes**

* Create users and passwords for HTPasswd authentication.
* Configure the Identity Provider for HTPasswd authentication.
* Assign cluster administration rights to users.

[student@workstation ~]$ **lab start auth-providers**

The command ensures that the cluster API is reachable, the httpd-utils package is installed, and that the authentication settings are configured to the installation defaults.

**Instructions**

1. Add an entry for two users, new\_admin and new\_developer. Assign the new\_admin user the redhat password, and assign the new\_developer user the developer password.
   1. Create an HTPasswd authentication file named htpasswd in the ~/DO280/labs/auth-providers/ directory. Add the new\_admin user with the redhat password. The file name is arbitrary; this exercise uses the ~/DO280/labs/auth-providers/htpasswd file.

Use the htpasswd command to populate the HTPasswd authentication file with the usernames and encrypted passwords. The -B option uses bcrypt encryption. By default, the htpasswd command uses the MD5 hashing algorithm if you do not specify another algorithm.

[student@workstation ~]$ **htpasswd -c -B -b ~/DO280/labs/auth-providers/htpasswd \**

**new\_admin redhat**

Adding password for user new\_admin

* 1. Add the new\_developer user with the developer password to the ~/DO280/labs/auth-providers/htpasswd file. The password for the new\_developer user is hashed with the MD5 algorithm, because no algorithm was specified and MD5 is the default hashing algorithm.

[student@workstation ~]$ **htpasswd -b ~/DO280/labs/auth-providers/htpasswd \**

**new\_developer developer**

Adding password for user new\_developer

* 1. Review the contents of the ~/DO280/labs/auth-providers/htpasswd file and verify that it includes two entries with hashed passwords: one for the new\_admin user and another for the new\_developer user.

[student@workstation ~]$ **cat ~/DO280/labs/auth-providers/htpasswd**

new\_admin:$2y$05$qQaFbpx4hbf4uZe.SMLSduTN8uN4DNJMJ4jE5zXDA57WrTRlpu2QS

new\_developer:$apr1$S0TxtLXl$QSRfBIufYP39pKNsIg/nD1

1. Log in to OpenShift and create a secret that contains the HTPasswd users file.
   1. Log in to the cluster as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Create a secret from the ~/DO280/labs/auth-providers/htpasswd file. To use the HTPasswd identity provider, you must define a secret with a key named htpasswd that contains the HTPasswd user file ~/DO280/labs/auth-providers/htpasswd.

[student@workstation ~]$ **oc create secret generic localusers \**

**--from-file htpasswd=~/DO280/labs/auth-providers/htpasswd \**

**-n openshift-config**

secret/localusers created

* 1. Assign the new\_admin user the cluster-admin role.

[student@workstation ~]$ **oc adm policy add-cluster-role-to-user \**

**cluster-admin new\_admin**

Warning: User 'new\_admin' not found

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "new\_admin"

### NOTE

The output indicates that the new\_admin user is not found. You can safely ignore this warning.

1. Update the HTPasswd identity provider for the cluster so that your users can authenticate. Configure the custom resource file and update the cluster.
   1. Export the existing OAuth resource to a file named oauth.yaml in the ~/DO280/labs/auth-providers directory.

[student@workstation ~]$ **oc get oauth cluster \**

**-o yaml > ~/DO280/labs/auth-providers/oauth.yaml**

### NOTE

For convenience, an oauth.yaml file that contains the completed custom resource file is downloaded to ~/DO280/solutions/auth-providers.

* 1. Edit the ~/DO280/labs/auth-providers/oauth.yaml file with your preferred text editor. You can choose the names of the identityProviders and fileData structures. For this exercise, use the myusers and localusers values, respectively.

The completed custom resource should match the following structure. Ensure that the htpasswd, mappingMethod, name, and type strings are at the same indentation level.

apiVersion: config.openshift.io/v1

kind: OAuth

...output omitted...

**spec:**

**identityProviders:**

- ldap:

...output omitted...

type: LDAP

**- htpasswd:**

**fileData:**

**name: localusers**

**mappingMethod: claim**

**name: myusers**

**type: HTPasswd**

* 1. Apply the custom resource that was defined in the previous step.
  2. [student@workstation ~]$ **oc replace -f ~/DO280/labs/auth-providers/oauth.yaml**

oauth.config.openshift.io/cluster replaced

### NOTE

Authentication changes require redeploying pods in the openshift-authentication namespace.

Use the watch command to examine the status of workloads in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get all -n openshift-authentication**

NAME READY STATUS RESTARTS AGE

pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m

...output omitted...

A few minutes after you ran the oc replace command, the redeployment starts. Wait until new pods are running. Press **Ctrl**+**C** to exit the watch command.

Provided that the previously created secret was created correctly, you can log in by using the HTPasswd identity provider.

1. Log in as the new\_admin and as the new\_developer user to verify the HTPasswd user configuration.
   1. Log in to the cluster as the new\_admin user to verify that the HTPasswd authentication is configured correctly. The authentication operator takes some time to load the configuration changes from the previous step.

### NOTE

If the authentication fails, then wait a few moments and try again.

[student@workstation ~]$ **oc login -u new\_admin -p redhat**

Login successful.

...output omitted...

* 1. Use the oc get nodes command to verify that the new\_admin user has the cluster-admin role.

[student@workstation ~]$ **oc get nodes**

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master,worker 13d v1.27.6+f67aeb3

* 1. Log in to the cluster as the new\_developer user to verify that the HTPasswd authentication is configured correctly.

[student@workstation ~]$ **oc login -u new\_developer -p developer**

Login successful.

...output omitted...

* 1. Use the oc get nodes command to verify that the new\_developer and new\_admin users do not have the same level of access.

[student@workstation ~]$ **oc get nodes**

Error from server (Forbidden): nodes is forbidden: User "new\_developer" cannot list resource "nodes" in API group "" at the cluster scope

* 1. Log in as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p redhat**

Login successful.

...output omitted...

* 1. List the current users.

[student@workstation ~]$ **oc get users**

NAME UID ... IDENTITIES

admin 6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24 ... ...

new\_admin 489c7402-d318-4805-b91d-44d786a92fc1 ... myusers:new\_admin

new\_developer 8dbae772-1dd4-4242-b2b4-955b005d9022 ... myusers:new\_developer

### NOTE

You might see additional users from previously completed exercises.

* 1. Display the list of current identities.

[student@workstation ~]$ **oc get identity**

NAME IDP NAME IDP USER NAME USER NAME

USER UID

... ... ... admin

6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24

myusers:new\_admin myusers new\_admin new\_admin

489c7402-d318-4805-b91d-44d786a92fc1

myusers:new\_developer myusers new\_developer new\_developer

8dbae772-1dd4-4242-b2b4-955b005d9022

### NOTE

You might see additional identities from previously completed exercises.

1. As the new\_admin user, create a HTPasswd user named manager with a password of redhat.
   1. Extract the file data from the secret to the ~/DO280/labs/auth-providers/htpasswd file.

[student@workstation ~]$ **oc extract secret/localusers -n openshift-config \**

**--to ~/DO280/labs/auth-providers/ --confirm**

/home/student/DO280/labs/auth-providers/htpasswd

* 1. Add an entry to your ~/DO280/labs/auth-providers/htpasswd file for the additional manager user with the redhat password.

[student@workstation ~]$ **htpasswd -b ~/DO280/labs/auth-providers/htpasswd \**

**manager redhat**

Adding password for user manager

* 1. Review the contents of your ~/DO280/labs/auth-providers/htpasswd file and verify that it includes three entries with hashed passwords: one each for the new\_admin, new\_developer, and manager users.

[student@workstation ~]$ **cat ~/DO280/labs/auth-providers/htpasswd**

new\_admin:$2y$05$qQaFbpx4hbf4uZe.SMLSduTN8uN4DNJMJ4jE5zXDA57WrTRlpu2QS

new\_developer:$apr1$S0TxtLXl$QSRfBIufYP39pKNsIg/nD1

manager:$apr1$HZ/9tC6b$j2OcHHg2GO2SSu1wyGOge.

* 1. You must update the secret after adding additional users. Use the oc set data secret command to update the secret. If the command fails, then wait a few moments for the oauth operator to finish reloading, and rerun the command.

[student@workstation ~]$ **oc set data secret/localusers \**

**--from-file htpasswd=~/DO280/labs/auth-providers/htpasswd \**

**-n openshift-config**

secret/localusers data updated

* 1. Use the watch command to examine the status of workloads in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get all -n openshift-authentication**

NAME READY STATUS RESTARTS AGE

pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m

...output omitted...

A few minutes after you ran the oc set data command, the redeployment starts. Wait until new pods are running. Press **Ctrl**+**C** to exit the watch command.

* 1. Log in to the cluster as the manager user.

### NOTE

If the authentication fails, then wait a few moments and try again.

[student@workstation ~]$ **oc login -u manager -p redhat**

Login successful.

...output omitted...

1. Create an auth-providers project, and then verify that the new\_developer user cannot access the project.
   1. As the manager user, create an auth-providers project.

[student@workstation ~]$ **oc new-project auth-providers**

Now using project "auth-providers" on server https://api.ocp4.example.com:6443".

...output omitted...

* 1. Log in as the new\_developer user.

[student@workstation ~]$ **oc login -u new\_developer -p developer**

Login successful.

...output omitted...

* 1. Attempt to delete the auth-providers project.

[student@workstation ~]$ **oc delete project auth-providers**

Error from server (Forbidden): projects.project.openshift.io "auth-providers" is forbidden: User "new\_developer" cannot delete resource "projects" in API group "project.openshift.io" in the namespace "auth-providers"

1. Change the password for the manager user.
   1. Log in as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p redhat**

Login successful.

...output omitted...

* 1. Extract the file data from the secret to the ~/DO280/labs/auth-providers/htpasswd file.

[student@workstation ~]$ **oc extract secret/localusers -n openshift-config \**

**--to ~/DO280/labs/auth-providers/ --confirm**

/home/student/DO280/labs/auth-providers/htpasswd

* 1. Generate a random user password and assign it to the MANAGER\_PASSWD variable.
     1. [student@workstation ~]$ **MANAGER\_PASSWD="$(openssl rand -hex 15)"**
  2. Update the manager user to use the stored password in the MANAGER\_PASSWD variable.

[student@workstation ~]$ **htpasswd -b ~/DO280/labs/auth-providers/htpasswd \**

**manager ${MANAGER\_PASSWD}**

Updating password for user manager

* 1. Update the secret.

[student@workstation ~]$ **oc set data secret/localusers \**

**--from-file htpasswd=~/DO280/labs/auth-providers/htpasswd \**

**-n openshift-config**

secret/localusers data updated

* 1. Use the watch command to examine the status of workloads in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get all -n openshift-authentication**

NAME READY STATUS RESTARTS AGE

pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m

...output omitted...

A few minutes after you ran the oc set data command, the redeployment starts. Wait until new pods are running. Press **Ctrl**+**C** to exit the watch command.

* 1. Log in as the manager user to verify the updated password.

[student@workstation ~]$ **oc login -u manager -p ${MANAGER\_PASSWD}**

Login successful.

...output omitted...

### NOTE

If the authentication fails, then wait a few moments and try again.

1. Remove the manager user.
   1. Log in as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p redhat**

Login successful.

...output omitted...

* 1. Extract the file data from the secret to the ~/DO280/labs/auth-providers/htpasswd file.

[student@workstation ~]$ **oc extract secret/localusers -n openshift-config \**

**--to ~/DO280/labs/auth-providers/ --confirm**

/home/student/DO280/labs/auth-providers/htpasswd

* 1. Delete the manager user from the ~/DO280/labs/auth-providers/htpasswd file.

[student@workstation ~]$ **htpasswd -D ~/DO280/labs/auth-providers/htpasswd manager**

Deleting password for user manager

* 1. Update the secret.

[student@workstation ~]$ **oc set data secret/localusers \**

**--from-file htpasswd=~/DO280/labs/auth-providers/htpasswd \**

**-n openshift-config**

secret/localusers data updated

* 1. Use the watch command to examine the status of workloads in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get all -n openshift-authentication**

NAME READY STATUS RESTARTS AGE

pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m

...output omitted...

A few minutes after you ran the oc set data command, the redeployment starts. Wait until new pods are running. Press **Ctrl**+**C** to exit the watch command.

* 1. Log in as the manager user. If the login succeeds, then try again until the login fails.

[student@workstation ~]$ **oc login -u manager -p ${MANAGER\_PASSWD}**

Login failed (401 Unauthorized)

Verify you have provided correct credentials.

* 1. Log in as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p redhat**

Login successful.

...output omitted...

* 1. Delete the identity resource for the manager user.

[student@workstation ~]$ **oc delete identity "myusers:manager"**

identity.user.openshift.io "myusers:manager" deleted

* 1. Delete the user resource for the manager user.

[student@workstation ~]$ **oc delete user manager**

user.user.openshift.io manager deleted

* 1. List the current users to verify that you deleted the manager user.

[student@workstation ~]$ **oc get users**

NAME UID ... IDENTITIES

admin 6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24 ... ...

new\_admin 489c7402-d318-4805-b91d-44d786a92fc1 ... myusers:new\_admin

new\_developer 8dbae772-1dd4-4242-b2b4-955b005d9022 ... myusers:new\_developer

* 1. Display the list of current identities to verify that you deleted the manager identity.

[student@workstation ~]$ **oc get identity**

NAME IDP NAME IDP USER NAME USER NAME

USER UID

... ... ... admin

6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24

myusers:new\_admin myusers new\_admin new\_admin

489c7402-d318-4805-b91d-44d786a92fc1

myusers:new\_developer myusers new\_developer new\_developer

8dbae772-1dd4-4242-b2b4-955b005d9022

* 1. Extract the secret and verify that only the new\_admin and new\_developer users are displayed. Using --to - sends the secret to STDOUT rather than saving it to a file.

[student@workstation ~]$ **oc extract secret/localusers -n openshift-config --to -**

# htpasswd

new\_admin:$2y$05$qQaFbpx4hbf4uZe.SMLSduTN8uN4DNJMJ4jE5zXDA57WrTRlpu2QS

new\_developer:$apr1$S0TxtLXl$QSRfBIufYP39pKNsIg/nD1

1. Remove the identity provider and clean up all users.
   1. Log in as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp**

Login successful.

...output omitted...

* 1. Delete the auth-providers project.

[student@workstation ~]$ **oc delete project auth-providers**

project.project.openshift.io "auth-providers" deleted

* 1. Edit the resource in place to remove the identity provider from OAuth:

[student@workstation ~]$ **oc edit oauth**

Delete all the lines under the ldap identity provider definition. Your file should match the following example:

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- ldap:

...output omitted...

type: LDAP

**# Delete all lines below**

- htpasswd:

fileData:

name: localusers

mappingMethod: claim

name: myusers

type: HTPasswd

Save your changes, and then verify that the oc edit command applied those changes:

oauth.config.openshift.io/cluster edited

* 1. Use the watch command to examine the status of workloads in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get all -n openshift-authentication**

NAME READY STATUS RESTARTS AGE

pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m

...output omitted...

A few minutes after you ran the oc edit command, the redeployment starts. Wait until new pods are running. Press **Ctrl**+**C** to exit the watch command.

* 1. Delete the localusers secret from the openshift-config namespace.

[student@workstation ~]$ **oc delete secret localusers -n openshift-config**

secret "localusers" deleted

* 1. Delete all identity resources.

[student@workstation ~]$ **oc delete identity --all**

identity.user.openshift.io "Red Hat Identity Management:dWlk...jb20" deleted

identity.user.openshift.io "myusers:new\_admin" deleted

identity.user.openshift.io "myusers:new\_developer" deleted

### NOTE

You might see additional identities from previously completed exercises.

* 1. Delete all user resources.

[student@workstation ~]$ **oc delete user --all**

user.user.openshift.io "admin" deleted

user.user.openshift.io "developer" deleted

user.user.openshift.io "new\_admin" deleted

user.user.openshift.io "new\_developer" deleted

### NOTE

You might see additional users from previously completed exercises.

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish auth-providers**

## **Define and Apply Permissions with RBAC**

### **Objectives**

* Define role-based access controls and apply permissions to users.

### **Role-based Access Control (RBAC)**

Role-based access control (RBAC) is a technique for managing access to resources in a computer system. In Red Hat OpenShift, RBAC determines whether a user can perform certain actions within the cluster or project. You can choose between two role types, depending on the user's level of responsibility: cluster and local.

### NOTE

Authorization is a separate step from authentication.

### **Authorization Process**

The authorization process is managed by rules, roles, and bindings.

| **RBAC Object** | **Description** |
| --- | --- |
| Rule | Allowed actions for objects or groups of objects. |
| Role | Sets of rules. Users and groups can be associated with multiple roles. |
| Binding | Assignment of users or groups to a role. |

### **RBAC Scope**

Red Hat OpenShift Container Platform (RHOCP) defines two groups of roles and bindings depending on the user's scope and responsibility: cluster roles and local roles.

| **Role Level** | **Description** |
| --- | --- |
| Cluster role | Users or groups with this role level can manage the OpenShift cluster. |
| Local role | Users or groups with this role level can manage only elements at a project level. |

### NOTE

Cluster role bindings take precedence over local role bindings.

#### **Managing RBAC with the CLI**

Cluster administrators can use the oc adm policy command to add and remove cluster roles and namespace roles.

To add a cluster role to a user, use the add-cluster-role-to-user subcommand:

[user@host ~]$ **oc adm policy add-cluster-role-to-user *cluster-role* *username***

For example, to change a regular user to a cluster administrator, use the following command:

[user@host ~]$ **oc adm policy add-cluster-role-to-user cluster-admin *username***

To remove a cluster role from a user, use the remove-cluster-role-from-user subcommand:

[user@host ~]$ **oc adm policy remove-cluster-role-from-user *cluster-role* *username***

For example, to change a cluster administrator to a regular user, use the following command:

[user@host ~]$ **oc adm policy remove-cluster-role-from-user cluster-admin *username***

Rules are defined by an action and a resource. For example, the create user rule is part of the cluster-admin role.

You can use the oc adm policy who-can command to determine whether a user can execute an action on a resource. For example:

[user@host ~]$ **oc adm policy who-can delete user**

### **Default Roles**

OpenShift ships with a set of default cluster roles that can be assigned locally or to the entire cluster. You can modify these roles for fine-grained access control to OpenShift resources. Other required steps are outside the scope of this course.

| **Default roles** | **Description** |
| --- | --- |
| admin | Users with this role can manage all project resources, including granting access to other users to access the project. |
| basic-user | Users with this role have read access to the project. |
| cluster-admin | Users with this role have superuser access to the cluster resources. These users can perform any action on the cluster, and have full control of all projects. |
| cluster-status | Users with this role can get cluster status information. |
| edit | Users with this role can create, change, and delete common application resources on the project, such as services and deployments. These users cannot act on management resources such as limit ranges and quotas, and cannot manage access permissions to the project. |
| self-provisioner | Users with this role can create projects. It is a cluster role, not a project role. |
| view | Users with this role can view project resources, but cannot modify project resources. |

The admin role gives a user access to project resources such as quotas and limit ranges, and also the ability to create applications. The edit role gives a user sufficient access to act as a developer inside the project, but working under the constraints that a project administrator configured.

Project administrators can use the oc policy command to add and remove namespace roles.

Add a specified role to a user with the add-role-to-user subcommand. For example:

[user@host ~]$ **oc policy add-role-to-user *role-name* *username* -n *project***

For example, run the following command to add the dev user to the basic-user cluster role in the wordpress project.

[user@host ~]$ **oc policy add-role-to-user basic-user dev -n wordpress**

Even though basic-user is a cluster role, the add-role-to-user subcommand limits the scope of the role to the wordpress namespace for the dev user.

### **User Types**

Interaction with OpenShift Container Platform is associated with a user. An OpenShift Container Platform user object represents a user who can be granted permissions in the system by adding roles to that user or to a user's group via role bindings.

**Regular users**

Most interactive OpenShift Container Platform users are regular users, and are represented with the User object. This type of user represents a person with access to the platform.

**System users**

Many system users are created automatically when the infrastructure is defined, mainly for the infrastructure to securely interact with the API. System users include a cluster administrator (with access to everything), a per-node user, users for routers and registries, and various others. An anonymous system user is used by default for unauthenticated requests.

System user names start with a system: prefix, such as system:admin, system:openshift-registry, and system:node:node1.example.com.

**Service accounts**

Service accounts are system users that are associated with projects. Workloads can use service accounts to invoke Kubernetes APIs.

Some service account users are created automatically during project creation. Project administrators can create more service accounts to grant extra privileges to workloads. By default, service accounts have no roles. Grant roles to service accounts to enable workloads to use specific APIs.

Service accounts are represented with the ServiceAccount object.

System account user names start with a system:serviceaccount:*namespace*: prefix, such as system:serviceaccount:default:deployer and system:serviceaccount:accounting:builder.

Every user must authenticate before they can access OpenShift Container Platform. API requests with no authentication or invalid authentication are authenticated as requests by the anonymous system user. After successful authentication, the policy determines what the user is authorized to do.

### **Group Management**

A group resource represents a set of users. Cluster administrators can use the oc adm groups command to add groups or to add users to groups. For example, run the following command to add the lead-developers group to the cluster:

[user@host ~]$ **oc adm groups new lead-developers**

Likewise, the following command adds the user1 user to the lead-developers group:

[user@host ~]$ **oc adm groups add-user lead-developers user1**

### REFERENCES

For more information about RBAC, refer to the Using RBAC to Define and Apply Permissions chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/authentication\_and\_authorization/index#using-rbac](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index" \l "using-rbac" \t "_top)

For more information about groups, refer to the Understanding Authentication chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/authentication\_and\_authorization/index#understanding-authentication](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index" \l "understanding-authentication" \t "_top)

[Kubernetes Namespaces](https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/" \t "_top)

## **Guided Exercise: Define and Apply Permissions with RBAC**

Define role-based access controls and apply permissions to users.

**Outcomes**

* Remove project creation privileges from users who are not OpenShift cluster administrators.
* Create OpenShift groups and add members to these groups.
* Create a project and assign project administration privileges to the project.
* As a project administrator, assign read and write privileges to different groups of users.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates some HTPasswd users for the exercise.

[student@workstation ~]$ **lab start auth-rbac**

**Instructions**

1. Log in to the OpenShift cluster and determine which cluster role bindings assign the self-provisioner cluster role.
   1. Log in to the cluster as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. List all cluster role bindings that reference the self-provisioner cluster role.

[student@workstation ~]$ **oc get clusterrolebinding -o wide | \**

**grep -E 'ROLE|self-provisioner'**

NAME ROLE ... GROUPS ...

self-provisioners **ClusterRole/self-provisioner** ... **system:authenticated:oauth**

1. Remove the privilege to create projects from all users who are not cluster administrators by deleting the self-provisioner cluster role from the system:authenticated:oauth virtual group.
   1. Confirm that the self-provisioners cluster role binding that you found in the previous step assigns the self-provisioner cluster role to the system:authenticated:oauth group.

[student@workstation ~]$ **oc describe clusterrolebindings 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**

* 1. Remove the self-provisioner cluster role from the system:authenticated:oauth virtual group, which deletes the self-provisioners role binding.

[student@workstation ~]$ **oc adm policy remove-cluster-role-from-group \**

**self-provisioner system:authenticated:oauth**

Warning: Your changes may get lost whenever a master is restarted, unless you prevent reconciliation of this rolebinding using the following command:

oc annotate clusterrolebinding.rbac self-provisioners 'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite

clusterrole.rbac.authorization.k8s.io/self-provisioner removed: "system:authenticated:oauth"

### NOTE

You can safely ignore the warning about your changes being lost.

* 1. Verify that the role is removed from the group. The cluster role binding self-provisioners should not exist.

[student@workstation ~]$ **oc describe clusterrolebindings self-provisioners**

Error from server (NotFound): clusterrolebindings.rbac.authorization.k8s.io "self-provisioners" not found

* 1. Determine whether any other cluster role bindings reference the self-provisioner cluster role.

[student@workstation ~]$ **oc get clusterrolebinding -o wide | \**

**grep -E 'ROLE|self-provisioner'**

NAME ROLE AGE USERS GROUPS SERVICEACCOUNTS

* 1. Log in as the leader user with the redhat password.

[student@workstation ~]$ **oc login -u leader -p redhat**

Login successful.

...output omitted...

* 1. Try to create a project. The operation should fail.
  2. [student@workstation ~]$ **oc new-project test**

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

1. Create a project and add project administration privileges to the leader user.
   1. Log in as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp**

Login successful.

...output omitted...

* 1. Create the auth-rbac project.

[student@workstation ~]$ **oc new-project auth-rbac**

Now using project "auth-rbac" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Grant project administration privileges to the leader user on the auth-rbac project.

[student@workstation ~]$ **oc policy add-role-to-user admin leader**

clusterrole.rbac.authorization.k8s.io/admin added: "leader"

1. Create the dev-group and qa-group groups and add their respective members.
   1. Create a group named dev-group.

[student@workstation ~]$ **oc adm groups new dev-group**

group.user.openshift.io/dev-group created

* 1. Add the developer user to the group that you created in the previous step.

[student@workstation ~]$ **oc adm groups add-users dev-group developer**

group.user.openshift.io/dev-group added: "developer"

* 1. Create a second group named qa-group.

[student@workstation ~]$ **oc adm groups new qa-group**

group.user.openshift.io/qa-group created

* 1. Add the qa-engineer user to the group that you created in the previous step.

[student@workstation ~]$ **oc adm groups add-users qa-group qa-engineer**

group.user.openshift.io/qa-group added: "qa-engineer"

* 1. Review all existing OpenShift groups to verify that they have the correct members.

[student@workstation ~]$ **oc get groups**

NAME USERS

Default SMB Group

admins admin

**dev-group developer**

developer

editors

ocpadmins admin

ocpdevs developer

**qa-group qa-engineer**

### NOTE

The lab environment already contains groups from the lab LDAP directory.

1. As the leader user, assign write privileges for dev-group and read privileges for qa-group to the auth-rbac project.
   1. Log in as the leader user.

[student@workstation ~]$ **oc login -u leader -p redhat**

Login successful.

...output omitted...

Using project "auth-rbac".

* 1. Add write privileges to the dev-group group on the auth-rbac project.

[student@workstation ~]$ **oc policy add-role-to-group edit dev-group**

clusterrole.rbac.authorization.k8s.io/edit added: "dev-group"

* 1. Add read privileges to the qa-group group on the auth-rbac project.

[student@workstation ~]$ **oc policy add-role-to-group view qa-group**

clusterrole.rbac.authorization.k8s.io/view added: "qa-group"

* 1. Review all role bindings on the auth-rbac project to verify that they assign roles to the correct groups and users. The following output omits default role bindings that OpenShift assigns to service accounts.

[student@workstation ~]$ **oc get rolebindings -o wide | grep -v '^system:'**

NAME ROLE AGE USERS GROUPS SERVICEACCOUNTS

admin ClusterRole/admin 60s admin

admin-0 ClusterRole/admin 45s leader

edit ClusterRole/edit 30s dev-group

view ClusterRole/view 15s qa-group

1. As the developer user, deploy an Apache HTTP Server to prove that the developer user has write privileges in the project. Also try to grant write privileges to the qa-engineer user to prove that the developer user has no project administration privileges.
   1. Log in as the developer user.

[student@workstation ~]$ **oc login -u developer -p developer**

Login successful.

...output omitted...

Using project "auth-rbac".

* 1. Deploy an Apache HTTP Server by using the standard image stream from OpenShift.

[student@workstation ~]$ **oc new-app --name httpd httpd:2.4**

...output omitted...

--> Creating resources ...

deployment.apps "httpd" created

service "httpd" created

--> Success

...output omitted...

### NOTE

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

* 1. Try to grant write privileges to the qa-engineer user. The operation should fail.

[student@workstation ~]$ **oc policy add-role-to-user edit qa-engineer**

Error from server (Forbidden): rolebindings.rbac.authorization.k8s.io is forbidden: User "developer" cannot list resource "rolebindings" in API group "rbac.authorization.k8s.io" in the namespace "auth-rbac"

1. Verify that the qa-engineer user can view objects in the auth-rbac project, but not modify anything.
   1. Log in as the qa-engineer user.

[student@workstation ~]$ **oc login -u qa-engineer -p redhat**

Login successful.

...output omitted...

Using project "auth-rbac".

* 1. Attempt to scale the httpd application. The operation should fail.

[student@workstation ~]$ **oc scale deployment httpd --replicas 3**

Error from server (Forbidden): deployments.apps "httpd" is forbidden: User "qa-engineer" cannot patch resource "deployments/scale" in API group "apps" in the namespace "auth-rbac"

1. Restore project creation privileges to all users.
   1. Log in as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp**

Login successful.

...output omitted...

* 1. Restore project creation privileges for all users by re-creating the self-provisioners cluster role binding that the OpenShift installer created.

[student@workstation ~]$ **oc adm policy add-cluster-role-to-group \**

**--rolebinding-name self-provisioners \**

**self-provisioner system:authenticated:oauth**

Warning: Group 'system:authenticated:oauth' not found

clusterrole.rbac.authorization.k8s.io/self-provisioner added: "system:authenticated:oauth"

### NOTE

You can safely ignore the warning that the group was not found.

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish auth-rbac**

## **Lab: Authentication and Authorization**

Configure the HTPasswd identity provider, create groups, and assign roles to users and groups.

**Outcomes**

* Create users and passwords for HTPasswd authentication.
* Configure the identity provider for HTPasswd authentication.
* Assign cluster administration rights to users.
* Remove the ability to create projects at the cluster level.
* Create groups and add users to groups.
* Manage user privileges in projects by granting privileges to groups.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start auth-review**

The command ensures that the cluster API is reachable, and that the cluster uses the initial lab authentication settings.

**Instructions**

1. Update the existing ~/DO280/labs/auth-review/tmp\_users HTPasswd authentication file to remove the analyst user. Ensure that the tester and leader users in the file use the L@bR3v!ew password. Add two entries to the file for the new\_admin and new\_developer users. Use the L@bR3v!ew password for each new user.
   1. Remove the analyst user from the ~/DO280/labs/auth-review/tmp\_users HTPasswd authentication file.
   2. [student@workstation ~]$ **htpasswd -D ~/DO280/labs/auth-review/tmp\_users analyst**

Deleting password for user analyst

* 1. Update the entries for the tester and leader users to use the L@bR3v!ew password. Add entries for the new\_admin and new\_developer users with the L@bR3v!ew password.

[student@workstation ~]$ **for NAME in tester leader new\_admin new\_developer ; \**

**do \**

**htpasswd -b ~/DO280/labs/auth-review/tmp\_users ${NAME} 'L@bR3v!ew' ; \**

**done**

Updating password for user tester

Updating password for user leader

Adding password for user new\_admin

Adding password for user new\_developer

* 1. Review the contents of the ~/DO280/labs/auth-review/tmp\_users file. This file does not contain a line for the analyst user. The file includes two new entries with hashed passwords for the new\_admin and new\_developer users.

[student@workstation ~]$ **cat ~/DO280/labs/auth-review/tmp\_users**

tester:$apr1$EyWSDib4$uLoUMpwohNWUrU5L5ogkB/

leader:$apr1$/O8SyNdp$gjr.P7FMJbK2IebFU0QQn/

**new\_admin:$apr1$M5WHRPR2$GbGDkTK8QTrW2S/f2/1Kt1**

**new\_developer:$apr1$dXdG8tWd$N8HA0SUe3TbqAhI049gOH0**

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Log in to your OpenShift cluster as the admin user with the redhatocp password. Configure your cluster to use the HTPasswd identity provider by using the defined user names and passwords in the ~/DO280/labs/auth-review/tmp\_users file. For grading, use the auth-review name for the secret.
   1. Log in to the cluster as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Create an auth-review secret by using the ~/DO280/labs/auth-review/tmp\_users file.

[student@workstation ~]$ **oc create secret generic auth-review \**

**--from-file htpasswd=/home/student/DO280/labs/auth-review/tmp\_users \**

**-n openshift-config**

secret/auth-review created

* 1. Export the existing OAuth resource to ~/DO280/labs/auth-review/oauth.yaml.

[student@workstation ~]$ **oc get oauth cluster \**

**-o yaml > ~/DO280/labs/auth-review/oauth.yaml**

* 1. Edit the ~/DO280/labs/auth-review/oauth.yaml file to add an identity provider by including the lines from the following example that are displayed in bold. Ensure that the htpasswd, mappingMethod, name, and type strings are at the same indentation level.

apiVersion: config.openshift.io/v1

kind: OAuth

...output omitted...

**spec:**

**identityProviders:**

- ldap:

...output omitted...

type: LDAP

**- htpasswd:**

**fileData:**

**name: auth-review**

**mappingMethod: claim**

**name: htpasswd**

**type: HTPasswd**

### NOTE

For convenience, the ~/DO280/solutions/auth-review/oauth.yaml file contains a minimal version of the OAuth configuration with the specified customizations.

* 1. Apply the customized resource that you defined in the previous step.

[student@workstation ~]$ **oc replace -f ~/DO280/labs/auth-review/oauth.yaml**

oauth.config.openshift.io/cluster replaced

* 1. A successful update to the oauth/cluster resource re-creates the oauth-openshift pods in the openshift-authentication namespace.

[student@workstation ~]$ **watch oc get pods -n openshift-authentication**

Wait until the new oauth-openshift pods are ready and running, and the previous pods have terminated.

Every 2.0s: oc get pods -n openshift-authentication ...

NAME READY STATUS RESTARTS AGE

oauth-openshift-68d6f666fd-z746p **1/1 Running** 0 42s

Press **Ctrl**+**C** to exit the watch command.

### NOTE

Pods in the openshift-authentication namespace redeploy when the oc replace command succeeds.

In this exercise, changes to authentication might require a few minutes to apply.

You can examine the status of pods and deployments in the openshift-authentication namespace to monitor the authentication status. You can also examine the authentication cluster operator for further status information.

Provided that the previously created secret was created correctly, you can log in by using the HTPasswd identity provider.

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Make the new\_admin user a cluster administrator. Log in as both the new\_admin and new\_developer users to verify HTPasswd user configuration and cluster privileges.
   1. Assign the new\_admin user the cluster-admin role.

[student@workstation ~]$ **oc adm policy add-cluster-role-to-user \**

**cluster-admin new\_admin**

Warning: User 'new\_admin' not found

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "new\_admin"

### NOTE

You can safely ignore the warning that the new\_admin user is not found.

* 1. Log in to the cluster as the new\_admin user to verify that HTPasswd authentication is configured correctly.

[student@workstation ~]$ **oc login -u new\_admin -p 'L@bR3v!ew'**

Login successful.

...output omitted...

* 1. Use the oc get nodes command to verify that the new\_admin user has the cluster-admin role. The names of the nodes from your cluster might be different.

[student@workstation ~]$ **oc get nodes**

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master,worker 14d v1.27.6+f67aeb3

* 1. Log in to the cluster as the new\_developer user to verify that the HTPasswd authentication is configured correctly.

[student@workstation ~]$ **oc login -u new\_developer -p 'L@bR3v!ew'**

Login successful.

...output omitted...

* 1. Use the oc get nodes command to verify that the new\_developer user does not have cluster administration privileges.

[student@workstation ~]$ **oc get nodes**

Error from server (Forbidden): nodes is forbidden: User "new\_developer" cannot list resource "nodes" in API group "" at the cluster scope

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. As the new\_admin user, prevent users from creating projects in the cluster.
   1. Log in to the cluster as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p 'L@bR3v!ew'**

Login successful.

...output omitted...

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

[student@workstation ~]$ **oc adm policy remove-cluster-role-from-group \**

**self-provisioner system:authenticated:oauth**

Warning: Your changes may get lost whenever a master is restarted, unless you prevent reconciliation of this rolebinding using the following command: oc annotate clusterrolebinding.rbac self-provisioners 'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite

clusterrole.rbac.authorization.k8s.io/self-provisioner removed: "system:authenticated:oauth"

### NOTE

You can safely ignore the warning about your changes being lost.

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create a managers group, and add the leader user to the group. Grant project creation privileges to the managers group. As the leader user, create the auth-review project.
   1. Create a managers group.

[student@workstation ~]$ **oc adm groups new managers**

group.user.openshift.io/managers created

* 1. Add the leader user to the managers group.

[student@workstation ~]$ **oc adm groups add-users managers leader**

group.user.openshift.io/managers added: "leader"

* 1. Assign the self-provisioner cluster role to the managers group.

[student@workstation ~]$ **oc adm policy add-cluster-role-to-group \**

**self-provisioner managers**

clusterrole.rbac.authorization.k8s.io/self-provisioner added: "managers"

* 1. As the leader user, create the auth-review project.

[student@workstation ~]$ **oc login -u leader -p 'L@bR3v!ew'**

Login successful.

...output omitted...

The user who creates a project is automatically assigned the admin role on the project.

[student@workstation ~]$ **oc new-project auth-review**

Now using project "auth-review" on server "https://api.ocp4.example.com:6443".

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create a developers group and grant edit privileges on the auth-review project. Add the new\_developer user to the group.
   1. Log in to the cluster as the new\_admin user.

[student@workstation ~]$ **oc login -u new\_admin -p 'L@bR3v!ew'**

Login successful.

...output omitted...

* 1. Create a developers group.

[student@workstation ~]$ **oc adm groups new developers**

group.user.openshift.io/developers created

* 1. Add the new\_developer user to the developers group.

[student@workstation ~]$ **oc adm groups add-users developers new\_developer**

group.user.openshift.io/developers added: "new\_developer"

* 1. Grant edit privileges to the developers group on the auth-review project.

[student@workstation ~]$ **oc policy add-role-to-group edit developers**

clusterrole.rbac.authorization.k8s.io/edit added: "developers"

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create a qa group and grant view privileges on the auth-review project. Add the tester user to the group.
   1. Create a qa group.

[student@workstation ~]$ **oc adm groups new qa**

group.user.openshift.io/qa created

* 1. Add the tester user to the qa group.

[student@workstation ~]$ **oc adm groups add-users qa tester**

group.user.openshift.io/qa added: "tester"

* 1. Grant view privileges to the qa group on the auth-review project.

[student@workstation ~]$ **oc policy add-role-to-group view qa**

clusterrole.rbac.authorization.k8s.io/view added: "qa"

1. [Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade auth-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish auth-review**

# Chapter 4.  Network Security

[Protect External Traffic with TLS](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04)

[Guided Exercise: Protect External Traffic with TLS](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s02)

[Configure Network Policies](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s03)

[Guided Exercise: Configure Network Policies](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s04)

[Protect Internal Traffic with TLS](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s05)

[Guided Exercise: Protect Internal Traffic with TLS](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s06)

[Lab: Network Security](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s07)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch04s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Protect network traffic between applications inside and outside the cluster. |
| **Objectives** | * Allow and protect network connections to applications inside an OpenShift cluster. * Restrict network traffic between projects and pods. * Configure and use automatic service certificates. |
| **Sections** | * Protect External Traffic with TLS (and Guided Exercise) * Configure Network Policies (and Guided Exercise) * Protect Internal Traffic with TLS (and Guided Exercise) |
| **Lab** | * Network Security |

## **Protect External Traffic with TLS**

### **Objectives**

* Allow and protect network connections to applications inside an OpenShift cluster.

### **Accessing Applications from External Networks**

OpenShift Container Platform offers many ways to expose your applications to external networks. You can expose HTTP and HTTPS traffic, TCP applications, and also non-TCP traffic. Some of these methods are service types, such as NodePort or load balancer, whereas others use their own API resource, such as Ingress and Route.

With OpenShift routes, you can expose your applications to external networks, to reach the applications with a unique, publicly accessible hostname. Routes rely on a router plug-in to redirect the traffic from the public IP to pods.

The following diagram shows how a route exposes an application that runs as pods in your cluster:

|  |
| --- |
|  |

Figure 4.1: Using routes to expose applications

### NOTE

For performance reasons, routers send requests directly to pods based on service configuration.

The dotted line in the diagram indicates this implementation. The router accesses the pods through the services network.

### **Securing Routes**

Routes can be either secured or unsecured. Secure routes support several types of transport layer security (TLS) termination to serve certificates to the client. Unsecured routes are the simplest to configure, because they require no key or certificates. By contrast, secured routes encrypt traffic to and from the pods.

A secured route specifies the TLS termination of the route. The following termination types are available:

**OpenShift Secure Routes**

**Edge**

With edge termination, TLS termination occurs at the router, before the traffic is routed to the pods. The router serves the TLS certificates, so you must configure them into the route; otherwise, OpenShift assigns its own certificate to the router for TLS termination. Because TLS is terminated at the router, connections from the router to the endpoints over the internal network are not encrypted.

**Passthrough**

With passthrough termination, encrypted traffic is sent straight to the destination pod without TLS termination from the router. In this mode, the application is responsible for serving certificates for the traffic. Passthrough is currently the only method that supports mutual authentication between the application and a client that accesses it.

**Re-encryption**

Re-encryption is a variation on edge termination, whereby the router terminates TLS with a certificate, and then re-encrypts its connection to the endpoint, which might have a different certificate. Therefore, the full path of the connection is encrypted, even over the internal network. The router uses health checks to determine the authenticity of the host.

#### **Securing Applications with Edge Routes**

Before creating a secure route, you need a TLS certificate. The following command shows how to create a secure edge route with a TLS certificate:

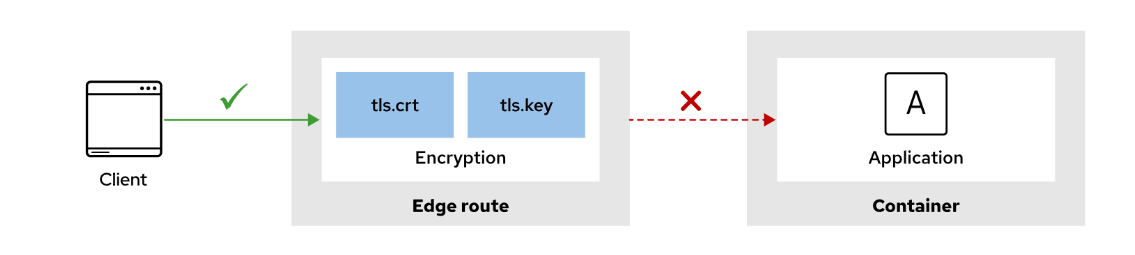
[user@host ~]$ **oc create route edge \**

**--service api-frontend --hostname api.apps.acme.com \**

**--key api.key --cert api.crt**

|  |  |
| --- | --- |
|  | The --key option requires the certificate private key. |
|  | The --cert option requires the signed certificate. |

When using a route in edge mode, the traffic between the client and the router is encrypted, but traffic between the router and the application is not encrypted:



|  |
| --- |
|  |

Figure 4.2: Securing applications with edge routes

### NOTE

Network policies can help you to protect the internal traffic between your applications or between projects.

#### **Securing Applications with Passthrough Routes**

The previous example demonstrates how to create an edge route, which means an OpenShift route that presents a certificate at the edge. Passthrough routes offer a secure alternative, because the application exposes its TLS certificate. As such, the traffic is encrypted between the client and the application.

To create a passthrough route, you need a certificate and a way for your application to access it. The best way to provide the certificate is by using OpenShift TLS secrets. Secrets are exposed via a mount point into the container.

The following diagram shows how you can mount a secret resource in your container. The application is then able to access your certificate.

|  |
| --- |
|  |

Figure 4.3: Securing applications with passthrough routes

#### **Securing Applications with Re-encrypt Routes**

Re-encrypt routes provide end-to-end encryption. First, re-encrypt routes terminate the encryption between an external client and the router. This encryption uses a certificate with a fully qualified domain name (FQDN) that is trusted by the client, such as the my-app.example.com hostname.

Then, the router re-encrypts the connection when accessing an internal cluster service. This internal communication requires a certificate for the target service with an OpenShift FQDN, such as the my-app.namespace.svc.cluster.local hostname.

The certificates for internal TLS connections require a public key infrastructure (PKI) to sign the certificate. With an OpenShift service certificate, you can mount a secret that contains a certificate and key pair into an application. This feature uses the OpenShift PKI to generate the certificate and key into a service-specific secret.

### REFERENCES

For more information about how to manage routes, refer to the Configuring Routes chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/networking/index#configuring-routes](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index" \l "configuring-routes" \t "_top)

For more information about how to configure ingress cluster traffic, refer to the Configuring Ingress Cluster Traffic chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/networking/index#configuring-ingress-cluster-traffic](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index" \l "configuring-ingress-cluster-traffic" \t "_top)

[Self-Serviced End-to-end Encryption Approaches for Applications Deployed in OpenShift](https://cloud.redhat.com/blog/self-serviced-end-to-end-encryption-approaches-for-applications-deployed-in-openshift" \t "_top)

## **Guided Exercise: Protect External Traffic with TLS**

Expose an application that is secured by TLS certificates.

**Outcomes**

* Deploy an application and create an unencrypted route for it.
* Create an OpenShift edge route with encryption.
* Update an OpenShift deployment to support a new version of the application.
* Create an OpenShift TLS secret and mount it to your application.
* Verify that the communication to the application is encrypted.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

The command ensures that the cluster API is reachable, and creates the network-ingress OpenShift project. The command also gives the developer user edit access on the project.

[student@workstation ~]$ **lab start network-ingress**

**Instructions**

As an application developer, you are ready to deploy your application in OpenShift. In this activity, you deploy two versions of the application: one that is exposed over unencrypted traffic (HTTP), and one that is exposed over secure traffic (HTTPS).

The container image, which is accessible at https://registry.ocp4.example.com:8443/redhattraining/todo-angular, has two tags: v1.1, which is the insecure version of the application, and v1.2, which is the secure version. Your organization uses its own certificate authority (CA) that can sign certificates for the following domains:

* \*.apps.ocp4.example.com
* \*.ocp4.example.com

The CA certificate is accessible at ~/DO280/labs/network-ingress/certs/training-CA.pem. The passphrase.txt file contains a unique password that protects the CA key. The certs directory also contains the CA key.

1. Log in to the OpenShift cluster and create the network-ingress project.
   1. Log in to the cluster as the developer user.

[student@workstation ~]$ **oc login -u developer -p developer \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Create the network-ingress project.

[student@workstation ~]$ **oc new-project network-ingress**

Now using project "network-ingress" on server "https://api.ocp4.example.com:6443".

...output omitted...

1. The OpenShift deployment file for the application is accessible at ~/DO280/labs/network-ingress/todo-app-v1.yaml. The deployment points to registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.1, which is the initial and unencrypted version of the application. The file defines the todo-http service that points to the application pod.

Create the application and expose the service.

* 1. Use the oc create command to deploy the application in the network-ingress OpenShift project.

[student@workstation ~]$ **oc create -f \**

**~/DO280/labs/network-ingress/todo-app-v1.yaml**

deployment.apps/todo-http created

service/todo-http created

* 1. Wait a few minutes, so that the application can start, and then review the resources in the project.

[student@workstation ~]$ **oc status**

...output omitted...

In project network-ingress on server https://api.ocp4.example.com:6443

svc/todo-http - 172.30.247.75:80 -> 8080

deployment/todo-http deploys registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.1

deployment #1 running for 16 seconds - 1 pod

...output omitted...

* 1. Run the oc expose command to create a route for accessing the application. Give the route a hostname of todo-http.apps.ocp4.example.com.

[student@workstation ~]$ **oc expose svc todo-http \**

**--hostname todo-http.apps.ocp4.example.com**

route.route.openshift.io/todo-http exposed

* 1. Retrieve the name of the route and copy it to the clipboard.

[student@workstation ~]$ **oc get routes**

NAME HOST/PORT PATH SERVICES PORT ...

todo-http **todo-http.apps.ocp4.example.com** todo-http 8080 ...

* 1. On the workstation machine, open Firefox and access the application URL. Confirm that you can see the application.
     + http://todo-http.apps.ocp4.example.com
  2. Open a new terminal tab and run the tcpdump command with the following options to intercept the traffic on port 80:
     + -i eth0 intercepts traffic on the main interface.
     + -A strips the headers and prints the packets in ASCII format.
     + -n disables DNS resolution.
     + port 80 is the port of the application.

Optionally, use the grep command to filter on JavaScript resources.

Start by retrieving the name of the main interface, whose IP is 172.25.250.9.

[student@workstation ~]$ **ip addr | grep 172.25.250.9**

inet 172.25.250.9/24 brd 172.25.250.255 scope global noprefixroute **eth0**

[student@workstation ~]$ **sudo tcpdump -i eth0 -A -n port 80 | grep "angular"**

### NOTE

The full command is available at ~/DO280/labs/network-ingress/tcpdump-command.txt.

* 1. On Firefox, refresh the page and notice the activity in the terminal. Press **Ctrl**+**C** to stop the capture.

...output omitted...

<script type="text/javascript" src="assets/js/libs/angular/angular.min.js">

<script type="text/javascript" src="assets/js/libs/angular/angular-route.min.js">

<script type="text/javascript" src="assets/js/libs/angular/angular-animate.min.js">

...output omitted...

1. Create a secure edge route. Edge certificates encrypt the traffic between the client and the router, but leave the traffic between the router and the service unencrypted. OpenShift generates its own certificate that it signs with its CA.

In later steps, you extract the CA to ensure that the route certificate is signed.

* 1. Go to ~/DO280/labs/network-ingress and run the oc create route command to define the new route.

Give the route a hostname of todo-https.apps.ocp4.example.com.

[student@workstation ~]$ **cd ~/DO280/labs/network-ingress**

[student@workstation network-ingress]$ **oc create route edge todo-https \**

**--service todo-http \**

**--hostname todo-https.apps.ocp4.example.com**

route.route.openshift.io/todo-https created

* 1. To test the route and read the certificate, open Firefox and access the application URL.
     + https://todo-https.apps.ocp4.example.com

Click the **padlock**, and then click the arrow next to **Connection secure**.

A screenshot of a computer

Description automatically generated

Firefox displays a message that the connection is verified by a certificate issuer that Mozilla does not recognize. This message is displayed because the route signed certificate comes from an internal CA that is installed on the classroom OS. This CA, although not recognized by Mozilla, is valid for the lab environment purposes. If your organization uses a custom public key infrastructure (PKI), then you might see the same message.

Click **More Information** to display the page information window.

A screenshot of a computer

Description automatically generated

Click **View Certificate** to display the certificate information.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Locate the CN entry to see that the OpenShift ingress operator created the certificate with its own CA.

A screenshot of a computer

Description automatically generated

* 1. From the terminal, use the curl command with the -I and -v options to retrieve the connection headers.

The Server certificate section shows some information about the certificate. The alternative name matches the name of the route. The output indicates that the remote certificate is trusted because it matches the CA.

[student@workstation network-ingress]$ **curl -I -v \**

**https://todo-https.apps.ocp4.example.com**

...output omitted...

\* Server certificate:

\* subject: O=EXAMPLE.COM; CN=.api.ocp4.example.com

\* start date: May 10 11:18:41 2021 GMT

\* expire date: May 10 11:18:41 2026 GMT

\* subjectAltName: host "**todo-https.apps.ocp4.example.com**" matched cert's "\*.apps.ocp4.example.com"

\* issuer: O=EXAMPLE.COM; CN=Red Hat Training Certificate Authority

\* **SSL certificate verify ok.**

...output omitted...

* 1. Although the traffic is encrypted at the edge with a certificate, you can still access the insecure traffic at the service level, because the pod behind the service does not offer an encrypted route.

Retrieve the IP address of the todo-http service.

[student@workstation network-ingress]$ **oc get svc todo-http \**

**-o jsonpath="{.spec.clusterIP}{'\n'}"**

172.30.102.29

* 1. Create a debug pod in the todo-http deployment. Use the Red Hat Universal Base Image (UBI), which contains tools to interact with containers.
  2. [student@workstation network-ingress]$ **oc debug -t deployment/todo-http \**
  3. **--image registry.ocp4.example.com:8443/ubi8/ubi:8.4**
  4. Starting pod/todo-http-debug ...
  5. Pod IP: 10.131.0.255
  6. If you don't see a command prompt, try pressing enter.

sh-4.4$

* 1. From the debug pod, use the curl command to access the service over HTTP. Replace the IP address with the one that you obtained in a previous step.

The output indicates that the application is available over HTTP.

sh-4.4$ **curl -v *172.30.102.29***

\* Rebuilt URL to: 172.30.102.29/

\* Trying 172.30.102.29...

\* TCP\_NODELAY set

\* Connected to 172.30.102.29 (172.30.102.29) port 80 (#0)

> **GET / HTTP/1.1**

> Host: 172.30.102.29

> User-Agent: curl/7.61.1

> Accept: \*/\*

>

< HTTP/1.1 200 OK

...output omitted...

* 1. Exit the debug pod.
  2. sh-4.4$ **exit**

Removing debug pod ...

* 1. Delete the edge route. In the following steps, you define the passthrough route.
  2. [student@workstation network-ingress]$ **oc delete route todo-https**

route.route.openshift.io "todo-https" deleted

1. Generate TLS certificates for the application.

In the following steps, you generate a CA-signed certificate that you attach as a secret to the pod. You then configure a secure route in passthrough mode and let the application expose that certificate.

* 1. Go to the ~/DO280/labs/network-ingress/certs directory and list the files.

[student@workstation network-ingress]$ **cd certs**

[student@workstation certs]$ **ls -l**

total 20

-rw-rw-r--. 1 student student 604 Nov 29 17:35 openssl-commands.txt

-rw-r--r--. 1 student student 33 Nov 29 17:35 passphrase.txt

-rw-r--r--. 1 student student 1743 Nov 29 17:35 training-CA.key

-rw-r--r--. 1 student student 1363 Nov 29 17:35 training-CA.pem

-rw-r--r--. 1 student student 406 Nov 29 17:35 training.ext

* 1. Generate the private key for your CA-signed certificate.

### NOTE

The following commands for generating a signed certificate are all available in the ~/DO280/labs/network-ingress/certs/openssl-commands.txt file.

[student@workstation certs]$ **openssl genrsa -out training.key 4096**

* 1. Generate the certificate signing request (CSR) for the todo-https.apps.ocp4.example.com hostname.

[student@workstation certs]$ **openssl req -new \**

**-key training.key -out training.csr \**

**-subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\**

**CN=todo-https.apps.ocp4.example.com"**

### WARNING

Type the request **subject** on one line. Alternatively, remove the -subj option and its content. Without the -subj option, the openssl command prompts you for the values; indicate a common name (CN) of todo-https.apps.ocp4.example.com.

* 1. Finally, generate the signed certificate. Notice the use of the -CA and -CAkey options for signing the certificate against the CA. Use the -passin option to reuse the password of the CA. Use the extfile option to define a Subject Alternative Name (SAN).

[student@workstation certs]$ **openssl x509 -req -in training.csr \**

**-passin file:passphrase.txt \**

**-CA training-CA.pem -CAkey training-CA.key -CAcreateserial \**

**-out training.crt -days 1825 -sha256 -extfile training.ext**

Certificate request self-signature ok

subject=C = US, ST = North Carolina, L = Raleigh, O = Red Hat, CN = todo-https.apps.ocp4.example.com

* 1. Ensure that the newly created certificate and key are present in the current directory.

[student@workstation certs]$ **ls -lrt**

total 36

-rw-r--r--. 1 student student 599 Jul 31 09:35 openssl-commands.txt

-rw-r--r--. 1 student student 33 Aug 3 12:38 passphrase.txt

-rw-r--r--. 1 student student 352 Aug 3 12:38 training.ext

-rw-------. 1 student student 1743 Aug 3 12:38 training-CA.key

-rw-r--r--. 1 student student 1334 Aug 3 12:38 training-CA.pem

-rw-------. 1 student student 1675 Aug 3 13:38 **training.key**

-rw-rw-r--. 1 student student 1017 Aug 3 13:39 training.csr

-rw-rw-r--. 1 student student 41 Aug 3 13:40 training-CA.srl

-rw-rw-r--. 1 student student 1399 Aug 3 13:40 **training.crt**

* 1. Return to the network-ingress directory. This step is important, because the next step involves creating a route that uses the self-signed certificate.

[student@workstation certs]$ **cd ~/DO280/labs/network-ingress**

1. Deploy a new version of your application.

The new version of the application expects a certificate and a key inside the container at /usr/local/etc/ssl/certs. The web server in that version is configured with SSL support. Create a secret to import the certificate from the workstation machine. In a later step, the application deployment requests that secret and exposes its content to the container at /usr/local/etc/ssl/certs.

* 1. Create a tls OpenShift secret named todo-certs. Use the --cert and --key options to embed the TLS certificates. Use training.crt as the certificate, and training.key as the key.

[student@workstation network-ingress]$ **oc create secret tls todo-certs \**

**--cert certs/training.crt --key certs/training.key**

secret/todo-certs created

* 1. The deployment file at ~/DO280/labs/network-ingress/todo-app-v2.yaml points to version 2 of the container image. Examine how the new version of the application is configured to support SSL certificates.

[student@workstation network-ingress]$ **cat todo-app-v2.yaml**

apiVersion: apps/v1

kind: Deployment

...output omitted...

volumeMounts:

- name: **tls-certs**

readOnly: true

mountPath: **/usr/local/etc/ssl/certs**

...output omitted...

volumes:

- name: **tls-certs**

secret:

secretName: **todo-certs**

---

apiVersion: v1

kind: Service

...output omitted...

ports:

- name: https

port: **8443**

protocol: TCP

targetPort: 8443

...output omitted...

The todo-certs secret with the SSL certificate is mounted in the container in the /usr/local/etc/ssl/certs directory to enable TLS for the application. Additionally, the todo-app-v2 deployment changes the service to include port 8443.

* 1. Run the oc create command to create a deployment that uses that image.

[student@workstation network-ingress]$ **oc create -f todo-app-v2.yaml**

deployment.apps/todo-https created

service/todo-https created

* 1. Wait a couple of minutes to ensure that the application pod is running. Use the oc set volumes command to review the volumes that are mounted inside the pod.

[student@workstation network-ingress]$ **oc set volumes deployment/todo-https**

todo-https

secret/todo-certs as tls-certs

mounted at /usr/local/etc/ssl/certs

1. Create the secure route.
   1. Run the oc create route command to define the new route.

Give the route a hostname of todo-https.apps.ocp4.example.com.

[student@workstation network-ingress]$ **oc create route passthrough todo-https \**

**--service todo-https --port 8443 \**

**--hostname todo-https.apps.ocp4.example.com**

route.route.openshift.io/todo-https created

* 1. Use the curl command in verbose mode to test the route and to read the certificate. Use the --cacert option to pass the CA certificate to the curl command.

The output indicates a match between the certificate chain and the application certificate. This match indicates that the OpenShift router forwards only packets that are encrypted by the application web server certificate.

[student@workstation network-ingress]$ **curl -vv -I \**

**--cacert certs/training-CA.pem \**

**https://todo-https.apps.ocp4.example.com**

...output omitted...

\* Server certificate:

\* **subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=todo-https.apps.ocp4.example.com**

\* start date: Jun 15 01:53:30 2021 GMT

\* expire date: Jun 14 01:53:30 2026 GMT

\* subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's "\*.apps.ocp4.example.com"

\* issuer: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=ocp4.example.com

\* **SSL certificate verify ok.**

...output omitted...

1. Create a debug pod to further confirm proper encryption at the service level.
   1. Retrieve the IP address of the todo-https service.

[student@workstation network-ingress]$ **oc get svc todo-https \**

**-o jsonpath="{.spec.clusterIP}{'\n'}"**

172.30.121.154

* 1. Create a debug pod in the todo-https deployment with the Red Hat UBI container image.

[student@workstation network-ingress]$ **oc debug -t deployment/todo-https \**

**--image registry.ocp4.example.com:8443/ubi8/ubi:8.4**

Starting pod/todo-https-debug ...

Pod IP: 10.128.2.129

If you don't see a command prompt, try pressing enter.

sh-4.4$

* 1. From the debug pod, use the curl command to access the service over HTTP. Replace the IP address with the one that you obtained in a previous step.

The output indicates that the application is not available over HTTP, and the web server redirects you to the secure version.

sh-4.4$ **curl -I *http://172.30.121.154***

**HTTP/1.1 301 Moved Permanently**

Server: nginx/1.14.1

Date: Tue, 15 Jun 2021 02:01:19 GMT

Content-Type: text/html

Connection: keep-alive

**Location:** **https://172.30.121.154:8443/**

* 1. Finally, access the application over HTTPS. Use the -k option, because the container does not have access to the CA certificate.

sh-4.4$ **curl -s -k *https://172.30.121.154:8443* | head -n5**

<!DOCTYPE html>

<html lang="en" ng-app="todoItemsApp" ng-controller="appCtl">

<head>

<meta charset="utf-8">

<title>ToDo app</title>

* 1. Exit the debug pod.

sh-4.4$ **exit**

Removing debug pod ...

1. Clean up the exercise directory and project.
   1. Change to the home directory.

[student@workstation network-ingress]$ **cd**

* 1. Delete the network-ingress project.
  2. [student@workstation ~]$ **oc delete project network-ingress**

project.project.openshift.io "network-ingress" deleted

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish network-ingress**

## **Configure Network Policies**

### **Objectives**

* Restrict network traffic between projects and pods.

### **Managing Network Policies in OpenShift**

With network policies, you can configure isolation policies for individual pods. Network policies do not require administrative privileges, and give developers more control over the applications in their projects. You can use network policies to create logical zones in the SDN that map to your organization network zones. The benefit of this approach is that the location of running pods becomes irrelevant, because with network policies, you can separate traffic regardless of where it originates.

In contrast to traditional firewalls, Kubernetes network policies control network traffic between pods by using labels instead of IP addresses. To manage network communication between pods in two namespaces, assign a label to the namespace that needs access to another namespace, and create a network policy that selects these labels. You can also use a network policy to select labels on individual pods to create ingress or egress rules. In network policies, use selectors under spec to assign which destination pods are affected by the policy, and selectors under spec.ingress to assign which source pods are allowed. The following command assigns the network=network-1 label to the network-1 namespace:

[user@host ~]$ **oc label namespace network-1 network=network-1**

The following examples describe network policies that allow communication between pods in the network-1 and network-2 namespaces:

* The following network policy applies to any pods with the deployment="product-catalog" label in the network-1 namespace. The network-2 namespace has the network=network-2 label. The policy allows TCP traffic over port 8080 from pods whose label is role="qa" in namespaces with the network=network-2 label.
* kind: NetworkPolicy
* apiVersion: networking.k8s.io/v1
* metadata:
* name: network-1-policy
* namespace: network-1
* spec:
* **podSelector:**
* matchLabels:
* deployment: **product-catalog**
* ingress:
* - from:
* - **namespaceSelector**:
* matchLabels:
* network: **network-2**
* **podSelector:**
* matchLabels:
* role: **qa**
* ports:
* - port: **8080**

protocol: **TCP**

|  |  |
| --- | --- |
|  | The top-level podSelector field is required and defines which pods use the network policy. If the podSelector is empty, then all pods in the namespace are matched. |
|  | The ingress field defines a list of ingress traffic rules to apply to the matched pods from the top-level podSelector field. |
|  | The from field defines a list of rules to match traffic from all sources. The selectors are not limited to the project in which the network policy is defined. |
|  | The ports field is a list of destination ports that allow traffic to reach the selected pods. |

* The following network policy allows traffic from any pods in namespaces with the network=network-1 label into any pods and ports in the network-2 namespace. This policy is less restrictive than the network-1 policy, because it does not restrict traffic from any pods in the network-1 namespace.
* kind: NetworkPolicy
* apiVersion: networking.k8s.io/v1
* metadata:
* name: network-2-policy
* namespace: network-2
* spec:
* **podSelector: {}**
* ingress:
* - from:
* - **namespaceSelector**:
* matchLabels:

network: **network-1**

### NOTE

Network policies are Kubernetes resources. As such, you can manage them with oc commands.

#### **Network Policies Between Projects**

One benefit of using network policies is to manage security between projects (tenants), which you cannot do with layer 2 technologies such as VLANs. With this approach, you can create tailored policies between projects to ensure that users can access only what they should (which conforms to the least privilege approach).

The fields in the network policy that take a list of objects can either be combined in the same object or can be listed as multiple objects. If combined, the conditions are combined with a logical AND. If separated in a list, the conditions are combined with a logical OR. With the logic options, you can create specific policy rules. The following examples highlight the differences that the syntax can make:

* This example combines the selectors into one rule, and thereby allows access only from pods with the app=mobile label in namespaces with the network=dev label. This sample shows a logical AND statement.
* ...output omitted...
* ingress:
* - from:
* - **namespaceSelector**:
* matchLabels:
* network: **dev**
* **podSelector:**
* matchLabels:

**app: mobile**

* By changing the podSelector field in the previous example to be an item in the from list, any pods in namespaces with the network=dev label or any pods with the app=mobile label from any namespace can reach the pods that match the top-level podSelector field. This sample shows a logical OR statement.
* ...output omitted...
* ingress:
* - from:
* - **namespaceSelector**:
* matchLabels:
* network: **dev**
* **- podSelector:**
* matchLabels:

app: **mobile**

#### **Deny-all Network Policies**

If a pod is matched by selectors in one or more network policies, then the pod accepts only connections that at least one of those network policies allows. A strict example is a policy to deny-all ingress traffic to pods in your project, including from other pods inside your project. An empty pod selector means that this policy applies to all pods in this project. The following policy blocks all traffic, because no ingress rules are defined. Traffic is blocked unless you also define an explicit policy that overrides this default behavior.

kind: NetworkPolicy

apiVersion: networking.k8s.io/v1

metadata:

name: default-deny

spec:

**podSelector: {}**

### IMPORTANT

If a pod does not match any network policies, then OpenShift does not restrict traffic to that pod. When creating an environment to allow network traffic only explicitly, you must include a deny-all policy.

#### **Allowing Access from OpenShift Cluster Services**

When you protect your pods by using network policies, OpenShift cluster services might need explicit policies to access pods. Several common scenarios require explicit policies, including the following ones:

* The router pods that enable access from outside the cluster by using ingress or route resources
* The monitoring service, if your application exposes metrics endpoints

The following policies allow ingress from OpenShift monitoring and ingress pods:

---

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-from-openshift-ingress

spec:

**podSelector: {}**

ingress:

- from:

- **namespaceSelector**:

matchLabels:

policy-group.network.openshift.io/ingress: ""

---

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-from-openshift-monitoring

spec:

**podSelector: {}**

ingress:

- from:

- **namespaceSelector**:

matchLabels:

network.openshift.io/policy-group: **monitoring**

### IMPORTANT

Network policies do not block traffic from pods that use host networking to pods in the same node.

For example, on a single-node cluster, a deny-all network policy does not prevent ingress pods that use the host network strategy from accessing application pods.

Inside a node, traffic from pods that use host networking is treated differently from traffic from other pods. Network policies control only internal traffic from pods that do not use host networking.

When traffic leaves a node, no such different treatment exists, and network policies control all traffic from other nodes.

For more information about this topic, refer to [Network Policies](https://kubernetes.io/docs/concepts/services-networking/network-policies/" \l "what-you-can-t-do-with-network-policies-at-least-not-yet" \t "_top)

### REFERENCES

For more information about network policy, refer to the Network Policy chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\_container\_platform/4.14/html-single/networking/index#network-policy](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index" \l "network-policy" \t "_top)

## **Guided Exercise: Configure Network Policies**

Create network policies and review pod isolation that these network policies created.

**Outcomes**

* Create network policies to control communication between pods.
* Verify that ingress traffic is limited to pods.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the environment is ready and downloads the necessary resource files for the exercise.

[student@workstation ~]$ **lab start network-policy**

**Instructions**

1. Log in to the OpenShift cluster and create the network-policy project.
   1. Log in to the cluster as the developer user with the developer password.

[student@workstation ~]$ **oc login -u developer -p developer \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Create the network-policy project.

[student@workstation ~]$ **oc new-project network-policy**

Now using project "network-policy" on server "https://api.ocp4.example.com:6443".

...output omitted...

1. Create two identical deployments named hello and test. Create a route to the hello deployment.
   1. Create the hello deployment that uses the registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 container image.

[student@workstation ~]$ **oc new-app --name hello \**

**--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0**

...output omitted...

--> Creating resources ...

imagestream.image.openshift.io "hello" created

deployment.apps "hello" created

service "hello" created

--> Success

...output omitted...

* 1. Create the test deployment that uses the registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 container image.

[student@workstation ~]$ **oc new-app --name test \**

**--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0**

...output omitted...

--> Creating resources ...

imagestream.image.openshift.io "test" created

deployment.apps "test" created

service "test" created

--> Success

...output omitted...

* 1. Use the oc expose command to create a route to the hello service.
  2. [student@workstation ~]$ **oc expose service hello**

route.route.openshift.io/hello exposed

1. Verify that the test pod can access the hello pod by using the oc rsh and curl commands.
   1. Open a second terminal and run the script at ~/DO280/labs/network-policy/display-project-info.sh. This script provides information about the pods, service, and route that are used in the rest of this exercise.

[student@workstation ~]$ **~/DO280/labs/network-policy/display-project-info.sh**

===================================================================

PROJECT: network-policy

POD NAME IP ADDRESS

hello-6c4984d949-g28c4 **10.8.0.13**

**test-c4d74c9d5-5pq9s** 10.8.0.14

SERVICE NAME CLUSTER-IP

hello **172.30.137.226**

test 172.30.159.119

ROUTE NAME HOSTNAME PORT

hello **hello-network-policy.apps.ocp4.example.com** 8080-tcp

===================================================================

* 1. Access the hello pod IP address from the test pod by using the oc rsh and curl commands.

[student@workstation ~]$ **oc rsh test-*c4d74c9d5-5pq9s* \**

**curl *10.8.0.13*:8080 | grep Hello**

<h1>Hello, world from nginx!</h1>

* 1. Access the hello service IP address from the test pod by using the oc rsh and curl commands.

[student@workstation ~]$ **oc rsh test-*c4d74c9d5-5pq9s* \**

**curl *172.30.137.226*:8080 | grep Hello**

<h1>Hello, world from nginx!</h1>

* 1. Access the hello route hostname by using the curl command.

[student@workstation ~]$ **curl -s hello-network-policy.apps.ocp4.example.com | \**

**grep Hello**

<h1>Hello, world from nginx!</h1>

1. Create a project named different-namespace that contains a deployment named sample-app.
   1. Create the different-namespace project.

[student@workstation ~]$ **oc new-project different-namespace**

Now using project "different-namespace" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Create the sample-app deployment from the registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 image. The web app listens on port 8080.

[student@workstation ~]$ **oc new-app --name sample-app \**

**--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0**

...output omitted...

--> Creating resources ...

imagestream.image.openshift.io "sample-app" created

deployment.apps "sample-app" created

service "sample-app" created

--> Success

...output omitted...

1. Access the hello and test pods in the network-policy project from the sample-app pod in the different-namespace project.
   1. In the second terminal, view the full name of the sample-app pod with the display-project-info.sh script.

[student@workstation ~]$ **~/DO280/labs/network-policy/display-project-info.sh**

===================================================================

PROJECT: network-policy

POD NAME IP ADDRESS

hello-6c4984d949-g28c4 **10.8.0.13**

test-c4d74c9d5-5pq9s **10.8.0.14**

SERVICE NAME CLUSTER-IP

hello 172.30.137.226

test 172.30.159.119

ROUTE NAME HOSTNAME PORT

hello **hello-network-policy.apps.ocp4.example.com** 8080-tcp

===================================================================

PROJECT: different-namespace

POD NAME

**sample-app-d5f945-spx9q**

===================================================================

* 1. In the first terminal, access the hello pod IP address from the sample-app pod by using the oc rsh and curl commands.

[student@workstation ~]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.13*:8080 | grep Hello**

<h1>Hello, world from nginx!</h1>

* 1. Access the test pod IP address from the sample-app pod by using the oc rsh and curl commands. Target the IP address that was previously retrieved for the test pod.

[student@workstation ~]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.14*:8080 | grep Hello**

<h1>Hello, world from nginx!</h1>

1. In the network-policy project, create a deny-all network policy by using the resource file at ~/DO280/labs/network-policy/deny-all.yaml.
   1. Switch to the network-policy project.

[student@workstation ~]$ **oc project network-policy**

Now using project "network-policy" on server "https://api.ocp4.example.com:6443".

* 1. Change to the ~/DO280/labs/network-policy directory.

[student@workstation ~]$ **cd ~/DO280/labs/network-policy**

* 1. Use a text editor to update the deny-all.yaml file with an empty podSelector field to target all pods in the network-policy project.

kind: NetworkPolicy

apiVersion: networking.k8s.io/v1

metadata:

name: deny-all

spec:

**podSelector: {}**

### NOTE

A solution is provided at ~/DO280/solutions/network-policy/deny-all.yaml.

* 1. Create the network policy with the oc create command.
  2. [student@workstation network-policy]$ **oc create -f deny-all.yaml**

networkpolicy.networking.k8s.io/deny-all created

1. Verify that the deny-all network policy forbids network access to pods in the network-policy project.
   1. Verify that the test pod can no longer access the IP address of the hello pod. Wait a few seconds, and then press **Ctrl**+**C** to exit the curl command that does not reply.

[student@workstation network-policy]$ **oc rsh test-*c4d74c9d5-5pq9s* \**

**curl *10.8.0.13*:8080 | grep Hello**

**^C**

command terminated with exit code 130

* 1. Switch to the different-namespace project.
  2. [student@workstation network-policy]$ **oc project different-namespace**

Now using project "different-namespace" on server "https://api.ocp4.example.com:6443".

* 1. Verify that the sample-app pod can no longer access the IP address of the test pod. Wait a few seconds, and then press **Ctrl**+**C** to exit the curl command that does not reply.

[student@workstation network-policy]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.14*:8080 | grep Hello**

**^C**

command terminated with exit code 130

1. Create a network policy to allow traffic to the hello pod in the network-policy project from the sample-app pod in the different-namespace project via TCP on port 8080. Use the resource file at ~/DO280/labs/network-policy/allow-specific.yaml.
   1. Use a text editor to replace the CHANGE\_ME sections in the allow-specific.yaml file as follows:

...output omitted...

spec:

podSelector:

matchLabels:

deployment: **hello**

ingress:

- from:

- namespaceSelector:

matchLabels:

network: **different-namespace**

podSelector:

matchLabels:

deployment: **sample-app**

ports:

- port: **8080**

protocol: **TCP**

### NOTE

A solution is provided at ~/DO280/solutions/network-policy/allow-specific.yaml.

* 1. Apply the network policy from the allow-specific.yaml file with the oc create command.

[student@workstation network-policy]$ **oc create -n network-policy -f \**

**allow-specific.yaml**

networkpolicy.networking.k8s.io/allow-specific created

* 1. View the network policies in the network-policy project.

[student@workstation network-policy]$ **oc get networkpolicies -n network-policy**

NAME POD-SELECTOR AGE

**allow-specific deployment=hello 11s**

deny-all <none> 5m6s

1. As the admin user, label the different-namespace namespace with the network=different-namespace label.
   1. Log in as the admin user.

[student@workstation network-policy]$ **oc login -u admin -p redhatocp**

Login successful.

...output omitted...

* 1. Apply the network=different-namespace label with the oc label command.

[student@workstation network-policy]$ **oc label namespace different-namespace \**

**network=different-namespace**

namespace/different-namespace labeled

### IMPORTANT

The allow-specific network policy uses labels to match the different-namespace namespace. By default, namespaces and projects do not get any labels automatically.

* 1. Confirm that the different-namespace label was applied.

[student@workstation network-policy]$ **oc describe namespace different-namespace**

Name: different-namespace

Labels: network=different-namespace

...output omitted...

* 1. Log in as the developer user.

[student@workstation network-policy]$ **oc login -u developer -p developer**

Login successful.

...output omitted...

1. Verify that the sample-app pod can access the IP address of the hello pod, but cannot access the IP address of the test pod.
   1. Switch to the different-namespace project.

[student@workstation network-policy]$ **oc project different-namespace**

Already on project "different-namespace" on server "https://api.ocp4.example.com:6443".

* 1. Access the hello pod in the network-policy namespace with the oc rsh and curl commands via the 8080 port.

[student@workstation network-policy]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.13*:8080 | grep Hello**

<h1>Hello, world from nginx!</h1>

* 1. Verify that the hello pod cannot be accessed on another port. Because the network policy allows access only to port 8080 on the hello pod, requests to any other port are ignored and eventually time out. Wait a few seconds, and then press **Ctrl**+**C** to exit the curl command when no response occurs.

[student@workstation network-policy]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.13*:8181 | grep Hello**

**^C**

command terminated with exit code 130

* 1. Verify that the test pod cannot be accessed from the sample-app pod. Wait a few seconds, and then press **Ctrl**+**C** to exit the curl command when no response occurs.

[student@workstation network-policy]$ **oc rsh sample-app-*d5f945-spx9q* \**

**curl *10.8.0.14*:8080 | grep Hello**

**^C**

command terminated with exit code 130

1. Verify if the hello route cannot access the hello pod.
   1. Verify if the hello pod cannot be accessed via its exposed route.

[student@workstation network-policy]$ **curl -s \**

**hello-network-policy.apps.ocp4.example.com**

<h1>Hello, world from nginx!</h1>

The lab environment is a single-node cluster. Because the ingress pods use host networking and the application pods are in the same node, the network policy does not block the traffic.

1. Create a network policy that allows traffic to the hello pod via the exposed route. Use the resource file at ~/DO280/labs/network-policy/allow-from-openshift-ingress.yaml.

This step does not have an effect on the lab environment, because the lab environment is a single-node cluster. On a cluster with multiple nodes, this step is required for correct ingress operation.

* 1. Use a text editor to replace the CHANGE\_ME values in the allow-from-openshift-ingress.yaml file as follows:

...output omitted...

spec:

podSelector: **{}**

ingress:

- from:

- namespaceSelector:

matchLabels:

**policy-group.network.openshift.io/ingress: ""**

### NOTE

A solution is provided at ~/DO280/solutions/network-policy/allow-from-openshift-ingress.yaml.

* 1. Apply the network policy from the allow-from-openshift-ingress.yaml file with the oc create command.

[student@workstation network-policy]$ **oc create -n network-policy -f \**

**allow-from-openshift-ingress.yaml**

networkpolicy.networking.k8s.io/allow-from-openshift-ingress created

* 1. View the network policies in the network-policy namespace.

[student@workstation network-policy]$ **oc get networkpolicies -n network-policy**

NAME POD-SELECTOR AGE

**allow-from-openshift-ingress <none> 10s**

allow-specific deployment=hello 8m16s

deny-all <none> 13m

* 1. Log in as the admin user.

[student@workstation network-policy]$ **oc login -u admin -p redhatocp**

Login successful.

...output omitted...

* 1. Access the hello pod via the exposed route with the curl command.

[student@workstation network-policy]$ **curl -s \**

**hello-network-policy.apps.ocp4.example.com | grep Hello**

<h1>Hello, world from nginx!</h1>

1. Close the terminal window that contains the output of the display-project-info.sh script, and navigate to the home directory.

[student@workstation network-policy]$ **cd**

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish network-policy**

## **Protect Internal Traffic with TLS**

### **Objectives**

* Configure and use automatic service certificates.

### **Zero-trust Environments**

Zero-trust environments assume that every interaction begins in an untrusted state. Users can access only files or objects that are specifically allowed; communication must be encrypted; and client applications must verify the authenticity of servers.

By default, OpenShift encrypts network traffic between nodes and the control plane, and prevents external entities from reading internal traffic. This encryption provides stronger security than default Kubernetes, which does not automatically encrypt internal traffic. Although the control plane traffic is encrypted, applications in OpenShift do not necessarily verify the authenticity of other applications or encrypt application traffic.

Zero-trust environments require that a trusted certificate authority (CA) signs the certificates that are used to encrypt traffic. By referencing the CA certificate, an application can cryptographically verify the authenticity of another application with a signed certificate.

### **Service Certificates**

OpenShift provides the service-ca controller to generate and sign service certificates for internal traffic. The service-ca controller creates a secret that it populates with a signed certificate and key. A deployment can mount this secret as a volume to use the signed certificate. Additionally, client applications need to trust the service-ca controller CA.

#### **Service Certificate Creation**

To generate a certificate and key pair, apply the service.beta.openshift.io/serving-cert-secret-name=*your-secret* annotation to a service. The service-ca controller creates the *your-secret* secret in the same namespace if it does not exist, and populates it with a signed certificate and key pair for the service.

[user@host ~]$ **oc annotate service hello \**

**service.beta.openshift.io/serving-cert-secret-name=hello-secret**

service/hello annotated

|  |  |
| --- | --- |
|  | The hello service is annotated. |
|  | The secret that contains the certificate and key pair is named hello-secret. |

After OpenShift generates the secret, you must mount the secret in the application deployment. The location to place the certificate and key is application-dependent. The following YAML patch is for an NGINX deployment:

spec:

template:

spec:

containers:

- name: hello

volumeMounts:

- name: **hello-volume**

mountPath: **/etc/pki/nginx/**

volumes:

- name: **hello-volume**

secret:

defaultMode: **420**

secretName: **hello-secret**

items:

- key: **tls.crt**

path: **server.crt**

- key: **tls.key**

path: **private/server.key**

|  |  |
| --- | --- |
|  | Defining the volume as hello-volume. |
|  | The application-specific mount path. |
|  | The read-write permissions that the application recommends. |
|  | The secret that the earlier annotation defined. |
|  | The secret has tls.crt as the signed certificate and tls.key as the key. |
|  | The application-specific destinations for the certificate and key. |

After mounting the secret to the application container, the application can use the signed certificate for TLS traffic.

#### **Client Service Application Configuration**

For a client service application to verify the validity of a certificate, the application needs the CA bundle that signed that certificate. The service-ca controller injects the CA bundle when you apply the service.beta.openshift.io/inject-cabundle=true annotation to an object. You can apply the annotation to configuration maps, API services, custom resource definitions (CRD), mutating webhooks, and validating webhooks.

**Configuration Maps**

Apply the service.beta.openshift.io/inject-cabundle=true annotation to a configuration map to inject the CA bundle into the data: { service-ca.crt } field. The service-ca controller replaces all data in the selected configuration map with the CA bundle. You must therefore use a dedicated configuration map to prevent overwriting existing data.

[user@host ~]$ **oc annotate configmap ca-bundle \**

**service.beta.openshift.io/inject-cabundle=true**

configmap/ca-bundle annotated

**API service**

Applying the annotation to an API service injects the CA bundle into the spec.caBundle field.

**CRD**

Applying the annotation to a CRD injects the CA bundle into the spec.conversion.webhook.clientConfig.caBundle field.

**Mutating or validating webhook**

Applying the annotation to a mutating webhook or validating webhook injects the CA bundle into the clientConfig.caBundle field.

#### **Key Rotation**

The service CA certificate is valid for 26 months by default and is automatically rotated after 13 months. After rotation is a 13-month grace period where the original CA certificate is still valid. During this grace period, each pod that is configured to trust the original CA certificate must be restarted in some way. A service restart automatically injects the new CA bundle.

You can also manually rotate the certificate for the service CA and for generated service certificates. To rotate a generated service certificate, delete the existing secret, and the service-ca controller automatically generates a new one.

[user@host ~]$ **oc delete secret *certificate-secret***

secret/certificate-secret deleted

To manually rotate the service CA certificate, delete the signing-key secret in the openshift-service-ca namespace.

[user@host ~]$ **oc delete secret/signing-key -n openshift-service-ca**

secret/signing-key deleted

This process immediately invalidates the former service CA certificate. You must restart all pods that use it, for TLS to function.

### **Alternatives to Service Certificates**

Other options can handle TLS encryption inside an OpenShift cluster, such as a service mesh or the certmanager operator.

You can use the certmanager operator to delegate the certificate signing process to a trusted external service, and also to renew a certificate.

You can also use Red Hat OpenShift Service Mesh for encrypted service-to-service communication and for other advanced features. Service mesh is an advanced topic and is not covered in the course.

### **Patching Kubernetes Resources**

You can modify objects in OpenShift in a repeatable way with the oc patch command. The oc patch command updates or adds fields in an existing object from a provided JSON or YAML snippet or file. A software developer might distribute a patch file or snippet to fix problems before a full update is available.

To patch an object from a snippet, use the oc patch command with the -p option and the snippet. The following example updates the hello deployment to have a CPU resource request of 100m with a JSON snippet:

[user@host ~]$ **oc patch deployment hello -p \**

**'{"spec":{"template":{"spec":{"resources":{"requests":{"cpu": "100m"}}}}}}'**

deployment/hello patched

To patch an object from a patch file, use the oc patch command with the --patch-file option and the location of the patch file. The following example updates the hello deployment to include the content of the ~/volume-mount.yaml patch file:

[user@host ~]$ **oc patch deployment hello --patch-file ~/volume-mount.yaml**

deployment.apps/hello patched

The contents of the patch file describe mounting a persistent volume claim as a volume:

spec:

template:

spec:

containers:

- name: hello

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html/

volumes:

- name: www

persistentVolumeClaim:

claimName: nginx-www

This patch results in the following manifest for the hello deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

...output omitted...

spec:

...output omitted...

template:

...output omitted...

spec:

containers:

...output omitted...

name: server

...output omitted...

volumeMounts:

- **mountPath: /usr/share/nginx/html/**

**name: www**

- mountPath: /etc/nginx/conf.d/

name: tls-conf

...output omitted...

volumes:

- configMap:

defaultMode: 420

name: tls-conf

name: tls-conf

- **persistentVolumeClaim:**

**claimName: nginx-www**

**name: www**

...output omitted...

The patch applies to the hello deployment regardless of whether the www volume mount exists. The oc patch command modifies existing fields in the object that are specified in the patch. If the beginning state of the hello deployment already contains data as follows, then the end result is the same as if the fields do not exist:

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

...output omitted...

spec:

...output omitted...

template:

...output omitted...

spec:

containers:

...output omitted...

name: server

...output omitted...

volumeMounts:

- mountPath: ***/usr/share/nginx/www/***

name: www

- mountPath: /etc/nginx/conf.d/

name: tls-conf

...output omitted...

volumes:

- configMap:

defaultMode: 420

name: tls-conf

name: tls-conf

- persistentVolumeClaim:

claimName: ***deprecated-www***

name: www

...output omitted...

|  |  |
| --- | --- |
|  | The www volume already exists. The patch replaces the existing data with the new data. |

### REFERENCES

For more information about service certificates, refer to the Securing Service Traffic Using Service Serving Certificate Secrets section in the Configuring Certificates chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance#add-service-serving>

For more information about service mesh, refer to the About OpenShift Service Mesh section in the Service Mesh 2.x chapter in the Red Hat OpenShift Container Platform 4.14 Service Mesh documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/service_mesh#ossm-about>

For more information about the cert-manager operator, refer to the cert-manager Operator for Red Hat OpenShift chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance#cert-manager-operator-for-red-hat-openshift>

For more information about the oc patch command, refer to the oc patch section in the OpenShift CLI Developer Command Reference chapter in the Red Hat OpenShift Container Platform 4.14 CLI Tools documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#oc-patch>

[Red Hat Topics - What Is Zero Trust?](https://www.redhat.com/en/topics/security/what-is-zero-trust)

## **Guided Exercise: Protect Internal Traffic with TLS**

Configure two applications to connect securely inside the cluster by using TLS certificates that OpenShift manages.

**Outcomes**

* Generate service certificates with the service-ca controller.
* Mount a service certificate by using secrets.
* Use a configuration map to inject a service certificate into a pod.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the OpenShift cluster is ready and creates the network-svccerts project and server deployment for the exercise. The command also creates a test pod named no-ca-bundle for use later in the exercise.

[student@workstation ~]$ **lab start network-svccerts**

**Instructions**

In this exercise, you work with the server deployment, which has an NGINX container that serves a "Hello World!" page with the HTTPS protocol. This deployment differs from earlier NGINX deployments, because it allows only the HTTPS protocol. The server application expects the existence of a certificate that you create in the exercise steps.

1. Log in to the OpenShift cluster as the admin user and switch to the network-svccerts project.
   1. Use the oc login command to log in to api.ocp4.example.com:6443 as the admin user with the redhatocp password.

[student@workstation ~]$ **oc login -u admin -p redhatocp \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

* 1. Use the oc project command to switch to the network-svccerts project.

[student@workstation ~]$ **oc project network-svccerts**

Now using project "network-svccerts" on server "https://api.ocp4.example.com:6443".

1. Generate a service certificate and secret that are named server-secret for the server service, and then mount the secret in the server deployment.
   1. Annotate the server service with service.beta.openshift.io/serving-cert-secret-name=server-secret by using the oc annotate command. It automatically creates a secret named server-secret, which is populated with a signed TLS key and certificate.

[student@workstation ~]$ **oc annotate service server \**

**service.beta.openshift.io/serving-cert-secret-name=server-secret**

service/server annotated

* 1. Use the oc describe command to view the service and secret descriptions to verify that OpenShift created the secret.

[student@workstation ~]$ **oc describe service server**

...output omitted...

Annotations: service.beta.openshift.io/serving-cert-secret-name: server-secret

service.beta.openshift.io/serving-cert-signed-by: openshift-service-serving-signer@1667565598

...output omitted...

[student@workstation ~]$ **oc describe secret server-secret**

Name: server-secret

Namespace: network-svccerts

...output omitted...

Type: kubernetes.io/tls

Data

====

tls.key: 1675 bytes

tls.crt: 2615 bytes

* 1. Use a text editor to create a patch file to mount the server-secret secret in the server deployment. Edit the resource file at ~/DO280/labs/network-svccerts/server-secret.yaml. Replace the CHANGE\_ME sections as shown in the following example:

spec:

template:

spec:

containers:

- name: server

volumeMounts:

- name: **server-secret**

mountPath: /etc/pki/nginx/

volumes:

- name: **server-secret**

secret:

defaultMode: 420

secretName: **server-secret**

items:

- key: **tls.crt**

path: server.crt

- key: **tls.key**

path: private/server.key

* 1. Apply the patch file to the server deployment with the oc patch command.

[student@workstation ~]$ **oc patch deployment server \**

**--patch-file ~/DO280/labs/network-svccerts/server-secret.yaml**

deployment.apps/server patched

* 1. Use the openssl s\_client command in the no-ca-bundle pod to verify that OpenShift supplied the server deployment with a certificate. Verify that the no-ca-bundle pod needs to configure the CA that issued the OpenShift service certificate for certificate validation.

[student@workstation ~]$ **oc exec no-ca-bundle -- \**

**openssl s\_client -connect server.network-svccerts.svc:443**

depth=1 CN = openshift-service-serving-signer@1667565598

CONNECTED(00000004)

---

Certificate chain

0 s:CN = server.network-svccerts.svc

i:CN = openshift-service-serving-signer@1667565598

1 s:CN = openshift-service-serving-signer@1667565598

i:CN = openshift-service-serving-signer@1667565598

---

...output omitted...

**verify error:num=19:self signed certificate in certificate chain**

DONE

### NOTE

The output shows the verify error:num=19:self signed certificate in certificate chain error, because the no-ca-bundle pod is not configured with the OpenShift cluster's CA bundle.

1. Generate the ca-bundle configuration map that contains the service CA bundle, and use it to create the client pod.
   1. Create an empty configuration map named ca-bundle by using the oc create command.

[student@workstation ~]$ **oc create configmap ca-bundle**

configmap/ca-bundle created

* 1. Annotate the ca-bundle configuration map with service.beta.openshift.io/inject-cabundle=true by using the oc annotate command.

[student@workstation ~]$ **oc annotate configmap ca-bundle \**

**service.beta.openshift.io/inject-cabundle=true**

configmap/ca-bundle annotated

* 1. View the YAML output of the ca-bundle configuration map to verify that the CA bundle is present.

[student@workstation ~]$ **oc get configmap ca-bundle -o yaml**

...output omitted...

data:

service-ca.crt: |

-----BEGIN CERTIFICATE-----

...output omitted...

* 1. Use a text editor to add the ca-bundle configuration map to the client.yaml pod definition. Edit the resource file at ~/DO280/labs/network-svccerts/client.yaml. Replace the CHANGE\_ME sections of the file as shown in the following example:

apiVersion: apps/v1

kind: Deployment

metadata:

annotations:

labels:

app: client

name: client

namespace: network-svccerts

spec:

replicas: 1

selector:

matchLabels:

deployment: client

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

annotations:

openshift.io/generated-by: OpenShiftNewApp

labels:

deployment: client

spec:

containers:

- image: registry.ocp4.example.com:8443/redhattraining/hello-world-nginx

imagePullPolicy: IfNotPresent

name: client-deploy

ports:

- containerPort: 8080

protocol: TCP

volumeMounts:

- mountPath: /etc/pki/ca-trust/extracted/pem

name: **trusted-ca**

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

dnsPolicy: ClusterFirst

restartPolicy: Always

schedulerName: default-scheduler

terminationGracePeriodSeconds: 30

volumes:

- configMap:

defaultMode: 420

name: **ca-bundle**

items:

- key: **service-ca.crt**

path: tls-ca-bundle.pem

name: **trusted-ca**

name: **trusted-ca**

* 1. Apply the client.yaml file with the oc apply command to create the client pod.

[student@workstation ~]$ **oc apply -f ~/DO280/labs/network-svccerts/client.yaml**

...output omitted...

pod/client created

1. Show that the server service is now accessible over HTTPS with a certificate that is signed by the OpenShift cluster.
   1. Use the curl command within the client pod to test that the server service is accessible on HTTPS.

[student@workstation ~]$ **oc exec deploy/client -- \**

**curl -s https://server.network-svccerts.svc**

<html>

<body>

<h1>Hello, world from nginx!</h1>

</body>

</html>

* 1. Use the openssl s\_client command within the client pod to verify that the certificate is signed by the OpenShift cluster.

[student@workstation ~]$ **oc exec deploy/client -- \**

**openssl s\_client -connect server.network-svccerts.svc:443**

CONNECTED(00000004)

---

Certificate chain

0 s:CN = server.network-svccerts.svc

i:CN = openshift-service-serving-signer@1667565598

1 s:CN = openshift-service-serving-signer@1667565598

i:CN = openshift-service-serving-signer@1667565598

---

...output omitted...

**verify return:1**

DONE

[student@workstation ~]$ **lab finish network-svccerts**

## **Lab: Network Security**

Configure firewall rules to protect microservice communication, and also configure TLS encryption between those microservices and for external access.

**Outcomes**

* Encrypt internal traffic between pods by using TLS service secrets that OpenShift generates.
* Route external traffic to terminate TLS within the cluster.
* Restrict ingress traffic for a group of pods by using network policies.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start network-review**

This command ensures that the environment is ready and copies the necessary files for this exercise.

This command also deploys an API that is composed of a product and a stock microservice to the network-review project.

The product microservice is the entry point to the API. The stock microservice provides only additional information to the product response. If the product microservice cannot reach the stock microservice, then the product microservice returns the -1 value.

The developer deployed the API without the security configuration. You must configure TLS for end-to-end communications and restrict the ingress to pods for both microservices.

To complete the exercise, the following URLs must respond without errors:

* https://stock.network-review.svc.cluster.local/product/1
* https://product.apps.ocp4.example.com/products

### NOTE

The lab start deploys solution files in the ~/DO280/solutions/network-review/ directory.

**Instructions**

1. Log in to your OpenShift cluster as the admin user with the redhatocp password.
   1. Use the oc login command to log in to your OpenShift cluster as the admin user.

[student@workstation ~]$ **oc login -u admin -p redhatocp \**

**https://api.ocp4.example.com:6443**

Login successful.

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the stock-service-cert secret for the OpenShift service certificate to encrypt communications between the product and the stock microservices.
   1. Change to the network-review project.

[student@workstation ~]$ **oc project network-review**

Now using project "network-review" on server "https://api.ocp4.example.com:6443"

* 1. Change to the ~/DO280/labs/network-review directory to access the lab files.

[student@workstation ~]$ **cd ~/DO280/labs/network-review**

* 1. Edit the stock-service.yaml manifest to configure the stock service with the service.beta.openshift.io/serving-cert-secret-name: stock-service-cert annotation. This annotation creates the stock-service-cert secret with the service certificate and the key.

apiVersion: v1

kind: Service

metadata:

name: stock

namespace: network-review

**annotations:**

**service.beta.openshift.io/serving-cert-secret-name: stock-service-cert**

spec:

...output omitted...

* 1. Apply the stock service changes by using the oc apply command.

[student@workstation network-review]$ **oc apply -f stock-service.yaml**

service/stock configured

* 1. Verify that the stock-service-cert secret contains a valid certificate for the stock.network-review.svc hostname in the tls.crt secret key. Decode the secret output with the base64 command by using the -d option. Then, use the openssl x509 command to read the output from standard input, and use the -text option to print the certificate in text form.

[student@workstation network-review]$ **oc get secret stock-service-cert \**

**--output="jsonpath={.data.tls\.crt}" \**

**| base64 -d \**

**| openssl x509 -text**

...output omitted...

X509v3 Subject Alternative Name:

**DNS:stock.network-review.svc, DNS:stock.network-review.svc.cluster.local**

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Configure TLS on the stock microservice by using the stock-service-cert secret that OpenShift generates.

Use the following settings in the deployment to configure TLS:

* 1. Set the path for the certificate and key to /etc/pki/stock/.
  2. Set the TLS\_ENABLED environment variable to "true".
  3. Update the liveness and readiness probes to use TLS.
  4. Change the service to listen on the standard HTTPS 443 port.
  5. Edit the stock-deployment.yaml file to mount the stock-service-cert secret on the /etc/pki/stock/ path.

apiVersion: apps/v1

kind: Deployment

metadata:

name: stock

namespace: network-review

spec:

...output omitted...

spec:

containers:

- name: stock

...output omitted...

env:

- name: TLS\_ENABLED

value: "false"

**volumeMounts:**

**- name: stock-service-cert**

**mountPath: /etc/pki/stock/**

**volumes:**

**- name: stock-service-cert**

**secret:**

**defaultMode: 420**

**secretName: stock-service-cert**

* 1. Edit the stock deployment in the stock-deployment.yaml file to configure TLS for the application and for the liveness and readiness probes.

apiVersion: apps/v1

kind: Deployment

metadata:

name: stock

namespace: network-review

spec:

...output omitted...

spec:

containers:

- name: stock

...output omitted...

ports:

- containerPort: 8085

readinessProbe:

httpGet:

port: 8085

path: /readyz

**scheme: HTTPS**

livenessProbe:

httpGet:

port: 8085

path: /livez

**scheme: HTTPS**

env:

- name: TLS\_ENABLED

**value: "true"**

...output omitted...

* 1. Apply the stock deployment updates by using the oc apply command.

[student@workstation network-review]$ **oc apply -f stock-deployment.yaml**

deployment/stock configured

* 1. Edit the stock-service.yaml file to configure the stock service to listen on the standard HTTPS 443 port.

apiVersion: v1

kind: Service

metadata:

name: stock

namespace: network-review

annotations:

service.beta.openshift.io/serving-cert-secret-name: stock-service-cert

spec:

selector:

app: stock

ports:

- **port: 443**

targetPort: 8085

**name: https**

* 1. Apply the stock service changes by using the oc apply command.

[student@workstation network-review]$ **oc apply -f stock-service.yaml**

service/stock configured

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Configure TLS between the product and the stock microservices by using the internal Certificate Authority (CA) from OpenShift.

The product microservice requires the following settings:

* 1. The CERT\_CA environment variable that is set to /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem to access the OpenShift CA
  2. The STOCK\_URL environment variable with the HTTPS protocol
  3. Edit the configuration map in the service-ca-configmap.yaml file to add the service.beta.openshift.io/inject-cabundle: "true" annotation. This annotation injects the OpenShift internal CA into the service-ca configuration map.

apiVersion: v1

kind: ConfigMap

metadata:

name: service-ca

namespace: network-review

**annotations:**

**service.beta.openshift.io/inject-cabundle: "true"**

data: {}

* 1. Create the service-ca configuration map by using the oc create command.

[student@workstation network-review]$ **oc create -f service-ca-configmap.yaml**

configmap/service-ca created

* 1. Verify that OpenShift injects the CA certificate by describing the service-ca configuration map with the oc describe command.

[student@workstation network-review]$ **oc describe configmap service-ca**

Name: service-ca

Namespace: network-review

Labels: <none>

Annotations: service.beta.openshift.io/inject-cabundle: true

Data

====

**service-ca.crt:**

-----

-----BEGIN CERTIFICATE-----

* 1. Edit the product-deployment.yaml file to configure the product deployment to use the service-ca configuration map, to add the CERT\_CA environment variable, and to update the STOCK\_URL environment variable to use the HTTPS protocol.

apiVersion: apps/v1

kind: Deployment

metadata:

name: product

namespace: network-review

spec:

...output omitted...

spec:

containers:

- name: product

...output omitted...

env:

- **name: CERT\_CA**

**value: "/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem"**

- name: TLS\_ENABLED

value: "false"

- name: STOCK\_URL

**value: "https://stock.network-review.svc"**

**volumeMounts:**

**- name: trusted-ca**

**mountPath: /etc/pki/ca-trust/extracted/pem**

**volumes:**

**- name: trusted-ca**

**configMap:**

**defaultMode: 420**

**name: service-ca**

**items:**

**- key: service-ca.crt**

**path: tls-ca-bundle.pem**

* 1. Apply the product deployment updates by using the oc apply command.

[student@workstation network-review]$ **oc apply -f product-deployment.yaml**

deployment/product configured

* 1. Send a request to the https://stock.network-review.svc/product/1 URL from product deployment to verify that you can query the stock microservice by using HTTPS. Run the oc exec command to run the curl command to send a request to the stock microservice.

[student@workstation network-review]$ **oc exec deployment/product \**

**-- curl -s https://stock.network-review.svc/product/1**

10

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Configure TLS on the product microservice by using a signed certificate by a corporate CA to accept TLS connections from outside the cluster.

You have the CA certificate and the signed certificate for the product.apps.ocp4.example.com domain in the certs directory of the lab.

Use the following settings in the deployment to configure TLS:

* 1. Set the path for the certificate and key to /etc/pki/product/.
  2. Set the TLS\_ENABLED environment variable to the "true" value.
  3. Update the liveness and readiness probes to use TLS.
  4. Create the passthrough-cert secret by using the product.pem certificate and the product.key key from the lab directory.

[student@workstation network-review]$ **oc create secret tls passthrough-cert \**

**--cert certs/product.pem --key certs/product.key**

secret/passthrough-cert created

* 1. Edit the product deployment to mount the passthrough-cert secret on the /etc/pki/product/ path.

apiVersion: apps/v1

kind: Deployment

metadata:

name: product

spec:

...output omitted...

spec:

containers:

- name: product

...output omitted...

volumeMounts:

**- name: passthrough-cert**

**mountPath: /etc/pki/product/**

- name: trusted-ca

mountPath: /etc/pki/ca-trust/extracted/pem

volumes:

**- name: passthrough-cert**

**secret:**

**defaultMode: 420**

**secretName: passthrough-cert**

- name: trusted-ca

configMap:

defaultMode: 420

name: service-ca

items:

- key: service-ca.crt

path: tls-ca-bundle.pem

* 1. Edit the product deployment to configure TLS for the application and for the liveness and readiness probes.

apiVersion: apps/v1

kind: Deployment

metadata:

name: product

spec:

...output omitted...

spec:

containers:

- name: product

...output omitted...

ports:

- containerPort: 8080

readinessProbe:

httpGet:

port: 8080

path: /readyz

**scheme: HTTPS**

livenessProbe:

httpGet:

port: 8080

path: /livez

**scheme: HTTPS**

env:

- name: TLS\_ENABLED

**value: "true"**

- name: STOCK\_URL

value: "https://stock.network-review.svc"

...output omitted...

* 1. Apply the product deployment updates by using the oc apply command.

[student@workstation network-review]$ **oc apply -f product-deployment.yaml**

deployment.apps/product configured

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Expose the product microservice to outer cluster access by using the FQDN in the signed certificate by the corporate CA. Use the product.apps.ocp4.example.com hostname.

Create a passthrough route for the product service by using the product.apps.ocp4.example.com hostname.

[student@workstation network-review]$ **oc create route passthrough product-https \**

**--service product --port 8080 \**

**--hostname product.apps.ocp4.example.com**

route.route.openshift.io/product-https created

* 1. Verify that you can query the product microservice from outside the cluster by using the curl command with the ca.pem CA certificate.

[student@workstation network-review]$ **curl --cacert certs/ca.pem \**

**https://product.apps.ocp4.example.com/products**

[{"id":1,"name":"rpi4\_4gb","stock":10},{"id":2,"name":"rpi4\_8gb","stock":5}]

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Configure network policies to accept only ingress connections to the stock pod on the 8085 port that come from a pod with the app=product label.
   1. Edit the stock-ingresspolicy.yaml to add the network policy specification.

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: ingress-stock-policy

spec:

**podSelector:**

**matchLabels:**

**app: stock**

**ingress:**

**- from:**

**- podSelector:**

**matchLabels:**

**app: product**

**ports:**

**- protocol: TCP**

**port: 8085**

* 1. Create the network policy.

[student@workstation network-review]$ **oc create -f stock-ingresspolicy.yaml**

networkpolicy.networking.k8s.io/stock-ingress-policy created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Configure network policies to accept only ingress connections to the product pod on the 8080 port that come from the OpenShift router pods.
   1. Edit the product-ingresspolicy.yaml file to accept ingress connections from router pods by adding a namespace selector with the policy-group.network.openshift.io/ingress label.

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: product-ingress-policy

spec:

**podSelector:**

**matchLabels:**

**app: product**

**ingress:**

**- from:**

**- namespaceSelector:**

**matchLabels:**

**policy-group.network.openshift.io/ingress: ""**

**ports:**

**- protocol: TCP**

**port: 8080**

* 1. Create the network policy.

[student@workstation network-review]$ **oc create -f product-ingresspolicy.yaml**

networkpolicy.networking.k8s.io/product-ingress-policy created

* 1. Change to the home directory.

[student@workstation network-ingress]$ **cd**

1. [Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade network-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish network-review**

# Chapter 5.  Expose non-HTTP/SNI Applications

[Load Balancer Services](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05)

[Guided Exercise: Load Balancer Services](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05s02)

[Multus Secondary Networks](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05s03)

[Guided Exercise: Multus Secondary Networks](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05s04)

[Lab: Expose non-HTTP/SNI Applications](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05s05)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch05s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Expose applications to external access without using an ingress controller. |
| **Objectives** | * Expose applications to external access by using load balancer services. * Expose applications to external access by using a secondary network. |
| **Sections** | * Load Balancer Services (and Guided Exercise) * Multus Secondary Networks (and Guided Exercise) |
| **Lab** | * Expose non-HTTP/SNI Applications |

## **Load Balancer Services**

### **Objectives**

* Expose applications to external access by using load balancer services.

### **Exposing Non-HTTP Services**

When you use Kubernetes, you run workloads that provide services to users. You create resources such as deployments to run workloads, for example a web application. Ingresses and routes provide a way to expose the services that these workloads implement. However, in some scenarios, ingresses and routes are not sufficient to expose the service that a pod provides.

Many internet services implement a process that listens on a given port and IP address. For example, a service that uses the 1.2.3.4 IP address runs an SSH server that listens on port 22. Clients connect to port 22 on that IP address to use the SSH service.

Web servers implement the HTTP protocol and other related protocols such as HTTPS.

Kubernetes ingresses and OpenShift routes use the virtual hosting property of the HTTP protocol to expose web services that are running on the cluster. Ingresses and routes run a single web server that uses virtual hosting to route each incoming request to a Kubernetes service by using the request hostname.

For example, ingresses can route requests for the https://a.example.com URL to a Kubernetes service in the cluster, and can route requests for the https://b.example.com URL to a different service in the cluster.

However, many protocols do not have equivalent features. Ingress and route resources can expose only HTTP services. To expose non-HTTP services, you must use a different resource. Because these resources cannot expose multiple services on the same IP address and port, they require more setup effort, and might require more resources, such as IP addresses.

### IMPORTANT

Preferably use ingresses and routes to expose services when possible.

### **Kubernetes Services**

Kubernetes workloads are flexible resources that can create many pods. By creating multiple pods for a workload, Kubernetes can provide increased reliability and performance. If a pod fails, then other pods can continue providing a service. With multiple pods, which possibly run on different systems, workloads can use more resources for increased performance.

However, if many pods provide a workload service, then users of the service can no longer access the service by using the combination of a single IP address and a port. To provide transparent access to workload services that run on multiple pods, Kubernetes uses resources of the Service type. A service resource contains the following information:

* A selector that describes the pods that run the service
* A list of the ports that provide the service on the pods

Different types of Kubernetes services exist, each with different purposes:

**Internal communication**

Services of the ClusterIP type provide service access within the cluster.

**Exposing services externally**

Services of the NodePort and LoadBalancer types, as well as the use of the external IP feature of ClusterIP services, expose services that are running in the cluster to outside the cluster.

Different providers can implement Kubernetes services, by using the type field of the service resource.

Although these services are useful in specific scenarios, some services require extra configuration, and they can pose security challenges. Load balancer services have fewer limitations and provide load balancing.

### **Load Balancer Services**

Load balancer services require the use of network features that are not available in all environments.

For example, cloud providers typically provide their own load balancer services. These services use features that are specific to the cloud provider.

If you run a Kubernetes cluster on a cloud provider, controllers in Kubernetes use the cloud provider's APIs to configure the required cloud provider resources for a load balancing service. On environments where managed load balancer services are not available, you must configure a load balancer component according to the specifics of your network.

### **The MetalLB Component**

MetalLB is a load balancer component that provides a load balancing service for clusters that do not run on a cloud provider, such as a bare metal cluster, or clusters that run on hypervisors. MetalLB operates in two modes: layer 2 and Border Gateway Protocol (BGP), with different properties and requirements. You must plan the use of MetalLB to consider your requirements and your network design.

MetalLB is an operator that you can install with the Operator Lifecycle Manager. After installing the operator, you must configure MetalLB through its custom resource definitions. In most situations, you must provide MetalLB with an IP address range.

### **Using Load Balancer Services**

When a cluster has a configured load balancer component, you can create services of the LoadBalancer type to expose non-HTTP services outside the cluster.

For example, the following resource definition exposes port 1234 on pods with the example value for the name label.

apiVersion: v1

kind: Service

metadata:

name: example-lb

namespace: example

spec:

ports:

- port: 1234

protocol: TCP

targetPort: 1234

selector:

name: example

type: LoadBalancer

|  |  |
| --- | --- |
|  | Exposed port |
|  | Pod selector |
|  | LoadBalancer service type You can also use the kubectl expose command with the --type LoadBalancer argument to create load balancer services imperatively. |

After you create the service, the load balancer component updates the service resource with information such as the public IP address where the service is available.

[user@host ~]$ **kubectl get service**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

example-lb LoadBalancer 172.30.21.79 **192.168.50.20** **1234**:31265/TCP 4m7s

You can now connect to the service on port 1234 of the 192.168.50.20 address.

You can also obtain the address from the status field of the resource.

[user@host ~]$ **oc get example-lb -o jsonpath="{.status.loadBalancer.ingress}"**

[{"ip":"192.168.50.20"}]

Each load balancer service allocates IP addresses for services by following different processes. For example, when installing MetalLB, you must provide ranges of IPs that MetalLB assigns to services.

After exposing a service by using a load balancer, always verify that the service is available from your intended network locations. Use a client for the exposed protocol to ensure connectivity, and test that load balancing works as expected. Some protocols might require further adjustments to work correctly behind a load balancer. You can also use network debugging tools, such as the ping and traceroute commands to examine connectivity.

### REFERENCES

For more information, refer to the Load Balancing with MetalLB chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#load-balancing-with-metallb>

[Kubernetes Services](https://kubernetes.io/docs/concepts/services-networking/service/)

[MetalLB on OpenShift](https://metallb.universe.tf/installation/clouds/#metallb-on-openshift-ocp)

## **Guided Exercise: Load Balancer Services**

Expose a deployment to external access by using a load balancer service.

**Outcomes**

* Use load balancer services to expose the video streams that the application produces.
* Access the video streams with a media player.
* Realize that external factors can cause a load balancer to fail.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start non-http-lb**

**Instructions**

1. Log in as the developer user, and list the YAML resource manifests for the video streaming application in the ~/DO280/labs/non-http-lb directory.
   1. Log in to the cluster as the developer user.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Create the non-http-lb project.
  2. [student@workstation ~]$ **oc new-project non-http-lb**
  3. Now using project "non-http-lb" on server ...

...output omitted...

* 1. Change to the ~/DO280/labs/non-http-lb directory.
  2. [student@workstation ~]$ **cd ~/DO280/labs/non-http-lb**

[student@workstation non-http-lb]$

* 1. List the contents of the directory. The YAML resource manifests represent three instances of the video streaming application.
  2. [student@workstation non-http-lb]$ **ls -l**
  3. total 12
  4. -rw-rw-r--. 1 student student 1561 Jun 21 16:29 virtual-rtsp-1.yaml
  5. -rw-rw-r--. 1 student student 1563 Jun 21 16:29 virtual-rtsp-2.yaml

-rw-rw-r--. 1 student student 1565 Jun 21 16:21 virtual-rtsp-3.yaml

* 1. Each deployment emulates the video stream from a security camera on port 8554.

| **Deployment** | **Video stream** | **Location** | **Image** |
| --- | --- | --- | --- |
| virtual-rtsp-1 | Camera 1 | Downtown | |  | | --- | |  | |
| virtual-rtsp-2 | Camera 2 | Roundabout | |  | | --- | |  | |
| virtual-rtsp-3 | Camera 3 | Intersection | |  | | --- | |  | |

1. Deploy the first instance of the application, and expose the video stream from the **downtown** camera by using a load balancer service.
   1. Create the first instance of the video stream deployment. This application produces the video stream of the **downtown** camera.
   2. [student@workstation non-http-lb]$ **oc apply -f virtual-rtsp-1.yaml**

deployment.apps/virtual-rtsp-1 created

* 1. Wait until the pod is running and the deployment is ready. Press **Ctrl**+**C** to exit the watch command.
  2. [student@workstation non-http-lb]$ **watch oc get deployments,pods**
  3. Every 2.0s: oc get deployments,pods workstation: Wed Jun 21 16:25:26 2023
  4. NAME **READY** UP-TO-DATE AVAILABLE AGE
  5. deployment.apps/virtual-rtsp-1 **1/1** 1 1 59s
  6. NAME READY **STATUS** RESTARTS AGE

pod/virtual-rtsp-1-98cd84d79a-qhn9r 1/1 **Running** 0 59s

* 1. Create a load balancer service to expose the first deployment.
  2. [student@workstation non-http-lb]$ **oc expose deployment/virtual-rtsp-1 \**
  3. **--type=LoadBalancer --target-port=8554**

service/virtual-rtsp-1 exposed

* 1. Get the external IP address of the load balancer service.
  2. [student@workstation non-http-lb]$ **oc get services**
  3. NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

virtual-rtsp-1 LoadBalancer 172.30.4.18 **192.168.50.20** 8554:32170/TCP 59s

* 1. Verify that you can connect to the external IP address of the load balancer service on port 8554.
  2. [student@workstation non-http-lb]$ **nc -vz 192.168.50.20 8554**
  3. Ncat: Version 7.91 ( https://nmap.org/ncat )
  4. Ncat: Connected to 192.168.50.20:8554.

Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds

* 1. Open the URL in the media player to confirm that the video stream of the **downtown** camera is working correctly.
     + rtsp://192.168.50.20:8554/stream
     + [student@workstation non-http-lb]$ **totem rtsp://192.168.50.20:8554/stream**

...output omitted...

Close the media player window after confirming that the video stream works correctly.

|  |
| --- |
|  |

1. Deploy the remaining instances of the video stream application. Expose the video streams from the **roundabout** and **intersection** cameras by using a load balancer service. Understand that the classroom is configured to provide only two IP addresses.
   1. Create the second instance of the video stream deployment. This application produces the video stream of the **roundabout** camera.
   2. [student@workstation non-http-lb]$ **oc apply -f virtual-rtsp-2.yaml**

deployment.apps/virtual-rtsp-2 created

* 1. Create the third instance of the video stream deployment. This application produces the video stream of the **intersection** camera.
  2. [student@workstation non-http-lb]$ **oc apply -f virtual-rtsp-3.yaml**

deployment.apps/virtual-rtsp-3 created

* 1. Wait until the pods are running and the deployments are ready. Press **Ctrl**+**C** to exit the watch command.
  2. [student@workstation non-http-lb]$ **watch oc get deployments,pods**
  3. Every 2.0s: oc get deployments,pods workstation: Wed Jun 21 16:30:33 2023
  4. NAME READY UP-TO-DATE AVAILABLE AGE
  5. deployment.apps/virtual-rtsp-1 1/1 1 1 5m
  6. deployment.apps/virtual-rtsp-2 **1/1** 1 1 60s
  7. deployment.apps/virtual-rtsp-3 **1/1** 1 1 30s
  8. NAME READY STATUS RESTARTS AGE
  9. pod/virtual-rtsp-1-98cd84d79a-qhn9r 1/1 Running 0 5m
  10. pod/virtual-rtsp-2-769b5bcb89-r8csp 1/1 **Running** 0 60s

pod/virtual-rtsp-3-6cdb9f7ffb-g6d9d 1/1 **Running** 0 30s

* 1. Create a load balancer service to expose the second deployment.
  2. [student@workstation non-http-lb]$ **oc expose deployment/virtual-rtsp-2 \**
  3. **--type=LoadBalancer --target-port=8554**

service/virtual-rtsp-2 exposed

* 1. Create a load balancer service to expose the third deployment.
  2. [student@workstation non-http-lb]$ **oc expose deployment/virtual-rtsp-3 \**
  3. **--type=LoadBalancer --target-port=8554**

service/virtual-rtsp-3 exposed

* 1. Get the external IP address of the second load balancer service.
  2. [student@workstation non-http-lb]$ **oc get services**
  3. NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) ...
  4. virtual-rtsp-1 LoadBalancer 172.30.94.188 192.168.50.20 8554:32325/TCP ...
  5. virtual-rtsp-2 LoadBalancer 172.30.15.148 **192.168.50.21** 8554:31640/TCP

virtual-rtsp-3 LoadBalancer 172.30.228.35 **<pending>** 8554:32089/TCP

|  |  |
| --- | --- |
|  | The second load balancer service has an associated external IP address. |
|  | No IP address is assigned to the third load balancer, and it is displayed as <pending> because all available load balancer IP addresses are in use. The MetalLB operator in the classroom uses the IPAddressPools configuration to restrict the available load balancer IP addresses to 192.168.50.20 and 192.168.50.21. |

* 1. Open the URL in the media player to confirm that the video stream of the **roundabout** camera is working correctly.
     + rtsp://192.168.50.21:8554/stream
     + [student@workstation non-http-lb]$ **totem rtsp://192.168.50.21:8554/stream**

...output omitted...

Close the media player window after confirming that the video stream works correctly.

|  |
| --- |
|  |

1. Delete the first service to reallocate the IP address to the third service, and view the video stream of the **intersection** camera.
   1. Delete the first service to release the assigned IP address.
   2. [student@workstation non-http-lb]$ **oc delete service/virtual-rtsp-1**

service "virtual-rtsp-1" deleted

* 1. Verify that the third service has an assigned external IP address.
  2. [student@workstation non-http-lb]$ **oc get services**
  3. NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) ...
  4. virtual-rtsp-2 LoadBalancer 172.30.15.148 192.168.50.21 8554:31640/TCP ...

**virtual-rtsp-3** LoadBalancer 172.30.228.35 **192.168.50.20** 8554:32089/TCP

|  |  |
| --- | --- |
|  | The IP address is now allocated to the third service. |

* 1. Open the URL in the media player to confirm that the video stream of the **intersection** camera is working correctly.
     + rtsp://192.168.50.20:8554/stream
     + [student@workstation non-http-lb]$ **totem rtsp://192.168.50.20:8554/stream**

...output omitted...

Close the media player window after confirming that the video stream works correctly.

|  |
| --- |
|  |

1. Clean up the resources.
   1. Change to the student HOME directory.
   2. [student@workstation non-http-lb]$ **cd**

[student@workstation ~]$

* 1. Delete all the services in the namespace.
  2. [student@workstation ~]$ **oc delete services --all**
  3. service "virtual-rstp-2" deleted

service "virtual-rstp-3" deleted

* 1. Delete all the deployments in the namespace.
  2. [student@workstation ~]$ **oc delete deployments --all**
  3. deployment.apps "virtual-rtsp-1" deleted
  4. deployment.apps "virtual-rtsp-2" deleted

deployment.apps "virtual-rtsp-3" deleted

* 1. Delete the non-http-lb project.
  2. [student@workstation ~]$ **oc delete project/non-http-lb**

project.project.openshift.io "non-http-lb" deleted

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish non-http-lb**

## **Lab: Expose non-HTTP/SNI Applications**

Expose applications to external access without using an ingress controller.

**Outcomes**

* Expose a non-http application to external access by using the LoadBalancer type service.
* Configure a network attachment definition for an isolated network.
* Make an application accessible outside the cluster on an isolated network by using an existing node network interface.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and configures the MetalLB operator to provide a single IP address, 192.168.50.20, for the load balancer services.

[student@workstation ~]$ **lab start non-http-review**

**Instructions**

1. Deploy the virtual-rtsp application to a new non-http-review-rtsp project as the developer user with the developer password, and verify that the virtual-rtsp pod is running.

The application consists of the ~/DO280/labs/non-http-review/virtual-rtsp.yaml file.

* 1. Log in to your OpenShift cluster as the developer user with the developer password.
  2. [student@workstation ~]$ **oc login -u developer -p developer \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

...output omitted...

* 1. Change to the ~/DO280/labs/non-http-review directory.

[student@workstation ~]$ **cd ~/DO280/labs/non-http-review**

* 1. Create a non-http-review-rtsp project.
  2. [student@workstation non-http-review]$ **oc new-project non-http-review-rtsp**
  3. Now using project "non-http-review-rtsp" on server ...

...output omitted...

* 1. Use the oc create command to create the virtual-rtsp deployment by using the virtual-rtsp.yaml file.
  2. [student@workstation non-http-review]$ **oc create -f virtual-rtsp.yaml**

deployment.apps/virtual-rtsp created

* 1. List the deployments and pods. Wait for the virtual-rtsp pod to be ready. Press **Ctrl**+**C** to exit the watch command.
  2. [student@workstation non-http-review]$ **watch oc get deployments,pods**
  3. NAME **READY** UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/virtual-rtsp **1/1** 1 1 21s
  5. NAME READY **STATUS** RESTARTS AGE

pod/virtual-rtsp-54d8d6b57d-6jsvm 1/1 **Running** 0 21s

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Expose the virtual-rtsp deployment by using the LoadBalancer service.
   1. Create a load balancer service for the virtual-rtsp deployment.
   2. [student@workstation non-http-review]$ **oc expose deployment/virtual-rtsp \**
   3. **--name=virtual-rtsp-loadbalancer --type=LoadBalancer**

service/virtual-rtsp-loadbalancer exposed

* 1. Retrieve the external IP address of the virtual-rtsp-loadbalancer service.
  2. [student@workstation non-http-review]$ **oc get svc/virtual-rtsp-loadbalancer**
  3. NAME TYPE ... **EXTERNAL-IP** PORT(S)

virtual-rtsp-loadbalancer LoadBalancer ... **192.168.50.20** 8554:32570/TCP

The virtual-rtsp-loadbalancer has the 192.168.50.20 external IP address.

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Access the virtual-rtsp application by using the URL in the media player. Run the totem rtsp://*EXTERNAL-IP*:8554/stream command to play the stream in the media player.
   1. Open the URL in the media player to confirm that the video stream is working correctly.

rtsp://192.168.50.20:8554/stream

[student@workstation non-http-review]$ **totem rtsp://192.168.50.20:8554/stream**

...output omitted...

|  |
| --- |
| A screenshot of a video  Description automatically generated |

Close the media player window after confirming that the video stream works correctly.

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Deploy the nginx deployment to a new non-http-review-nginx project as the developer user with the developer password, and verify that the nginx pod is running. The application consists of the ~/DO280/labs/non-http-review/nginx.yaml file.

### IMPORTANT

The exercise is using an HTTP application as a stand-in for testing connectivity to an external network.

* 1. Create a non-http-review-nginx project.
  2. [student@workstation non-http-review]$ **oc new-project non-http-review-nginx**
  3. Now using project "non-http-review-nginx" on server ...

...output omitted...

* 1. Use the oc apply command to create the nginx deployment by using the nginx.yaml file.
  2. [student@workstation non-http-review]$ **oc apply -f nginx.yaml**

deployment.apps/nginx created

* 1. List the deployments and pods. Wait for the nginx pod to be ready. Press **Ctrl**+**C** to exit the watch command.
  2. [student@workstation non-http-review]$ **watch oc get deployments,pods**
  3. NAME **READY** UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/nginx **1/1** 1 1 53s
  5. NAME READY **STATUS** RESTARTS AGE

pod/nginx-649779cbd-d6sbv 1/1 **Running** 0 53s

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Configure a network attachment definition for the ens4 interface, so that the isolated network can be attached to a pod.

The master01 node has two Ethernet interfaces. The ens3 interface is the main network interface of the cluster. The ens4 interface is an additional network interface for exercises that require an additional network. The ens4 interface is attached to a 192.168.51.0/24 network, with the 192.168.51.10 IP address.

You can modify the ~/DO280/labs/non-http-review/network-attachment-definition.yaml file to configure a network attachment definition by using the following parameters:

| **Parameter** | **Value** |
| --- | --- |
| name | custom |
| type | host-device |
| device | ens4 |
| ipam.type | static |
| ipam.addresses | {"address": "192.168.51.10/24"} |

* 1. Log in to your OpenShift cluster as the admin user with the redhatocp password.
  2. [student@workstation non-http-review]$ **oc login -u admin -p redhatocp \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

...output omitted...

* 1. Edit the ~/DO280/labs/non-http-review/network-attachment-definition.yaml file. Use the custom name, the host-device type, and the ens4 device. Configure IP address management to use the static type, with the 192.168.51.10/24 address.
  2. apiVersion: k8s.cni.cncf.io/v1
  3. kind: NetworkAttachmentDefinition
  4. metadata:
  5. name: **custom**
  6. spec:
  7. config: |-
  8. {
  9. "cniVersion": "0.3.1",
  10. "name": **"custom"**,
  11. "type": **"host-device"**,
  12. **"device": "ens4",**
  13. "ipam": {
  14. "type": "**static**",
  15. **"addresses": [**
  16. **{"address": "192.168.51.10/24"}**
  17. **]**
  18. }

}

* 1. Use the oc create command to create the network attachment definition.
  2. [student@workstation non-http-review]$ **oc create -f \**
  3. **network-attachment-definition.yaml**

networkattachmentdefinition.k8s.cni.cncf.io/custom created

[Hide Solution](https://rol.redhat.com/rol/app/)

1. The nginx application does not contain any services, so the application is not accessible outside the pod network.

Assign the ens4 network interface exclusively to the nginx pod, by using the custom network attachment definition. Edit the nginx deployment to add the k8s.v1.cni.cncf.io/networks annotation with the custom value as the developer user with the developer password.

* 1. Log in to the OpenShift cluster as the developer user with the developer password.
  2. [student@workstation non-http-review]$ **oc login -u developer -p developer \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

...output omitted...

* 1. Edit the ~/DO280/labs/non-http-review/nginx.yaml file to add the k8s.v1.cni.cncf.io/networks annotation with the custom value.
  2. ...output omitted...
  3. spec:
  4. replicas: 1
  5. selector:
  6. matchLabels:
  7. app: nginx
  8. strategy:
  9. type: Recreate
  10. template:
  11. metadata:
  12. labels:
  13. app: nginx
  14. **annotations:**
  15. **k8s.v1.cni.cncf.io/networks: custom**
  16. spec:
  17. containers:

...output omitted...

* 1. Use the oc apply command to add the annotation.
  2. [student@workstation non-http-review]$ **oc apply -f nginx.yaml**

deployment.apps/nginx configured

* 1. Wait for the nginx pod to be ready. Press **Ctrl**+**C** to exit the watch command.
  2. [student@workstation non-http-review]$ **watch oc get deployments,pods**
  3. NAME **READY** UP-TO-DATE AVAILABLE AGE
  4. deployment.apps/nginx **1/1** 1 1 34m
  5. NAME READY **STATUS** RESTARTS AGE

pod/nginx-6f45d9f89-wp2gg 1/1 **Running** 0 53s

* 1. Examine the k8s.v1.cni.cncf.io/networks-status annotation in the pod.
  2. [student@workstation ~]$ **oc get pod nginx-*6f45d9f89-wp2gg* \**
  3. **-o jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status}'**
  4. [{
  5. "name": "ovn-kubernetes",
  6. "interface": "eth0",
  7. "ips": [
  8. "10.8.0.82"
  9. ],
  10. "mac": "0a:58:0a:08:00:52",
  11. "default": true,
  12. "dns": {}
  13. },{
  14. "name": **"non-http-review-nginx/custom"**,
  15. "interface": "net1",
  16. "ips": [
  17. **"192.168.51.10"**
  18. ],
  19. "mac": "52:54:00:01:33:0a",
  20. "dns": {}

}]

### NOTE

The period is the JSONPath field access operator. Normally, you use the period to access parts of the resource, such as in the .metadata.annotations JSONPath expression. To access fields that contain periods with JSONPath, you must escape the periods with a backslash (\).

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Verify that you can access the nginx application from the utility machine by using the following URL:

http://*isolated-network-IP-address*:8080

* 1. Use the ssh command to connect to the utility machine.
  2. [student@workstation non-http-review]$ **ssh utility**
  3. ...output omitted...

[student@utility ~]$

* 1. Verify that the nginx application is accessible. Use the IP address on the isolated network to access the nginx application.
  2. [student@utility ~]$ **curl 'http://192.168.51.10:8080/'**
  3. <html>
  4. <body>
  5. <h1>Hello, world from nginx!</h1>
  6. </body>

</html>

* 1. Exit the SSH session to go back to the workstation machine.
  2. [student@utility ~]$ **exit**
  3. logout
  4. Connection to utility closed.

[student@workstation non-http-review]$

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Verify that you cannot access the nginx application from the workstation machine, because the workstation machine cannot access the isolated network.
   1. Verify that the nginx application is not accessible from the workstation machine.
   2. [student@workstation non-http-review]$ **curl 'http://192.168.51.10:8080/'**

curl: (7) Failed to connect to 192.168.51.10 port 8080: Connection timed out

* 1. Change to the student HOME directory.
  2. [student@workstation non-http-review]$ **cd**

[student@workstation ~]$

1. [Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade non-http-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish non-http-review**

# Chapter 6.  Enable Developer Self-Service

[Project and Cluster Quotas](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06)

[Guided Exercise: Project and Cluster Quotas](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s02)

[Per-Project Resource Constraints: Limit Ranges](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s03)

[Guided Exercise: Per-Project Resource Constraints: Limit Ranges](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s04)

[The Project Template and the Self-Provisioner Role](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s05)

[Guided Exercise: The Project Template and the Self-Provisioner Role](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s06)

[Lab: Enable Developer Self-Service](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s07)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch06s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Configure clusters for safe self-service by developers from multiple teams, and disallow self-service if operations staff must provision projects. |
| **Objectives** | * Configure compute resource quotas and Kubernetes resource count quotas per project and cluster-wide. * Configure default and maximum compute resource requirements for pods per project. * Configure default quotas, limit ranges, role bindings, and other restrictions for new projects, and the allowed users to self-provision new projects. |
| **Sections** | * Project and Cluster Quotas (and Guided Exercise) * Per-Project Resource Constraints: Limit Ranges (and Guided Exercise) * The Project Template and the Self-Provisioner Role (and Guided Exercise) |
| **Lab** | * Enable Developer Self-Service |

## **Project and Cluster Quotas**

### **Objectives**

* Configure compute resource quotas and Kubernetes resource count quotas per project and cluster-wide.

### **Limiting Workloads**

Kubernetes clusters can run heterogeneous workloads across many compute nodes. By using Kubernetes role-based access control (RBAC), cluster administrators can allow users to create workloads on their own. Although RBAC can limit the kinds of resources that users can create, administrators might want further measures to ensure correct operation of the cluster.

Clusters have limited resources, such as CPU, RAM, and storage. If workloads on a cluster exceed the available resources, then workloads might not work correctly. A cluster that is configured to autoscale might also incur unwanted economic costs if the cluster scales to accommodate unexpected workloads.

To help with this issue, Kubernetes workloads can reserve resources and declare resource limits. Workloads can specify the following properties:

**Resource limits**

Kubernetes can limit the resources that a workload consumes. Workloads can specify an upper bound of the resources that they expect to use under normal operation. If a workload malfunctions or has unexpected load, then resource limits prevent the workload from consuming an excessive amount of resources and impacting other workloads.

**Resource requests**

Workloads can declare their minimum required resources. Kubernetes tracks requested resources by workloads, and prevents deployments of new workloads if the cluster has insufficient resources. Resource requests ensure that workloads get their needed resources.

These measures prevent workloads from affecting other workloads. However, cluster administrators might need to prevent other risks.

For example, users might mistakenly create unwanted workloads. The resource requests of those unwanted workloads can prevent legitimate workloads from executing.

By dividing workloads into namespaces, Kubernetes can offer enhanced protection features. The namespace structure often mirrors the organization that runs the cluster. Kubernetes introduces resource quotas to limit resource usage by the combined workloads in a namespace.

### **Resource Quotas**

Kubernetes administrators can create resources of the ResourceQuota type in a namespace for this purpose. When a resource quota exists in a namespace, Kubernetes prevents the creation of workloads that exceed the quota.

Whereas quota features in other systems often act on users or groups of users, Kubernetes resource quotas act on namespaces.

apiVersion: v1

kind: ResourceQuota

metadata:

name: memory

namespace: example

spec:

hard:

limits.memory: 4Gi

requests.memory: 2Gi

scopes: {}

scopeSelector: {}

|  |  |
| --- | --- |
|  | The hard key lists restrictions. |
|  | The scopes and scopeSelector keys define which namespace resources the quota applies to. This course does not cover those keys. |

The following sections describe the compute and object count quotas that you can include in the hard key. Other components can define other quotas and enforce them.

#### **Compute Resource Quotas**

You can set the following compute quotas:

* limits.cpu
* limits.memory
* requests.cpu
* requests.memory

Limit quotas interact with resource limits, and request quotas interact with resource requests.

Limit quotas control the maximum compute resources that the workloads in a namespace can consume. Consider a namespace where all workloads have a memory limit. No individual workload can consume enough memory to cause a problem. However, because users can create any number of workloads, the workloads of a namespace can consume enough memory to cause a problem for workloads in other namespaces. If you set a namespace memory usage limit, then the workloads in the namespace cannot consume more memory than this limit.

Request quotas control the maximum resources that workloads in a namespace can reserve. If you do not set namespace request quotas, then a single workload can request any quantity of resources, such as RAM or CPU. This request can cause further requests in other namespaces to fail. By setting namespace request quotas, the total requested resources by workloads in a namespace cannot exceed the quota.

Excessive quotas can cause resource underutilization and can limit workload performance unnecessarily.

After setting any compute quota, all workloads must define the corresponding request or resource limit. For example, if you create a limits.cpu quota, then the workloads that you create require the resources.limits.cpu key.

#### **Object Count Quotas**

A quota can also limit the number of resources of a given type in a namespace. For example, you can create a quota that prevents the creation of more than 10 deployments in a namespace.

Clusters store resource definitions in a backing store. Kubernetes backing stores are databases, and like any other database, the more data that they store, the more resources are needed for adequate performance. Namespaces with many resources can impact Kubernetes performance. Additionally, any process that creates cluster resources might malfunction and create unwanted resources.

Setting object count quotas can limit the damage from accidents, and maintain adequate cluster performance.

### NOTE

Red Hat validates the performance of OpenShift up to a specific number of objects in a set of configurations. If you are planning a large cluster, then these results can help you to size the cluster and to establish object count quotas.

See the references section for more information.

Some Kubernetes resources might affect external systems. For example, creating a persistent volume might create an entity in the storage provider. Many persistent volumes might cause issues in the storage provider. Examine the systems that your cluster interacts with to learn about possible resource constraints, and establish object count quotas to prevent issues.

Use the count/*resource\_type* syntax to set a quota for resources of the core group. Use the oc api-resources command with an empty api-group parameter to list resources of the core group.

[user@host ~]$ **oc api-resources --api-group="" --namespaced=true**

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

...output omitted...

For resources in other groups, use the count/*resource\_type*.*group* syntax.

Kubernetes initially supported quotas for a limited set of resource types. These quotas do not use the count/*resource\_type* syntax. You might find a services quota instead of a count/services quota. The Resource Quotas reference further describes these quotas.

### **Applying Project Quotas**

Navigate to **Administration** → **ResourceQuotas** to create a resource quota from the web console. The YAML editor loads an example resource quota that you can edit for your needs.

You can also use the oc command to create a resource quota. The oc command can create resource quotas without requiring a complete resource definition. Execute the oc create resourcequota --help command to display examples and help for creating resource quotas without a complete resource definition.

For example, execute the following command to create a resource quota that limits the number of pods in a namespace:

[user@host ~]$ **oc create resourcequota example --hard=count/pods=1**

resourcequota/example created

The previous command is equivalent to creating a resource quota with the following definition:

apiVersion: v1

kind: ResourceQuota

metadata:

name: example

spec:

hard:

count/pods: "1"

After creating a resource quota, the status key in the resource describes the current values and limits in the quota.

[user@host ~]$ **oc get quota example -o yaml**

apiVersion: v1

kind: ResourceQuota

metadata:

creationTimestamp: "2024-01-30T17:59:52Z"

name: example

namespace: default

resourceVersion: "193658"

uid: df12b484-4e78-4920-acb4-e04ab286a4a1

spec:

hard:

count/pods: "1"

status:

hard:

count/pods: "1"

used:

count/pods: "0"

The oc get and oc describe commands show resource quota information in a custom format. The oc get command displays the status of the quota in resource lists:

[user@host ~]$ **oc get quota**

NAME AGE REQUEST LIMIT

example 9m54s count/pods: 1/1

Resource quotas generate the kube\_resourcequota metric. You can examine this metric for planning and trend analysis.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 6.1:

The kube\_resourcequota metric

### **Troubleshooting Resource Quotas**

Because resource quotas are extensible, Kubernetes cannot verify that a resource quota is correct. For example, the following command creates a resource quota that has no effect:

[user@host ~]$ **oc create resourcequota example --hard=count/deployment=1**

resourcequota/example created

The correct syntax for limiting the number of deployments is count/deployments.apps.

To ensure that a resource quota is correct, you can use the following procedures:

* Create a quota with an artificially low value in a testing environment, and ensure that the resource quota has an effect.
* Review the quota status.

For example, if a namespace contains a deployment, then an incorrectly defined resource quota shows 0 deployments:

[user@host ~]$ **oc get resourcequota**

NAME AGE REQUEST LIMIT

example 2m47s count/deployment: **0**/1

However, a correctly defined resource quota shows the deployment:

[user@host ~]$ **oc get resourcequota**

NAME AGE REQUEST LIMIT

example 4s count/deployments.apps: **1**/1

Exceeding a quota often produces an error immediately. For example, if you create a deployment that exceeds the deployment quota, then the deployment creation fails.

[user@host ~]$ **oc create deployment --image=nginx hello**

error: failed to create deployment: deployments.apps "hello" is forbidden: exceeded quota: example, requested: count/deployments.apps=1, used: count/deployments.apps=1, limited: count/deployments.apps=1

However, some quotas do not cause operations to fail immediately. For example, if you set a resource quota for pods, then creating a deployment appears to succeed, but the deployment never becomes available. When a resource quota is acting indirectly, namespace events might provide further information.

[user@host ~]$ **oc get event --sort-by .metadata.creationTimestamp**

LAST SEEN TYPE REASON OBJECT MESSAGE

...output omitted...

10s Normal ScalingReplicaSet deployment/hello Scaled up replica set hello-5cdfd9c858 to 1

9s Warning FailedCreate replicaset/hello-5cdfd9c858 Error creating: pods "hello-5cdfd9c858-zsgn9" is forbidden: exceeded quota: example, requested: count/pods=1, used: count/pods=1, limited: count/pods=1

5s Warning FailedCreate replicaset/hello-5cdfd9c858 (combined from similar events): Error creating: pods "hello-5cdfd9c858-h2dv4" is forbidden: exceeded quota: example, requested: count/pods=1, used: count/pods=1, limited: count/pods=1

The web console also shows quota information. Navigate to **Administration** → **ResourceQuotas** to view resource quotas and their status. The project pages on both the developer and administrator perspectives also show the quotas that apply to a specific project.

### **Creating Quotas Across Multiple Projects**

Cluster administrators can use resource quotas to apply restrictions to namespaces.

Resource restrictions often follow organization structure. Although namespaces often reflect organization structure, cluster administrators might apply restrictions to resources without being limited to a single namespace.

For example, a group of developers manages many namespaces. Namespace quotas can limit RAM usage per namespace. However, a cluster administrator cannot limit total RAM usage by all workloads that the group of developers manages.

OpenShift introduces cluster resource quotas for those scenarios.

Cluster resource quotas follow a similar structure to namespace resource quotas. However, cluster resource quotas use selectors to choose which namespaces the quota applies to.

Cluster resource quotas selectors use set-based requirements.

The following example shows a cluster resource quota:

apiVersion: quota.openshift.io/v1

kind: ClusterResourceQuota

metadata:

name: example

spec:

quota:

hard:

limits.cpu: 4

selector:

annotations: {}

labels:

matchLabels:

kubernetes.io/metadata.name: example

|  |  |
| --- | --- |
|  | The quota key contains the quota definition. This key follows the structure of the ResourceQuota specification. The hard key is nested inside the quota key, instead of being directly nested inside the spec key as in resource quotas. |
|  | The selector key defines which namespaces the cluster resource quota applies to. Other Kubernetes features, such as services and network policies, use the same selectors. |

Navigate to **Administration** → **CustomResourceDefinitions** to create a cluster resource quota with the web console.

You can also use the oc command to create a cluster quota. The oc command can create quotas without requiring a complete resource definition. Execute the oc create clusterresourcequota --help command to display examples and help about creating cluster resource quotas without a complete resource definition.

For example, execute the following command to create a resource quota that limits total CPU requests. The quota limits the total CPU requests on namespaces that have the group label with the dev value.

[user@host ~]$ **oc create clusterresourcequota example --project-label-selector=group=dev --hard=requests.cpu=10**

clusterresourcequota/example created

Cluster resource quotas collect total resource usage across namespaces and enforce the limits. The following example shows the status of the previous cluster resource quota:

apiVersion: quota.openshift.io/v1

kind: ClusterResourceQuota

metadata:

name: example

spec:

quota:

hard:

requests.cpu: "10"

selector:

annotations: null

labels:

matchLabels:

group: dev

status:

namespaces:

- namespace: example-3

status:

hard:

requests.cpu: "10"

used:

requests.cpu: 500m

- namespace: example-2

status:

hard:

requests.cpu: "10"

used:

requests.cpu: 250m

\_...output omitted...\_

total:

hard:

requests.cpu: "10"

used:

requests.cpu: 2250m

|  |  |
| --- | --- |
|  | The namespaces key lists the namespaces that the quota applies to. For each namespace, the used key shows the current utilization. |
|  | The total key aggregates the data in the namespaces key. |

Users might not have read access to cluster resource quotas. OpenShift creates resources of the AppliedClusterResourceQuota type in namespaces that are affected by cluster resource quotas. Project administrators can review quota usage by reviewing the AppliedClusterResourceQuota resources. For example, use the oc describe command to view the cluster resource quotas that apply to a specific namespace:

[user@host ~]$ **oc describe AppliedClusterResourceQuota -n example-2**

Name: example

Created: 9 minutes ago

Labels: <none>

Annotations: <none>

Namespace Selector: ["example-3" "example-2" "example-4" "example-1"]

Label Selector: group=dev

AnnotationSelector: map[]

Resource Used Hard

-------- ---- ----

requests.cpu 2250m 10

### NOTE

The --all-namespaces argument to oc commands such as the get and describe commands does not work with AppliedClusterResourceQuota resources. These resources are listed only when you select a namespace.

Navigate to **Administration** → **ResourceQuotas** to view quotas and their status. This page displays cluster quotas along with namespace quotas. Although you can view resources of the ClusterResourceQuota type and create resources of the ResourceQuota type in the ResourceQuotas page, you cannot create objects of the ClusterResourceQuota in this page.

The project pages on both the developer and administrator perspectives also show the cluster quotas that apply to a specific project.

### REFERENCES

For more information, refer to the Quotas chapter in the Red Hat OpenShift Container Platform 4.14 Building Applications documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#quotas>

For more information about object counts, refer to the Planning Your Environment According to Object Maximums chapter in the Red Hat OpenShift Container Platform 4.14 Scalability and Performance documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/scalability_and_performance/index#planning-your-environment-according-to-object-maximums>

[Requests and Limits](https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits)

[Resource Quotas](https://kubernetes.io/docs/concepts/policy/resource-quotas/)

## **Guided Exercise: Project and Cluster Quotas**

Configure quotas for a project so that applications cannot scale to consume all capacity of a cluster node.

**Outcomes**

* Verify that requesting resources in one namespace can prevent creation of workloads in different namespaces.
* Set a quota to prevent workloads in a namespace from requesting excessive resources.
* Verify that you can continue to create workloads in different namespaces.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and deletes the namespaces that you use in this exercise.

[student@workstation ~]$ **lab start selfservice-quotas**

**Instructions**

1. Log in to your OpenShift cluster as the developer user with the developer password.
   1. Log in to the cluster as the developer user.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

1. Create a selfservice-quotas project.
   1. Use the oc new-project command to create the project.
   2. [student@workstation ~]$ **oc new-project selfservice-quotas**
   3. Now using project "selfservice-quotas" on server "https://api.ocp4.example.com:6443".

...output omitted...

1. Create a deployment with a container that requests one CPU.
   1. Use the oc create command to create the deployment.
   2. [student@workstation ~]$ **oc create deployment test \**
   3. **--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx**

deployment.apps/test created

* 1. Use the oc set resources command to request one CPU in the container specification.
  2. [student@workstation ~]$ **oc set resources deployment test --requests=cpu=1**

deployment.apps/test resource requirements updated

* 1. Use the oc get command to ensure that the deployment starts a pod correctly.
  2. [student@workstation ~]$ **oc get pod,deployment**
  3. NAME READY STATUS RESTARTS AGE
  4. pod/test-8b9fdfbd9-bltlc 1/1 Running 0 13s
  5. NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/test 1/1 1 1 49s

Execute the command until the deployment and the pod are ready.

1. Try to scale the deployment to eight replicas.
   1. Use the oc scale command to scale the deployment.
   2. [student@workstation ~]$ **oc scale deployment test --replicas=8**

deployment.apps/test scaled

* 1. Use the oc get command to view pods and deployments.
  2. [student@workstation ~]$ **oc get pod,deployment**
  3. NAME READY STATUS RESTARTS AGE
  4. pod/test-6c66b55cb5-2kclt 1/1 Running 0 48m
  5. pod/test-6c66b55cb5-5n58r 0/1 Pending 0 5s
  6. pod/test-6c66b55cb5-8x929 0/1 Pending 0 5s
  7. pod/test-6c66b55cb5-blgms 0/1 Pending 0 5s
  8. pod/test-6c66b55cb5-d6z42 1/1 Running 0 6s
  9. pod/test-6c66b55cb5-fc8bk 0/1 Pending 0 5s
  10. pod/test-6c66b55cb5-t29dh 0/1 Pending 0 6s
  11. pod/test-6c66b55cb5-xqr66 0/1 Pending 0 6s
  12. NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/test 2/8 8 2 54m

Out of eight pods that the deployment creates, only some of them change to Running status. The other pods stay in Pending status. Not all replicas of the deployment are ready and available.

* 1. Use the oc get command to list events. Sort the events by their creation timestamp.
  2. [student@workstation ~]$ **oc get event --sort-by .metadata.creationTimestamp**
  3. LAST SEEN TYPE REASON OBJECT MESSAGE
  4. ...output omitted...
  5. 3m58s Normal ScalingReplicaSet deployment/test Scaled up replica set test-6c66b55cb5 to 8
  6. 3m58s Normal Scheduled pod/test-6c66b55cb5-d6z42 Successfully assigned selfservice-quotas/test-6c66b55cb5-d6z42 to master01
  7. 3m57s **Warning FailedScheduling** pod/test-6c66b55cb5-5n58r 0/1 nodes are available: 1 **Insufficient cpu. preemption: 0/1 nodes are available:** 1 No preemption victims found for incoming pod..

...output omitted...

Replicas fail to schedule, because the cluster has insufficient CPU.

1. Examine the cluster as an administrator.
   1. Log in to the cluster as the admin user with the redhatocp password.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp**
   3. Login successful.

...output omitted...

* 1. Use the oc adm top command to display the resource usage of nodes.
  2. [student@workstation ~]$ **oc adm top node**
  3. NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

master01 772m 14% 10185Mi 68%

The cluster does not show high CPU usage.

* 1. Use the oc describe command to view the node details.
  2. [student@workstation ~]$ **oc describe node/master01**
  3. Name: master01
  4. ...output omitted...
  5. Capacity:
  6. **cpu: 6**
  7. ...output omitted...
  8. Allocatable:
  9. **cpu: 5500m**
  10. ...output omitted...
  11. Allocated resources:
  12. (Total limits may be over 100 percent, i.e., overcommitted.)
  13. Resource Requests Limits
  14. -------- -------- ------
  15. cpu **4627m (84%)** 0 (0%)
  16. memory 12102Mi (81%) 0 (0%)
  17. ephemeral-storage 0 (0%) 0 (0%)
  18. hugepages-1Gi 0 (0%) 0 (0%)
  19. hugepages-2Mi 0 (0%) 0 (0%)

...output omitted...

The node has a capacity of six CPUs, and has more than five allocatable CPUs. However, over five CPUs are requested, so less than one CPU is available for new workloads.

1. Create a test project as an administrator, and verify that you cannot create new workloads that request a CPU.
   1. Use the oc new-project command to create the project.
   2. [student@workstation ~]$ **oc new-project test**
   3. Now using project "test" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Use the oc create command to create the deployment.
  2. [student@workstation ~]$ **oc create deployment test \**
  3. **--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx**

deployment.apps/test created

* 1. Use the oc set resources command to request one CPU in the container specification.
  2. [student@workstation ~]$ **oc set resources deployment test --requests=cpu=1**

deployment.apps/test resource requirements updated

* 1. Use the oc get command to review the pods and deployments in the test namespace.
  2. [student@workstation ~]$ **oc get pod,deployment**
  3. NAME READY STATUS RESTARTS AGE
  4. pod/test-8b9fdfbd9-rrn7t 0/1 Pending 0 8s
  5. pod/test-c454765f-vkt96 1/1 Running 0 100s
  6. NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/test 1/1 1 1 100s

The deployment created one pod before adding the CPU request. When you updated the deployment to request a CPU, the deployment tried to replace the pod to add the CPU request. The new pod is in the Pending state, because the cluster has less than one CPU available to request.

The workload in the selfservice-quotas namespace prevents the creation of workloads in other namespaces.

* 1. Use the oc delete command to delete the test namespace.
  2. [student@workstation ~]$ **oc delete namespace test**

namespace "test" deleted

1. As an administrator, scale the deployment to one replica.
   1. Use the oc project command to switch to the selfservice-quotas project.
   2. [student@workstation ~]$ **oc project selfservice-quotas**

Now using project "selfservice-quotas" on server "https://api.ocp4.example.com:6443".

* 1. Use the oc scale command to scale the test deployment to one replica.
  2. [student@workstation ~]$ **oc scale deployment test --replicas=1**

deployment.apps/test scaled

1. Create a quota to prevent workloads in the selfservice-quotas namespace from requesting more than one CPU.
   1. Use the oc create command to create the quota.
   2. [student@workstation ~]$ **oc create quota one-cpu --hard=requests.cpu=1**

resourcequota/one-cpu created

* 1. Use the oc get command to verify the quota.
  2. [student@workstation ~]$ **oc get quota one-cpu -o yaml**
  3. apiVersion: v1
  4. kind: ResourceQuota
  5. metadata:
  6. creationTimestamp: "2024-01-30T18:26:49Z"
  7. name: one-cpu
  8. namespace: selfservice-quotas
  9. ...output omitted...
  10. spec:
  11. hard:
  12. requests.cpu: "1"
  13. status:
  14. hard:
  15. requests.cpu: "1"
  16. used:

requests.cpu: "1"

The test deployment already requests one CPU.

1. Try to scale the deployment to eight replicas and to create a second deployment.
   1. Use the oc scale command to scale the deployment.
   2. [student@workstation ~]$ **oc scale deployment test --replicas=8**

deployment.apps/test scaled

* 1. Use the oc create command to create a second deployment.
  2. [student@workstation ~]$ **oc create deployment test2 \**
  3. **--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx**

deployment.apps/test2 created

* 1. Use the oc get command to review pods and deployments.
  2. [student@workstation ~]$ **oc get pod,deployment**
  3. NAME READY STATUS RESTARTS AGE
  4. pod/test-6c66b55cb5-mdxjl 1/1 Running 0 2m58s
  5. NAME READY UP-TO-DATE AVAILABLE AGE
  6. deployment.apps/test 1/8 1 1 3m20s

deployment.apps/test2 0/1 0 0 14s

The test deployment creates only two pods. The second deployment does not create any pods.

* 1. Use the oc get command to examine the quota status.
  2. [student@workstation ~]$ **oc get quota one-cpu -o yaml**
  3. apiVersion: v1
  4. kind: ResourceQuota
  5. metadata:
  6. name: one-cpu
  7. namespace: selfservice-quotas
  8. ...output omitted...
  9. spec:
  10. hard:
  11. requests.cpu: "1"
  12. status:
  13. hard:
  14. requests.cpu: "1"
  15. used:

requests.cpu: "1"

The used status is kept at 1 because the test2 deployment can't request more resources in the quota.

* 1. Use the oc get command to list events. Sort the events by their creation timestamp.
  2. [student@workstation ~]$ **oc get event --sort-by .metadata.creationTimestamp**
  3. LAST SEEN TYPE REASON OBJECT MESSAGE
  4. ...output omitted...
  5. 4m42s Warning FailedCreate replicaset/test-6c66b55cb5 (combined from similar events): Error creating: pods "`test`-6c66b55cb5-djrr9" is forbidden: **exceeded quota: one-cpu**, requested: requests.cpu=1, used: requests.cpu=2, limited: requests.cpu=2
  6. 9m3s Warning FailedCreate replicaset/**test2**-7b9df44445 Error creating: pods "test2-7b9df44445-98wxp" is forbidden: **failed quota: one-cpu: must specify requests.cpu** for: hello-world-nginx

...output omitted...

The test deployment cannot create further pods, because the new pods would exceed the quota. The test2 deployment cannot create pods, because the deployment does not set a CPU request.

1. Create a test project to verify that you can create new workloads in other namespaces that request CPU resources.
   1. Use the oc new-project command to create the project.
   2. [student@workstation ~]$ **oc new-project test**
   3. Now using project "test" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Use the oc create command to create the deployment.
  2. [student@workstation ~]$ **oc create deployment test --image \**
  3. **registry.ocp4.example.com:8443/redhattraining/hello-world-nginx**

deployment.apps/test created

* 1. Use the oc set resources command to request one CPU in the container specification.
  2. [student@workstation ~]$ **oc set resources deployment test --requests=cpu=1**

deployment.apps/test resource requirements updated

* 1. Use the oc get command to review the pods and deployments in the test namespace.
  2. [student@workstation ~]$ **oc get pod,deployment**
  3. NAME READY STATUS RESTARTS AGE
  4. pod/test-8b9fdfbd9-447w9 1/1 Running 0 21s
  5. NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/test 1/1 1 1 51s

Even though you cannot create further workloads in the selfservice-quotas namespace, you can create workloads that request CPUs in other namespaces when the node has CPUs and memory available.

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish selfservice-quotas**

## **Per-Project Resource Constraints: Limit Ranges**

### **Objectives**

* Configure default and maximum compute resource requirements for pods per project.

### **Managing Namespace Resources**

Cluster administrators can set resource quotas on namespaces. Namespace quotas limit the resources that workloads in a namespace use. Quotas address resource management at the cluster level.

Kubernetes users might have further resource management needs within a namespace.

* Users might accidentally create workloads that consume too much of the namespace quota. These unwanted workloads might prevent other workloads from running.
* Users might forget to set workload limits and requests, or might find it time-consuming to configure limits and requests. When a namespace has a quota, creating workloads fails if the workload does not define values for the limits or requests in the quota.

Kubernetes introduces limit ranges to help with these issues. Limit ranges are namespaced objects that define limits for workloads within the namespace.

### **Limit Ranges**

The following YAML file shows an example limit range:

apiVersion: v1

kind: LimitRange

metadata:

name: mem-limit-range

namespace: default

spec:

limits:

- default:

memory: 512Mi

defaultRequest:

memory: 256Mi

type: Container

Limit ranges can specify the following limit types:

**Default limit**

Use the default key to specify default limits for workloads.

**Default request**

Use the defaultRequest key to specify default requests for workloads.

**Maximum**

Use the max key to specify the maximum value of both requests and limits.

**Minimum**

Use the min key to specify the minimum value of both requests and limits.

**Limit-to-request ratio**

The maxLimitRequestRatio key controls the relationship between limits and requests. If you set a ratio of two, then the resource limit cannot be more than twice the request.

This course does not cover limit-to-request ratios in detail.

Limit ranges can apply to containers, pods, images, image streams, and persistent volume claims.

#### **Setting Maximum and Minimum Limit Ranges**

When you set the max key, users cannot create workloads that declare limits or that make resource requests over the maximum.

Use maximums to prevent accidentally high resource requests and limits. These situations can exhaust quotas and cause other issues.

Consider allowing users who create workloads to edit maximum limit ranges. Although maximum limit ranges act as a convenient safeguard, excessively low limits can prevent users from creating legitimate workloads.

Minimum limit ranges are useful to ensure that users create workloads with enough requests and limits. If users create such workloads often, then consider adding minimums.

#### **Setting Defaults**

Defaults are convenient in namespaces with quotas, and eliminate a need to declare limits explicitly in each workload. When a quota is present, all workloads must specify the corresponding limits and requests. When you set the default and defaultRequest keys, workloads use the requests and limits from the limit range by default.

Defaults are especially convenient in scenarios where many workloads are created dynamically. For example, continuous integration tools might run tests for each change to a source code repository. Each test can create multiple workloads. Because many tests can run concurrently, the resource usage of testing workloads can be significant. Setting quotas for testing workloads is often needed to limit resource usage. If you set CPU and RAM quotas for requests and limits, then the continuous integration tool must set the corresponding limits in every testing workload. Setting defaults can save time with configuring limits. However, determining appropriate defaults might be complex for namespaces with varied workloads.

### **Creating Limit Ranges**

Consider a namespace with the following quota:

apiVersion: v1

kind: ResourceQuota

metadata:

name: example

namespace: example

spec:

hard:

limits.cpu: "8"

limits.memory: 8Gi

requests.cpu: "4"

requests.memory: 4Gi

The following command creates a deployment:

[user@host ~]$ **oc create deployment example --image=*image***

deployment.apps/example created

The quota prevents the deployment from creating pods:

[user@host ~]$ **oc get event --sort-by .metadata.creationTimestamp**

LAST SEEN TYPE REASON OBJECT MESSAGE

...output omitted...

13s Warning FailedCreate replicaset/example-74c57c8dff Error creating: pods "example-74c57c8dff-rzl7w" is forbidden: failed quota: example: must specify limits.cpu for: hello-world-nginx; limits.memory for: hello-world-nginx; requests.cpu for: hello-world-nginx; requests.memory for: hello-world-nginx

...output omitted...

The following limit range includes all types of limits:

apiVersion: v1

kind: LimitRange

metadata:

name: example

namespace: example

spec:

limits:

- default:

cpu: 500m

memory: 512Mi

defaultRequest:

cpu: 250m

memory: 256Mi

max:

cpu: "1"

memory: 1Gi

min:

cpu: 125m

memory: 128Mi

type: Container

Limit ranges do not affect existing pods. If you delete the deployment and run the oc create command again, then the deployment creates a pod with the applied limit range.

[user@host ~]$ **oc describe pod**

...output omitted...

Containers:

hello-world-nginx:

Limits:

cpu: 500m

memory: 512Mi

Requests:

cpu: 250m

memory: 256Mi

...output omitted...

The values correspond to the default and defaultRequest keys in the limit range.

The deployment does not contain any limits in the specification. The Kubernetes API server includes an admission controller that enforces limit ranges. The controller affects pod definitions, but not deployments, stateful sets, or other workloads.

You can replace the CPU limit, or add other resource specifications, by using the oc set resources command:

[user@host ~]$ **oc set resources deployment example --limits=cpu=*new-cpu-limit***

You can experiment with different CPU limits.

If you request CPU values outside the range that the min and max keys define, then Kubernetes does not create the pods, and it logs warnings.

[user@host ~]$ **oc get event --sort-by .metadata.creationTimestamp**

LAST SEEN TYPE REASON OBJECT MESSAGE

...output omitted...

5m43s Warning FailedCreate replicaset/example-7c4dfc5fb8 Error creating: pods "example-7c4dfc5fb8-q7x94" is forbidden: maximum cpu usage per Container is 1, but limit is 1200m

...output omitted...

5m26s Warning FailedCreate replicaset/example-798d65c854 Error creating: pods "example-798d65c854-b94k8" is forbidden: minimum cpu usage per Container is 125m, but request is 100m

...output omitted...

### NOTE

When you experiment with deployments and resource quotas, consider what happens when you modify a deployment. Modifications create a replacement replica set, and the existing replica set also continues to run until the rollout completes.

The pods of both replica sets count towards the resource quota.

If the new replica set satisfies the quota, but the combined replica sets exceed the quota, then the rollout cannot complete.

When creating a limit range, you can specify any combination of the default, defaultRequest, min, and max keys. However, if you do not specify the default or defaultRequest keys, then Kubernetes modifies the limit range to add these keys. These keys are copied from the min or max keys. For more predictable behavior, always specify the default and defaultRequest keys if you specify the min or max keys.

Also, the values for CPU or memory keys must follow these rules:

* The max value must be higher than or equal to the default value.
* The default value must be higher than or equal to the defaultRequest value.
* The defaultRequest value must be higher than or equal to the min value.

Do not create conflicting limit ranges in a namespace. For example, if two default CPU values are specified, then it would be unclear which one is applied.

### REFERENCES

For more information, refer to the Restrict Resource Consumption with Limit Ranges section in the Working with Clusters chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-cluster-limit-ranges>

[Limit Ranges](https://kubernetes.io/docs/concepts/policy/limit-range/)

## **The Project Template and the Self-Provisioner Role**

### **Objectives**

* Configure default quotas, limit ranges, role bindings, and other restrictions for new projects, and the allowed users to self-provision new projects.

### **Project Creation**

Kubernetes provides namespaces to isolate workloads.

Namespace metadata has security implications in clusters. For example, policy controllers might use namespace labels to limit capabilities in a namespace. If users can modify namespaces, then malicious users can modify namespace metadata to override security measures.

Additionally, namespaces are not namespaced. Therefore, granting granular access to namespaces poses some challenges. For example, with Kubernetes role-based access control, you cannot allow users to list a subset of namespaces. However, to allow users to list their namespaces, you must allow them to list all namespaces.

### NOTE

Listing resources and viewing individual resources are different operations. You can grant users permissions to view specific namespaces, but listing namespaces requires a separate permission.

OpenShift introduces projects to improve security and users' experience of working with namespaces. The OpenShift API server adds the Project resource type. When you make a query to list projects, the API server lists namespaces, filters the visible namespaces to your user, and returns the visible namespaces in project format.

Additionally, OpenShift introduces the ProjectRequest resource type. When you create a project request, the OpenShift API server creates a namespace from a template. By using a template, cluster administrators can customize namespace creation. For example, cluster administrators can ensure that new namespaces have specific permissions, resource quotas, or limit ranges.

These features provide self-service management of namespaces. Cluster administrators can allow users to create namespaces without allowing users to modify namespace metadata. Administrators can also customize the creation of namespaces to ensure that namespaces follow organizational requirements.

### **Planning a Project Template**

You can add any namespaced resource to the project template. For example, you can add resources of the following types:

**Roles and role bindings**

Add roles and role bindings to the template to grant specific permissions in new projects. The default template grants the admin role to the user who requests the project. You can keep this permission or use another similar permission, such as granting the admin role to a group of users. You can also add different permissions, such as more granular permissions over specific resource types.

**Resource quotas and limit ranges**

Add resource quotas to the project template to ensure that all new projects have resource limits. If you add resource quotas, then creating workloads requires explicit resource limit declarations. Consider adding limit ranges to reduce the effort for workload creation.

Even with quotas in all namespaces, users can create projects to continue adding workloads to a cluster. If this scenario is a concern, then consider adding cluster resource quotas to the cluster.

**Network policies**

Add network policies to the template to enforce organizational network isolation requirements.

### **Creating a Project Template**

The oc adm create-bootstrap-project-template command prints a template that you can use to create your own project template.

This template has the same behavior as the default project creation in OpenShift. The template adds a role binding that grants the admin cluster role over the new namespace to the user who requests the project.

Project templates use the same template feature as the oc new-app command.

Execute the following command to create a file with an initial template:

[user@host ~]$ **oc adm create-bootstrap-project-template -o yaml > *file***

This initial template has the following content:

apiVersion: template.openshift.io/v1

kind: Template

metadata:

creationTimestamp: null

name: project-request

objects:

- apiVersion: project.openshift.io/v1

kind: Project

metadata:

annotations:

openshift.io/description: ${PROJECT\_DESCRIPTION}

openshift.io/display-name: ${PROJECT\_DISPLAYNAME}

openshift.io/requester: ${PROJECT\_REQUESTING\_USER}

creationTimestamp: null

name: ${PROJECT\_NAME}

spec: {}

status: {}

- apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

creationTimestamp: null

name: admin

namespace: ${PROJECT\_NAME}

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: admin

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: User

name: ${PROJECT\_ADMIN\_USER}

parameters:

- name: PROJECT\_NAME

- name: PROJECT\_DISPLAYNAME

- name: PROJECT\_DESCRIPTION

- name: PROJECT\_ADMIN\_USER

- name: PROJECT\_REQUESTING\_USER

|  |  |
| --- | --- |
|  | The resources that OpenShift creates in new namespaces |
|  | The project resource |
|  | A role binding to grant the admin role to the requesting user |
|  | The parameters that are available to the template |

When a user requests a project, OpenShift replaces the ${*VARIABLE*} syntax with the parameters of the project request, and creates the objects in the objects key.

Modify the object list to add the required resources for new namespaces.

The YAML output of oc commands that return lists of objects is formatted similarly to the template objects key.

[user@host ~]$ oc get limitrange,resourcequota -o yaml

apiVersion: v1

items:

- apiVersion: v1

kind: LimitRange

metadata:

creationTimestamp: "2024-01-31T17:48:23Z"

name: example

namespace: example

resourceVersion: "881771"

uid: d0c19c60-00a9-4028-acc5-22680f1ea658

spec:

limits:

- default:

cpu: 500m

memory: 512Mi

defaultRequest:

cpu: 250m

memory: 256Mi

max:

cpu: "1"

memory: 1Gi

min:

cpu: 125m

memory: 128Mi

type: Container

- apiVersion: v1

kind: ResourceQuota

metadata:

creationTimestamp: "2024-01-31T17:48:04Z"

name: example

namespace: example

resourceVersion: "881648"

uid: 108f0771-dc11-4289-ae76-6514d58bbece

spec:

hard:

count/pods: "1"

status:

...output omitted...

kind: List

metadata:

resourceVersion: ""

Some common resources in project templates, such as quotas, do not have strict validation. For example, if the previous template contains the count/pod text instead of the count/pods text, then the quota does not work. You can create the project template, and new namespaces contain the quota, but the quota does not have an effect. To define a project template and to reduce the risk of errors, you can perform the following steps:

* Create a namespace.
* Create your chosen resources and test until you get the intended behavior.
* List the resources in YAML format.
* Edit the resource listing to ensure that the definitions create the correct resources. For example, remove elements that do not apply to resource creation, such as the creationTimestamp or status keys.
* Replace the namespace name with the ${PROJECT\_NAME} value.
* Add the list of resources to the project template that the oc adm create-bootstrap-project-template command generates.

### NOTE

Extracting a resource definition from an existing resource might not always produce correct results. Besides including elements that do not apply to resource creation, existing definitions might contain attributes that generate unexpected behavior. For example, a controller might add to resources some annotations that are unsuitable for template definitions.

Even after testing the resources in a test namespace, always verify that the projects that are created from your template have only the required behavior.

Use the oc create command to create the template resource in the openshift-config namespace:

[user@host ~]$ **oc create -f *template* -n openshift-config**

template.template.openshift.io/project-request created

### **Configuring the Project Template**

Update the projects.config.openshift.io/cluster resource to use the new project template. Modify the spec section. By default, the name of the project template is project-request.

apiVersion: config.openshift.io/v1

kind: Project

metadata:

...output omitted...

name: cluster

...output omitted...

**spec:**

**projectRequestTemplate:**

**name: project-request**

A successful update to the projects.config.openshift.io/cluster resource rolls out a new version of the apiserver deployment in the openshift-apiserver namespace. After the new apiserver deployment completes, new projects create the resources in the customized project template.

### NOTE

During the apiserver deployment rollout, API requests can produce unexpected results.

To revert to the original project template, modify the projects.config.openshift.io/cluster resource to clear the spec resource to match the spec: {} format.

### **Managing Self-provisioning Permissions**

Users with the self-provisioner cluster role can create projects. By default, the self-provisioner role is bound to all authenticated users.

Control the binding of the role to limit which users can request new projects.

### IMPORTANT

Remember that users with namespace permissions can create namespaces that do not use the project template.

Use the oc describe command to view the role bindings.

[user@host ~]$ **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

This role binding has an rbac.authorization.kubernetes.io/autoupdate annotation. This annotation protects roles and bindings from modifications that can interfere with the working of clusters. When the API server starts, the cluster restores resources with this annotation automatically, unless you set the annotation to the false value.

To make changes, disable automatic updates with the annotation, and edit the subjects in the binding.

### IMPORTANT

The oc adm policy remove-cluster-role-from-group command removes the cluster role binding when you remove the last subject.

Use extra caution or avoid this command to manage protected role bindings. The command removes the permission, but only until the API server restarts. Removing the permission permanently after deleting the binding is a lengthier process than changing the subjects.

For example, to disable self-provisioning, execute the following commands:

[user@host ~]$ **oc annotate clusterrolebinding/self-provisioners \**

**--overwrite rbac.authorization.kubernetes.io/autoupdate=false**

clusterrolebinding.rbac.authorization.k8s.io/self-provisioners annotated

[user@host ~]$ **oc patch clusterrolebinding.rbac self-provisioners \**

**-p '{"subjects": null}'**

clusterrolebinding.rbac.authorization.k8s.io/self-provisioners patched

You can also use the oc edit command to modify any value of a resource. The command launches the vi editor to apply your modifications. For example, to change the subject of the role binding from the system:authenticated:oauth group to the provisioners group, execute the followign command:

[user@host ~]$ **oc edit clusterrolebinding/self-provisioners**

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

...output omitted...

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: self-provisioner

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: **provisioners**

### REFERENCES

For more information, refer to the Configuring Project Creation section in the Projects chapter in the Red Hat OpenShift Container Platform 4.14 Building Applications documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#configuring-project-creation>

[Customizing OpenShift Project Creation](https://developers.redhat.com/blog/2020/02/05/customizing-openshift-project-creation/)

## **Guided Exercise: The Project Template and the Self-Provisioner Role**

Restrict the ability to self-provision projects to a group of users, and ensure that all users from that group have write privileges on all projects that any of them creates. Also, ensure that their new projects are constrained by a limit range that restricts memory usage.

**Outcomes**

* Limit project creation to a group of users.
* Customize project creation.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command performs the following actions:

* Ensure that the cluster API is reachable.
* Create the provisioner1 and provisioner2 users with the redhat password.

[student@workstation ~]$ **lab start selfservice-projtemplate**

**Instructions**

In this exercise, you configure the cluster so that only members of the provisioners group can create projects. Members of the provisioners group have full permissions on new projects. Users cannot create workloads that request more than 1 GiB of RAM in new projects.

1. Log in to your OpenShift cluster as the admin user with the redhatocp password.
   1. Log in to the cluster as the admin user.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

1. Allow only members of the provisioners group to create projects.
   1. Examine the provisioners group.
   2. [student@workstation ~]$ **oc describe group provisioners**
   3. Name: provisioners
   4. Created: 12 seconds ago
   5. Labels: <none>
   6. Annotations: <none>
   7. Users: provisioner1

provisioner2

The provisioners group contains the provisioner1 and provisioner2 users.

* 1. Use the oc edit command to edit the self-provisioners cluster role binding.

[student@workstation ~]$ **oc edit clusterrolebinding self-provisioners**

The oc edit command launches the vi editor to apply your modifications. Change the subject of the role binding from the system:authenticated:oauth group to the provisioners group.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

...output omitted...

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: self-provisioner

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: **provisioners**

### NOTE

The rbac.authorization.kubernetes.io/autoupdate annotation protects this cluster role binding. If the API server restarts, then Kubernetes restores this cluster role binding.

In this exercise context, you are not required to make the change permanent.

Not in this exercise, but in a real-world context, you would make the change permanent by using the following command:

[user@host ~]$ **oc annotate clusterrolebinding/self-provisioners \**

**--overwrite rbac.authorization.kubernetes.io/autoupdate=false**

1. Verify that users outside the provisioners group cannot create projects.
   1. Log in to the cluster as the developer user with the developer password.
   2. [student@workstation ~]$ **oc login -u developer -p developer**
   3. Login successful.

**You don't have any projects. Contact your system administrator to request a project.**

After the role binding is changed, the oc login command reports that you must contact your system administrator to request a project, because the developer user cannot create projects.

* 1. Verify that the developer user cannot create projects.
  2. [student@workstation ~]$ **oc new-project test**

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

1. Verify that members of the provisioners group can create projects.
   1. Log in to the cluster as the provisioner1 user with the redhat password.
   2. [student@workstation ~]$ **oc login -u provisioner1 -p redhat**
   3. Login successful.
   4. You don't have any projects. You can try to create a new project, by running

...output omitted...

* 1. Create a project by using the oc new-project command.
  2. [student@workstation ~]$ **oc new-project test**
  3. Now using project "test" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Verify that you can create resources in the test project.
  2. [student@workstation ~]$ **oc create configmap test**

configmap/test created

1. Verify that another member of the provisioners group cannot access the test project.
   1. Log in to the cluster as the provisioner2 user with the redhat password.
   2. [student@workstation ~]$ **oc login -u provisioner2 -p redhat**
   3. Login successful.
   4. You don't have any projects. You can try to create a new project, by running

oc new-project <projectname>

The oc login command reports that the provisioner2 user does not have any projects.

* 1. Try to change to the test project with the oc project command.
  2. [student@workstation ~]$ **oc project test**
  3. error: You are not a member of project "test".

You are not a member of any projects. You can request a project to be created with the 'new-project' command.

1. Log in to the cluster as the admin user with the redhatocp password, to clean up.
   1. Log in as the admin user.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp**

...output omitted...

* 1. Delete the test project.
  2. [student@workstation ~]$ **oc delete project test**

project.project.openshift.io "test" deleted

1. Create a namespace to design a project template. Add a limit range that prevents users from creating workloads that request more than 1 GiB of RAM.
   1. Use the oc create namespace command to create the template-test namespace.
   2. [student@workstation ~]$ **oc create namespace template-test**

namespace/template-test created

* 1. Edit the ~/DO280/labs/selfservice-projtemplate/limitrange.yaml file to add the limit. The file must match the following content:
  2. apiVersion: v1
  3. kind: LimitRange
  4. metadata:
  5. name: **max-memory**
  6. namespace: **template-test**
  7. spec:
  8. limits:
  9. **- max:**
  10. **memory: 1Gi**

type: Container

* 1. Use the oc create command to create the limit range that the ~/DO280/labs/selfservice-projtemplate/limitrange.yaml file defines.
  2. [student@workstation ~]$ **oc create \**
  3. **-f ~/DO280/labs/selfservice-projtemplate/limitrange.yaml**

limitrange/max-memory created

* 1. Examine the ~/DO280/labs/selfservice-projtemplate/deployment.yaml file. This file defines a deployment that requests 2 GiB of RAM.
  2. apiVersion: apps/v1
  3. kind: Deployment
  4. metadata:
  5. ...output omitted...
  6. name: test
  7. spec:
  8. ...output omitted...
  9. template:
  10. ...output omitted...
  11. spec:
  12. containers:
  13. - image: registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0
  14. name: hello-world-nginx
  15. **resources:**
  16. **limits:**

**memory: 2Gi**

* 1. Create the deployment by using the ~/DO280/labs/selfservice-projtemplate/deployment.yaml file.
  2. [student@workstation ~]$ **oc create \**
  3. **-f ~/DO280/labs/selfservice-projtemplate/deployment.yaml \**
  4. **-n template-test**

deployment.apps/test created

* 1. Examine the pods and events in the template-test namespace.
  2. [student@workstation ~]$ **oc get pod -n template-test**

No resources found in template-test namespace.

[student@workstation ~]$ **oc get event -n template-test \**

**--sort-by=metadata.creationTimestamp**

LAST SEEN TYPE REASON OBJECT MESSAGE

...output omitted...

39s Warning FailedCreate replicaset/test-846769884c Error creating: pods "test-846769884c-5zjhw" is forbidden: maximum memory usage per Container is 1Gi, but limit is 2Gi

The limit range maximum prevents the deployment from creating pods.

1. Define the project template.

The ~/DO280/solutions/selfservice-projtemplate/template.yaml file contains a solution.

* 1. Use the oc adm create-bootstrap-project-template command to print an initial project template. Redirect the output to the template.yaml file.
  2. [student@workstation ~]$ **oc adm create-bootstrap-project-template \**

**-o yaml >template.yaml**

* 1. Use the oc command to list the limit range in YAML format. Redirect the output to append to the template.yaml file.
  2. [student@workstation ~]$ **oc get limitrange -n template-test \**

**-o yaml >>template.yaml**

* 1. Edit the template.yaml file to perform the following operations:
     + Apply the following changes to the subjects key in the admin role binding:
       - Change the kind key to Group.
       - Change the name key to provisioners.
     + Move the limit range to immediately after the role binding definition.
     + Replace the namespace: template-test text with the namespace: ${PROJECT\_NAME} text.
     + Remove any left-over content after the parameters block.
     + Remove the following keys from the limit range and quota definitions:
       - creationTimestamp
       - resourceVersion
       - uid

If you use the vi editor, then you can use the following procedure to move a block of text:

* + - Move to the beginning of the block.
    - Press **V** to enter visual line mode. This mode selects entire lines for manipulation.
    - Move to the end of the block. The editor highlights the selected lines.
    - Press **d** to delete the lines and to store them in a register for later use.
    - Move to the destination.
    - Press **P** to insert the lines that are stored in the register.

You can also press **dd** to delete entire lines, and press **.** to repeat the operation.

The resulting file should match the following content:

apiVersion: template.openshift.io/v1

kind: Template

metadata:

creationTimestamp: null

name: project-request

objects:

- apiVersion: project.openshift.io/v1

kind: Project

metadata:

annotations:

openshift.io/description: ${PROJECT\_DESCRIPTION}

openshift.io/display-name: ${PROJECT\_DISPLAYNAME}

openshift.io/requester: ${PROJECT\_REQUESTING\_USER}

creationTimestamp: null

name: ${PROJECT\_NAME}

spec: {}

status: {}

- apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

creationTimestamp: null

name: admin

namespace: ${PROJECT\_NAME}

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: admin

subjects:

- apiGroup: rbac.authorization.k8s.io

**kind: Group**

**name: provisioners**

**- apiVersion: v1**

**kind: LimitRange**

**metadata:**

**name: max-memory**

**namespace: ${PROJECT\_NAME}**

**spec:**

**limits:**

**- default:**

**memory: 1Gi**

**defaultRequest:**

**memory: 1Gi**

**max:**

**memory: 1Gi**

**type: Container**

parameters:

- name: PROJECT\_NAME

- name: PROJECT\_DISPLAYNAME

- name: PROJECT\_DESCRIPTION

- name: PROJECT\_ADMIN\_USER

- name: PROJECT\_REQUESTING\_USER

### NOTE

The limit range has default and defaultRequest limits, although the definition does not contain these keys. When creating a limit range, always set the default and defaultRequest limits for more predictable behavior.

1. Create and configure the project template.
   1. Use the oc command to create the project template.
   2. [student@workstation ~]$ **oc create -f template.yaml -n openshift-config**

template.template.openshift.io/project-request created

* 1. Use the oc edit command to change the global cluster project configuration.

[student@workstation ~]$ **oc edit projects.config.openshift.io cluster**

Edit the resource to match the following content:

apiVersion: config.openshift.io/v1

kind: Project

metadata:

...output omitted...

name: cluster

...output omitted...

spec:

**projectRequestTemplate:**

**name: project-request**

* 1. Use the watch command to view the API server pods.
  2. [student@workstation ~]$ **watch oc get pod -n openshift-apiserver**
  3. NAME READY STATUS RESTARTS AGE

apiserver-6b7b... 2/2 Running 0 2m30s

Wait until new pods are rolled out. The rollout can take a few minutes to start. Press **Ctrl**+**C** to exit the watch command.

1. Create a project as the provisioner1 user.
   1. Log in to the cluster as the provisioner1 user with the redhat password.
   2. [student@workstation ~]$ **oc login -u provisioner1 -p redhat**
   3. Login successful.

...output omitted...

* 1. Create a project by using the oc new-project command.
  2. [student@workstation ~]$ **oc new-project test**
  3. Now using project "test" on server "https://api.ocp4.example.com:6443".

...output omitted...

1. Verify that the provisioner2 user can access the test project and create resources. Verify that the limit range has the intended effect.
   1. Log in to the cluster as the provisioner2 user with the redhat password.
   2. [student@workstation ~]$ **oc login -u provisioner2 -p redhat**
   3. Login successful.
   4. You have one project on this server: "test"

Using project "test".

The oc login command reports that the provisioner2 user has the test project. The command selects the project.

* 1. Create a resource on the test project.
  2. [student@workstation ~]$ **oc create configmap test**

configmap/test created

The provisioner2 user can create resources in a project that the provisioner1 user created.

* 1. Create a deployment that exceeds the limit range by using the ~/DO280/labs/selfservice-projtemplate/deployment.yaml file.
  2. [student@workstation ~]$ **oc create \**
  3. **-f ~/DO280/labs/selfservice-projtemplate/deployment.yaml**

deployment.apps/test created

* 1. Examine the pods and events in the template-test namespace.
  2. [student@workstation ~]$ **oc get pod**

No resources found in test namespace.

[student@workstation ~]$ **oc get event --sort-by=metadata.creationTimestamp**

LAST SEEN TYPE REASON OBJECT MESSAGE

...output omitted...

39s Warning FailedCreate replicaset/test-846769884c Error creating: pods "test-846769884c-5zjhw" is forbidden: maximum memory usage per Container is 1Gi, but limit is 2Gi

The limit range works as expected.

**Finish**

[student@workstation ~]$ **lab finish selfservice-projtemplate**

# Chapter 7.  Manage Kubernetes Operators

[Kubernetes Operators and the Operator Lifecycle Manager](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07)

[Quiz: Kubernetes Operators and the Operator Lifecycle Manager](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s02)

[Install Operators with the Web Console](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s03)

[Guided Exercise: Install Operators with the Web Console](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s04)

[Install Operators with the CLI](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s05)

[Guided Exercise: Install Operators with the CLI](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s06)

[Lab: Manage Kubernetes Operators](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s07)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch07s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Install and update operators that the Operator Lifecycle Manager and the Cluster Version Operator manage. |
| **Objectives** | * Explain the operator pattern and different approaches for installing and updating Kubernetes operators. * Install and update operators by using the web console and the Operator Lifecycle Manager. * Install and update operators by using the Operator Lifecycle Manager APIs. |
| **Sections** | * Kubernetes Operators and the Operator Lifecycle Manager (and Matching Quiz) * Install Operators with the Web Console (and Guided Exercise) * Install Operators with the CLI (and Guided Exercise) |
| **Lab** | * Manage Kubernetes Operators |

## **Kubernetes Operators and the Operator Lifecycle Manager**

### **Objectives**

* Describe the operator pattern and different approaches for installing and updating Kubernetes operators.

### **The Operator Pattern**

You can deploy workloads to Kubernetes with resources such as deployments, replica sets, stateful sets, daemon sets, jobs, and cron jobs. All of these resources create a workload that runs software that is packaged as a container image, in different modalities. For example, jobs execute a one-off task; cron jobs execute tasks periodically; and the other resources create persistent workloads. Resources such as deployments, stateful sets, or daemon sets differ on how the workload is distributed in a cluster.

These resources are sufficient to deploy many workloads. However, more complex workloads might require significant work to deploy with only these resources. For example, a workload can involve different component workloads, such as a database server, a back-end service, and a front-end service.

A workload might have maintenance tasks that can be automated, such as backing up data or updating the workload.

The operator pattern is a way to implement reusable software to manage such complex workloads.

An operator typically defines custom resources (CRs). The operator CRs contain the needed information to deploy and manage the workload. For example, an operator that deploys database servers defines a database resource where you can specify the database name, sizing requirements, and other parameters.

The operator watches the cluster for instances of the CRs, and then creates the Kubernetes resources to deploy the custom workload. For example, when you create a database resource, the database operator creates a stateful set and a persistent volume that provide the database that is described in the database resource. If the database resource describes a backup schedule and target, then the operator creates a cron job that backs up the database to the target according to the schedule.

By using operators, cluster administrators create CRs that describe a complex workload, and the operator creates and manages the workload.

### **Deploying Operators**

Many pieces of software implement the operator pattern, in different ways.

**Cluster operators**

Cluster operators provide the platform services of OpenShift, such as the web console and the OAuth server.

**Add-on operators**

OpenShift includes the Operator Lifecycle Manager (OLM). The OLM helps users to install and update operators in a cluster. Operators that the OLM manages are also known as add-on operators, in contrast with cluster operators that implement platform services.

**Other operators**

Software providers can create software that follows the operator pattern, and then distribute the software as manifests, Helm charts, or any other software distribution mechanism.

#### **Cluster Operators**

The Cluster Version Operator (CVO) installs and updates cluster operators as part of the OpenShift installation and update processes.

The CVO provides cluster operator status information as resources of the ClusterOperator type. Inspect the cluster operator resources to examine cluster health.

[user@host ~]$ **oc get clusteroperator**

NAME VERSION AVAILABLE PROGRESSING DEGRADED ... MESSAGE

authentication 4.14.0 True False False ...

baremetal 4.14.0 True False False ...

cloud-controller-manager 4.14.0 True False False ...

...output omitted...

The status of cluster operator resources includes conditions to help with identifying cluster issues. The oc command shows the message that is associated with the latest condition. This message can provide further information about cluster issues.

To view cluster operator resources in the web console, navigate to **Administration** → **Cluster Settings**, and then click the **ClusterOperators** tab.

#### **The Operator Lifecycle Manager and the OperatorHub**

Administrators can use the OLM to install, update, and remove operators.

You can use the web console to interact with the OLM. The OLM also follows the operator pattern, and so the OLM provides CRs to manage operators with the Kubernetes API.

The OLM uses operator catalogs to find available operators to install. Operator catalogs are container images that provide information about available operators, such as descriptions and available versions.

OpenShift includes several default catalogs:

**Red Hat**

Red Hat packages, ships, and supports operators in this catalog.

**Certified**

Independent software vendors support operators in this catalog.

**Community**

Operators without official support.

**Marketplace**

Commercial operators that you can buy from Red Hat Marketplace.

You can also create your own catalogs, or mirror catalogs for offline clusters.

### NOTE

The lab environment includes a single catalog with the operators you use in the course. The lab environment hosts the contents of this catalog, so that the course can be completed without internet access.

The OLM creates a resource of the PackageManifest type for each available operator. The web console also displays available operators and provides a wizard to install operators. You can also install operators by using the Subscription CR and other CRs.

### NOTE

Operators that are installed with the OLM have a different lifecycle from cluster operators. The CVO installs and updates cluster operators in lockstep with the cluster. Administrators use the OLM to install, update, and remove operators independently from cluster updates.

Some operators might require additional steps to install, update, or remove.

### **Implementing Operators**

An operator is composed of a set of custom resource definitions and a Kubernetes workload. The operator workload uses the Kubernetes API to watch instances of the CRs and to create matching workloads.

### NOTE

A cluster contains two workload sets for each operator.

* The operator workload, which the OLM manages
* The workloads that are associated with the custom resources, and which the operator manages

You can implement operators to automate any manual Kubernetes task that fits the operator pattern. You can use most software development platforms to create operators. The following SDKs provide components and frameworks to help with developing operators:

**The Operator SDK**

The Operator SDK contains tools to develop operators with the Go programming language, and Ansible. The Operator SDK also contains tools to package Helm charts as operators.

**The Java Operator SDK**

The Java Operator SDK contains tools to develop operators with the Java programming language. The Java Operator SDK has a Quarkus extension to develop operators with the Quarkus framework.

### REFERENCES

For more information, refer to the Operators guide in the Red Hat OpenShift Container Platform 4.14 documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index>

[Operator SDK](https://sdk.operatorframework.io/)

[Java Operator SDK](https://javaoperatorsdk.io/)

[Quarkus Operator SDK](https://github.com/quarkiverse/quarkus-operator-sdk)

## **Install Operators with the Web Console**

### **Objectives**

* Install and update operators by using the web console and the Operator Lifecycle Manager.

### **Installing Operators with the Web Console**

The OpenShift web console provides a graphical interface to the Operator Lifecycle Manager (OLM). The **OperatorHub** page lists available operators and provides an interface for installing them. The **Installed Operators** page lists installed operators. You can inspect and uninstall operators from the **Installed Operators** page.

### **The Install Operator Wizard**

Navigate to **Operators** → **OperatorHub** to display the list of available operators. The **OperatorHub** page displays operators, and has filters to locate operators by category, source, provider, subscription required, and other criteria.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 7.1: Available operators

Click an operator to display further information.

### IMPORTANT

Before installing an operator, review the operator information and consult the operator documentation. You might need to configure the operator further for successful deployment.

Click **Install** to begin the **Install Operator** wizard.

You can choose installation options in the **Install Operator** wizard.

**Update channel**

You can choose the most suitable operator update channel for your requirements. For more information, refer to [the section called “Operator Update Channels”](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s05).

**Installation mode**

The default **All namespaces on the cluster (default)** installation mode should be suitable for most operators. This mode configures the operator to monitor all namespaces for custom resources.

For example, an operator that deploys database servers defines a custom resource that describes a database server. When using the **All namespaces on the cluster (default)** installation mode, users can create those custom resources in their namespaces. Then, the operator deploys database servers in the same namespaces, along with other user workloads.

Cluster administrators can combine this mode with self-service features and other namespace-based features, such as role-based access control and network policies, to control user usage of operators.

**Installed namespace**

The OLM installs the operator workload to the selected namespace in this option. Some operators install by default to the openshift-operators namespace. Other operators suggest creating a namespace.

Although users might require access to the workloads that the operator manages, typically only cluster administrators require access to the operator workload.

**Update approval**

The OLM updates operators automatically when new versions are available. Choose manual updates to prevent automatic updates.

For an operator that includes monitoring in its definition, the wizard displays a further option to enable the monitoring. Adding monitoring from non-Red Hat operators is not supported.

The installation mode and installed namespace options are related. Review the documentation of the operator to learn the supported options.

After you configure the installation, click **Install**. The web console creates subscription and operator group resources according to the selected options in the wizard. After the installation starts, the web console displays progress information.

### **Viewing Installed Operators**

When the OLM finishes installing an operator, click **View Operator** to display the **Operator details** page. You can also view information about installed operators by navigating to **Operators** → **Installed Operators**.

The **Installed Operators** page lists the installed cluster service version (CSV) resources that correspond to installed operators.

Every version of an operator has a CSV. The OLM uses information from the CSV to install the operator. The OLM updates the status key of the CSV with installation information.

CSVs are namespaced, so the **Installed Operator** page has a similar namespace filter to other web console pages. Operators that were installed with the "all namespaces" mode have a CSV in all namespaces.

### NOTE

The operator installation mode determines which namespaces the operator monitors for custom resources. This mode is a distinct option from the installed namespace option, which determines the operator workload namespace.

The **Installed Operators** page shows information such as the operator status and available updates. Click an operator to navigate to the **Operator details** page.

The **Operator details** page contains the following tabs, where you can view further details and perform other actions.

**Details**

Displays information about the CSV.

**YAML**

Displays the CSV in YAML format.

**Subscription**

In this tab, you can change installation options, such as the update channel and update approval. This tab also links to the install plans of the operator. When you configure an operator for manual updates, you approve install plans for updates in this tab.

**Events**

Lists events that are related to the operator.

The **Operator details** page also has tabs for custom resources. For each custom resource that the operator defines, a web console tab lists all resources of that type. Additionally, the **All instances** tab aggregates all resources of types that the operator defines.

### **Using Operators**

Custom resources are the most common way to interact with operators. You can create custom resources by using the custom resource tabs on the **Installed Operators** page. Select the tab to correspond to the custom resource type to create, and then click the create button.

Custom resources use the same creation page as other Kubernetes resources. You can choose either the YAML view or the form view to configure the new resource.

In the YAML view, you use the YAML editor to compose the custom resource. The editor provides a starting template that you can customize. The YAML view also displays documentation about the custom resource schema. The oc explain command provides the same documentation.

The form view presents a set of fields in a resource. Instead of composing a full YAML definition, you can edit the fields individually. When complete, OpenShift creates a resource from the values in the form.

Fields might provide help text and further configuration help. For example, fields with a limited set of values might provide a drop-down list with the possible values. The form view might provide more guidance, but might not contain fields to customize all possible options of a custom resource.

### **Troubleshooting Operators**

The OLM might fail to install or update operators, or operators might not work correctly.

To identify operator installation issues, examine the status and conditions of the CSV, subscription, and install plan resources.

### NOTE

Installation issues can be operator-specific, so consult the documentation of malfunctioning operators to determine support options.

To troubleshoot further issues that cause operators to work incorrectly, first identify the operator workload. The **Operator Deployments** field in the **Operator details** page shows operator workloads. Operators might create further workloads, including workloads that follow the definitions that you provide in custom resources.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 7.2: Operator deployments

Identify and troubleshoot the operator workload as with any other Kubernetes workload. The following resources are common starting points when troubleshooting:

* The status of Kubernetes workload resources, such as deployments or stateful sets
* Pod logs and their status
* Events

### REFERENCES

For more information, refer to the Installing from OperatorHub Using the Web Console section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-installing-from-operatorhub-using-web-console_olm-adding-operators-to-a-cluster>

For more information about monitoring configuration, refer to the Maintenance and Support for Monitoring section in the Configuring the Monitoring Stack chapter in the Red Hat OpenShift Container Platform 4.14 Monitoring documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/monitoring/index#maintenance-and-support_configuring-the-monitoring-stack>

## **Install Operators with the CLI**

### **Objectives**

* Install and update operators by using the Operator Lifecycle Manager APIs.

### **Installing Operators**

To install an operator, you must perform the following steps:

* Locate the operator to install.
* Review the operator and its documentation for installation options and requirements.
  + Decide the update channel to use.
  + Decide the installation mode. For most operators, you should make them available to all namespaces.
  + Decide to deploy the operator workload to an existing namespace or to a new namespace.
  + Decide whether the Operator Lifecycle Manager (OLM) applies updates automatically, or requires an administrator to approve updates.
* Create an operator group if needed for the installation mode.
* Create a namespace for the operator workload if needed.
* Create the operator subscription.
* Review and test the operator installation.

### **Operator Resources**

The OLM uses the following resource types:

**Catalog source**

Each catalog source resource references an operator repository. Periodically, the OLM examines the catalog sources in the cluster and retrieves information about the operators in each source.

**Package manifest**

The OLM creates a package manifest for each available operator. The package manifest contains the required information to install an operator, such as the available channels.

**Operator group**

Operator groups define how the OLM presents operators across namespaces.

**Subscription**

Cluster administrators create subscriptions to install operators.

**Operator**

The OLM creates operator resources to store information about installed operators.

**Install plan**

The OLM creates install plan resources as part of the installation and update process. When requiring approvals, administrators must approve install plans.

**Cluster service version (CSV)**

Each version of an operator has a corresponding CSV. The CSV contains the information that the OLM requires to install the operator.

When installing an operator, an administrator must create only the subscription and the operator group. The OLM generates all other resources automatically.

### **Examining Available Operators**

Examine catalog sources in the openshift-marketplace namespace to know which catalog sources are available in a cluster.

[user@host ~]$ **oc get catalogsource -n openshift-marketplace**

NAME DISPLAY TYPE ... AGE

do280-catalog-cs do280 Operator Catalog Cs grpc ... 7d6h

The OLM creates a package manifest for each available operator that a catalog source references. List the package manifests to know which operators are available for installation.

[user@host ~]$ **oc get packagemanifests**

NAME CATALOG AGE

lvms-operator do280 Operator Catalog Cs 7d6h

kubevirt-hyperconverged do280 Operator Catalog Cs 7d6h

file-integrity-operator do280 Operator Catalog Cs 7d6h

compliance-operator do280 Operator Catalog Cs 7d6h

metallb-operator do280 Operator Catalog Cs 7d6h

To gather the required information to install an operator, view the details of a specific package manifest. Use the oc describe command on a package manifest to view details about an operator.

[user@host ~]$ **oc describe packagemanifest lvms-operator -n openshift-marketplace**

Name: lvms-operator

...output omitted...

Spec:

Status:

Catalog Source: do280-catalog-cs

Catalog Source Display Name: do280 Operator Catalog Cs

Catalog Source Namespace: openshift-marketplace

Catalog Source Publisher:

Channels:

Current CSV: lvms-operator.v4.14.1

Current CSV Desc:

Annotations:

...output omitted...

Capabilities: Seamless Upgrades

Categories: Storage

Container Image: registry.redhat.io/lvms4/lvms-rhel9-operator@sha256:545a...67e9

Description: Logical volume manager storage provides dynamically provisioned local storage for container workloads

...output omitted...

operatorframework.io/suggested-namespace: openshift-storage

operators.openshift.io/infrastructure-features: ["csi", "disconnected"]

operators.openshift.io/valid-subscription: ["OpenShift Container Platform", "OpenShift Platform Plus"]

operators.operatorframework.io/builder: operator-sdk-v1.23.0

...output omitted...

Apiservicedefinitions:

Customresourcedefinitions:

Owned:

Kind: LogicalVolume

Name: logicalvolumes.topolvm.io

Version: v1

Description: LVMCluster is the Schema for the lvmclusters API

Display Name: LVMCluster

Kind: LVMCluster

Name: lvmclusters.lvm.topolvm.io

Version: v1alpha1

Kind: LVMVolumeGroupNodeStatus

Name: lvmvolumegroupnodestatuses.lvm.topolvm.io

Version: v1alpha1

Kind: LVMVolumeGroup

Name: lvmvolumegroups.lvm.topolvm.io

Version: v1alpha1

Description: Logical volume manager storage provides dynamically provisioned local storage.

Display Name: LVM Storage

Install Modes:

Supported: true

Type: OwnNamespace

Supported: true

Type: SingleNamespace

Supported: false

Type: MultiNamespace

Supported: false

Type: AllNamespaces

...output omitted...

Links:

Name: Source Repository

URL: https://github.com/openshift/lvm-operator

...output omitted...

Maturity: alpha

Provider:

Name: Red Hat

...output omitted...

Version: 4.14.1

Name: stable-4.14

Default Channel: stable-4.14

Package Name: lvms-operator

Provider:

Name: Red Hat

Events: <none>

|  |  |
| --- | --- |
|  | The catalog source and namespace for the operator, which are required to identify the operator when creating the subscription. |
|  | Examine the available channels and CSVs to decide which upgrade path to use. |
|  | The description and links provide useful information and documentation for installation and uninstallation procedures. |
|  | The install modes provide information about supported namespace operation modes. |

### **Installing Operators**

After you examine the package manifest, review the operator documentation. Operators might require specific installation procedures.

If you decide to deploy the operator workload to a new namespace, then create the namespace. Many operators recommend to use the existing openshift-operators namespace, or require specific namespaces.

Determine whether you need to create an operator group. Operators use the operator group in their namespace. Operators monitor custom resources in the namespaces that the operator group targets.

The openshift-operators namespace contains a global-operators operator group. Operators that are installed in the openshift-operators namespace use this operator group and monitor all namespaces.

If the global-operators operator group is not suitable, then create another operator group. The following YAML definition describes the structure of an operator group:

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: ***name***

namespace: ***namespace***

spec:

targetNamespaces:

- ***namespace***

|  |  |
| --- | --- |
|  | Operators follow the operator group in the namespace that they are deployed in. |
|  | List the namespaces that the operator monitors for custom resources. You can also use the spec.selector field to select namespaces by using labels. |

After creating the necessary namespaces or operator groups, you create a subscription. The following YAML file is an example of a subscription:

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: lvms-operator

namespace: openshift-storage

spec:

channel: stable-4.14

name: lvms-operator

source: do280-catalog-cs

installPlanApproval: Automatic

sourceNamespace: openshift-marketplace

|  |  |
| --- | --- |
|  | The namespace for the operator workload |
|  | The update channel, from the discovered information from the oc describe packagemanifest command |
|  | The package manifest to subscribe to |
|  | The source catalog, from the discovered information from the oc describe packagemanifest command |
|  | The install plan approval mode, either Automatic or Manual |

#### **Install Plans**

The OLM creates an install plan resource to represent the required process to install or update an operator. The OLM updates the operator resource to reference the install plan in the status.components.refs field. You can view the reference by using the oc describe command on the operator resource.

[user@host ~]$ **oc describe operator file-integrity-operator**

Name: file-integrity-operator.openshift-file-integrity

Namespace:

Labels: <none>

Annotations: <none>

API Version: operators.coreos.com/v1

Kind: Operator

...output omitted...

Status:

Components:

...output omitted...

Refs:

API Version: operators.coreos.com/v1alpha1

Kind: InstallPlan

Name: **install-pmh78**

Namespace: **openshift-file-integrity**

API Version: operators.coreos.com/v1alpha1

Conditions:

Last Transition Time: 2024-01-26T17:53:27Z

Message: all available catalogsources are healthy

Reason: AllCatalogSourcesHealthy

Status: False

Type: CatalogSourcesUnhealthy

Last Transition Time: 2024-01-26T17:53:49Z

Reason: **RequiresApproval**

Status: True

Type: InstallPlanPending

Kind: Subscription

Name: file-integrity-operator

Namespace: openshift-file-integrity

Events: <none>

If the install plan mode is set to Manual in the subscription, then you must manually approve the install plan. To approve an install plan, change the spec.approved field to true. For example, you can use the oc patch command to approve an install plan:

[user@host ~]$ **oc patch installplan *install-pmh78* --type merge -p \**

**'{"spec":{"approved":true}}' -n openshift-file-integrity**

installplan.operators.coreos.com/install-pmh78 patched

With an Automatic install plan mode, the OLM applies updates as soon as they are available.

### **Using Operators**

Typically, operators create custom resource definitions. You create instances of those custom resources to use the operator. Review the operator documentation to learn how to use an operator.

Additionally, you can learn about the available custom resource definitions by examining the operator. The CSV contains a list of the custom resource definitions in the spec.customresourcedefinitions field. For example, use the following command to list the custom resource definitions:

[user@host ~]$ **oc get csv metallb-operator.v4.14.0-202401151553 \**

**-o jsonpath="{.spec.customresourcedefinitions.owned[\*].name}{'\n'}"**

addresspools.metallb.io addresspools.metallb.io bfdprofiles.metallb.io bgpadvertisements.metallb.io bgppeers.metallb.io bgppeers.metallb.io communities.metallb.io ipaddresspools.metallb.io l2advertisements.metallb.io metallbs.metallb.io

You can also use the oc explain command to view the description of individual custom resource definitions.

### **Troubleshooting Operators**

Some operators require additional steps to install or update. Review the documentation to validate whether you performed all necessary steps, and to learn about support options.

You can examine the status of the operator, install plan, and CSV resources. When installing or updating operators, the OLM updates those resources with progress information.

Even if the OLM installs an operator correctly, the operator might not function correctly.

Operators typically contain two kinds of workloads:

* The operator workload, which monitors custom resources.
* The workloads that individual instances of the custom resources created.

The spec.install.spec.deployments in the CSV contains the deployments that the OLM creates when installing an operator. These deployments often correspond to the operator workload. However, the operator might create further deployments either for its own workload, or for the workloads that are associated with custom resources.

### REFERENCES

For more information about operators, refer to the Operators Overview chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#operators-overview>

For more information about installing operators, refer to the Installing from OperatorHub Using the CLI section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-installing-operator-from-operatorhub-using-cli_olm-adding-operators-to-a-cluster>

For more information about operator groups, refer to the Operator Groups section in the Understanding Operators chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-operatorgroups-about_olm-understanding-olm>

## **Guided Exercise: Install Operators with the CLI**

Install an operator by using the command-line interface and Kubernetes manifests.

**Outcomes**

* Install operators from the CLI with manual updates.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster is ready, and removes the openshift-file-integrity namespace and File Integrity operator if they exist.

[student@workstation ~]$ **lab start operators-cli**

**Instructions**

In this exercise, you install the File Integrity operator with manual updates. The documentation of the File Integrity operator contains specific installation instructions.

For more information, refer to the Installing the File Integrity Operator Using the CLI section in the File Integrity Operator chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance/index#installing-file-integrity-operator-using-cli_file-integrity-operator-installation>

1. Log in to the OpenShift cluster as the admin user with the redhatocp password.
2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
3. **https://api.ocp4.example.com:6443**
4. Login successful.

...output omitted...

1. Find the details of the File Integrity operator within the OpenShift package manifests.
   1. View the available operators within the OpenShift Marketplace by using the oc get command.
   2. [student@workstation ~]$ **oc get packagemanifests**
   3. NAME CATALOG AGE
   4. **file-integrity-operator do280 Operator Catalog Cs 37h**
   5. lvms-operator do280 Operator Catalog Cs 37h
   6. compliance-operator do280 Operator Catalog Cs 37h
   7. metallb-operator do280 Operator Catalog Cs 37h

kubevirt-hyperconverged do280 Operator Catalog Cs 37h

* 1. Examine the File Integrity operator package manifest by using the oc describe command.
  2. [student@workstation ~]$ **oc describe packagemanifest file-integrity-operator**
  3. Name: **file-integrity-operator**
  4. ...output omitted...
  5. Spec:
  6. Status:
  7. Catalog Source: **do280-catalog-cs**
  8. Catalog Source Display Name: do280 Operator Catalog Cs
  9. Catalog Source Namespace: **openshift-marketplace**
  10. Catalog Source Publisher:
  11. Channels:
  12. ...output omitted...
  13. **Install Modes:**
  14. **Supported: true**
  15. **Type: OwnNamespace**
  16. **Supported: true**
  17. **Type: SingleNamespace**
  18. **Supported: false**
  19. **Type: MultiNamespace**
  20. **Supported: true**
  21. **Type: AllNamespaces**
  22. ...output omitted...
  23. Name: **stable**
  24. Default Channel: **stable**
  25. Package Name: **file-integrity-operator**

...output omitted...

The operator is in the do280-catalog-cs catalog source in the openshift-marketplace namespace. The operator has a single channel with the v1 name. The operator has the file-integrity-operator name.

1. Install the File Integrity operator. By following the operator installation instructions, you must install the operator in the openshift-file-integrity namespace. Also, you must make the operator available only in that namespace. The File Integrity operator requires you to create a namespace with specific labels.
   1. The operator documentation provides a YAML definition of the required namespace. The definition is available in the ~/DO280/labs/operators-cli/namespace.yaml path. Examine the definition and create the namespace.
   2. apiVersion: v1
   3. kind: Namespace
   4. metadata:
   5. labels:
   6. openshift.io/cluster-monitoring: "true"
   7. pod-security.kubernetes.io/enforce: privileged

name: openshift-file-integrity

[student@workstation ~]$ **oc create -f ~/DO280/labs/operators-cli/namespace.yaml**

namespace/openshift-file-integrity created

* 1. Create an operator group in the operator namespace. The operator group targets the same namespace. You can use the template in the ~/DO280/labs/operators-cli/operator-group.yaml path. Edit the file and configure the namespaces.
  2. apiVersion: operators.coreos.com/v1
  3. kind: OperatorGroup
  4. metadata:
  5. name: file-integrity-operator
  6. namespace: **openshift-file-integrity**
  7. spec:
  8. targetNamespaces:

- **openshift-file-integrity**

Create the operator group.

[student@workstation ~]$ **oc create \**

**-f ~/DO280/labs/operators-cli/operator-group.yaml**

operatorgroup.operators.coreos.com/file-integrity-operator created

* 1. Create the subscription in the operator namespace. You can use the template in the ~/DO280/labs/operators-cli/subscription.yaml path. Edit the file with the data that you obtained in a previous step. Set the approval policy to Manual.
  2. apiVersion: operators.coreos.com/v1alpha1
  3. kind: Subscription
  4. metadata:
  5. name: file-integrity-operator
  6. namespace: **openshift-file-integrity**
  7. spec:
  8. channel: **"stable"**
  9. installPlanApproval: **Manual**
  10. name: **file-integrity-operator**
  11. source: **do280-catalog-cs**

sourceNamespace: openshift-marketplace

Create the subscription.

[student@workstation ~]$ **oc create -f ~/DO280/labs/operators-cli/subscription.yaml**

subscription.operators.coreos.com/file-integrity-operator created

1. Approve the install plan.
   1. Examine the operator resource that the OLM created.
   2. [student@workstation ~]$ **oc describe operator file-integrity-operator**
   3. Name: file-integrity-operator.openshift-file-integrity
   4. ...output omitted...
   5. Status:
   6. Components:
   7. Label Selector:
   8. Match Expressions:
   9. Key: operators.coreos.com/file-integrity-operator.openshift-file-integrity
   10. Operator: Exists
   11. Refs:
   12. ...output omitted...
   13. Kind: InstallPlan
   14. Name: **install-4wsq6**
   15. Namespace: **openshift-file-integrity**
   16. API Version: operators.coreos.com/v1alpha1
   17. Conditions:
   18. Last Transition Time: 2024-01-26T10:38:22Z
   19. Message: all available catalogsources are healthy
   20. Reason: AllCatalogSourcesHealthy
   21. Status: False
   22. Type: CatalogSourcesUnhealthy
   23. Last Transition Time: 2024-01-26T10:38:21Z
   24. Reason: RequiresApproval
   25. Status: True
   26. **Type: InstallPlanPending**
   27. Kind: Subscription
   28. Name: file-integrity-operator
   29. Namespace: openshift-file-integrity

Events: <none>

Verify that the operator has a condition of the InstallPlanPending type. The operator can have other conditions, and they do not indicate a problem. The operator references the install plan. You use the install plan name in a later step. If the install plan is not generated, then wait a few moments and run the oc describe command again.

* 1. View the install plan specification with the oc get command. Replace the name with the install plan name that you obtained in a previous step.
  2. [student@workstation ~]$ **oc get installplan -n openshift-file-integrity \**
  3. ***install-4wsq6* -o jsonpath='{.spec}{"\n"}'**

{"approval":"Manual","approved":false,"clusterServiceVersionNames":["file-integrity-operator.v1.3.3","file-integrity-operator.v1.3.3"],"generation":1}

The install plan is set to manual approval, and the approved field is set to false.

* 1. Approve the install plan with the oc patch command. Replace the name with the install plan name that you obtained in a previous step.
  2. [student@workstation ~]$ **oc patch installplan *install-4wsq6* --type merge -p \**
  3. **'{"spec":{"approved":true}}' -n openshift-file-integrity**

installplan.operators.coreos.com/install-4wsq6 patched

* 1. Verify that the operator installs successfully, by using the oc describe command. Check the latest transaction for the current status. The installation might not complete immediately. If the installation is not complete, then wait a few minutes and view the status again.
  2. [student@workstation ~]$ **oc describe operator file-integrity-operator**
  3. ...output omitted...
  4. Status:
  5. Components:
  6. Label Selector:
  7. Match Expressions:
  8. Key: operators.coreos.com/file-integrity-operator.openshift-file-integrity
  9. Operator: Exists
  10. Refs:
  11. ...output omitted...
  12. Conditions:
  13. Last Transition Time: **2024-01-26T18:21:03Z**
  14. Last Update Time: 2024-01-26T18:21:03Z
  15. Message: **install strategy completed with no errors**
  16. Reason: InstallSucceeded
  17. Status: True
  18. Type: Succeeded
  19. Kind: ClusterServiceVersion
  20. Name: file-integrity-operator.v1.0.0
  21. Namespace: openshift-file-integrity

...output omitted...

* 1. Examine the workloads in the openshift-file-integrity namespace.
  2. [student@workstation ~]$ **oc get all -n openshift-file-integrity**
  3. Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
  4. NAME READY STATUS RESTARTS AGE
  5. pod/file-integrity-operator-6985588576-x2k49 1/1 Running 1 (50s ago) 56s
  6. ...output omitted...
  7. NAME READY UP-TO-DATE AVAILABLE AGE
  8. deployment.apps/file-integrity-operator 1/1 1 1 56s
  9. NAME DESIRED CURRENT READY AGE

replicaset.apps/file-integrity-operator-6985588576 1 1 1 56s

The namespace has a ready deployment.

1. Test the operator to ensure that it is functional. The operator watches FileIntegrity resources, runs file integrity checks on nodes, and creates FileIntegrityNodeStatus with the results of the checks.
   1. Create a FileIntegrity custom resource by applying the file at ~/DO280/labs/operators-cli/worker-fileintegrity.yaml with the oc apply command.
   2. [student@workstation ~]$ **oc apply -f \**
   3. **~/DO280/labs/operators-cli/worker-fileintegrity.yaml**

fileintegrity.fileintegrity.openshift.io/worker-fileintegrity created

* 1. Verify that the operator functions, by viewing the worker-fileintegrity object with the oc describe command.
  2. [student@workstation ~]$ **oc describe fileintegrity worker-fileintegrity \**
  3. **-n openshift-file-integrity**
  4. Name: worker-fileintegrity
  5. Namespace: openshift-file-integrity
  6. Labels: <none>
  7. Annotations: <none>
  8. API Version: fileintegrity.openshift.io/v1alpha1
  9. Kind: FileIntegrity
  10. ...output omitted...
  11. Spec:
  12. Config:
  13. Grace Period: 900
  14. Max Backups: 5
  15. Node Selector:
  16. node-role.kubernetes.io/worker:
  17. Tolerations:
  18. Effect: NoSchedule
  19. Key: node-role.kubernetes.io/master
  20. Operator: Exists
  21. Effect: NoSchedule
  22. Key: node-role.kubernetes.io/infra
  23. Operator: Exists

Events: <none>

* 1. Use oc edit to edit the Grace Period to 60 in the FileIntegrity custom resource to trigger a failure.
  2. [student@workstation ~]$ **oc edit fileintegrity worker-fileintegrity \**
  3. **-n openshift-file-integrity**
  4. Name: worker-fileintegrity
  5. Namespace: openshift-file-integrity
  6. Labels: <none>
  7. Annotations: <none>
  8. API Version: fileintegrity.openshift.io/v1alpha1
  9. Kind: FileIntegrity
  10. ...output omitted...
  11. Spec:
  12. Config:
  13. **Grace Period: 60**
  14. Max Backups: 5
  15. Node Selector:
  16. node-role.kubernetes.io/worker:
  17. Tolerations:
  18. Effect: NoSchedule
  19. Key: node-role.kubernetes.io/master
  20. Operator: Exists
  21. Effect: NoSchedule
  22. Key: node-role.kubernetes.io/infra
  23. Operator: Exists

Events: <none>

* 1. Verify that the operator automatically creates a FileIntegrityNodeStatus object, by using the oc get command. You might need to wait a few minutes for the object to generate.

### NOTE

The first file integrity resource that you create might not work correctly.

If the operator does not create the FileIntegrityNodeStatus resource in a few minutes, then delete the FileIntegrity resource and create it again.

The exercise outcome does not depend on obtaining a FileIntegrityNodeStatus resource.

[student@workstation ~]$ **oc get fileintegritynodestatuses \**

**-n openshift-file-integrity**

NAME NODE STATUS

worker-fileintegrity-master01 master01 Succeeded

* 1. After FileIntegrityNodeStatus has successfully been created, run this as the admin user to modify the node's filesystem: oc debug node/master01 — touch /host/etc/foobar
  2. [student@workstation ~]$ **oc debug node/master01 -- touch /host/etc/foobar**
  3. Starting pod/master01-debug-l92pd ...
  4. To use host binaries, `run chroot /host`

Removing debug pod ...

* 1. Run oc get configmaps -n openshift-file-integrity to list configmaps in the openshift-file-integrity namespace.
  2. [student@workstation ~]$ **oc get configmaps -n openshift-file-integrity --watch**
  3. NAME DATA AGE
  4. aide-pause 1 109m
  5. aide-reinit 1 109m
  6. **aide-worker-fileintegrity-master01-failed 1 108m**
  7. kube-root-ca.crt 1 117m
  8. openshift-service-ca.crt 1 117m

worker-fileintegrity 1 109m

### NOTE

It may take several minutes for aide-worker-fileintegrity-master01-failed to show. Use the --watch flag and wait a few minutes until the failed configmap shows to move on to the next step. Press **Ctrl**+**C** to exit.

* 1. Run oc describe to view the report in aide-worker-fileintegrity-master01-failed configmap in the openshift-file-integrity namespace.
  2. [student@workstation ~]$ **oc describe \**
  3. **configmap/aide-worker-fileintegrity-master01-failed \**
  4. **-n openshift-file-integrity**
  5. Name: aide-worker-fileintegrity-master01-failed
  6. Namespace: openshift-file-integrity
  7. Labels: file-integrity.openshift.io/node=master01
  8. file-integrity.openshift.io/owner=worker-fileintegrity
  9. file-integrity.openshift.io/result-log=
  10. Annotations: file-integrity.openshift.io/files-added: 1
  11. file-integrity.openshift.io/files-changed: 0
  12. file-integrity.openshift.io/files-removed: 0
  13. Data
  14. \====
  15. integritylog:
  16. \----
  17. Start timestamp: 2024-01-26 18:31:16 +0000 (AIDE 0.16)
  18. AIDE found differences between database and filesystem!!
  19. Summary:
  20. Total number of entries: 32359
  21. Added entries: 1
  22. Removed entries: 0
  23. Changed entries: 0
  24. ---------------------------------------------------
  25. Added entries:
  26. ---------------------------------------------------
  27. f++++++++++++++++: /hostroot/etc/cni/multus/certs/multus-client-2024-01-26-15-14-01.pem
  28. f++++++++++++++++: /hostroot/etc/foobar
  29. ---------------------------------------------------
  30. The attributes of the (uncompressed) database(s):
  31. ---------------------------------------------------
  32. /hostroot/etc/kubernetes/aide.db.gz
  33. MD5 : UswXQiVa/VpjlXF1rCP0vA==
  34. SHA1 : s6t06MCRrDgc4xOWnX6vk5rflGU=
  35. RMD160 : jvDdvAOC7/tI0TjDe7Kzmy5nUk8=
  36. TIGER : TjW192YTQBmG4oGza7siI6CBRnztgrp6
  37. SHA256 : E8rWurdI9HgGP6402qWY+lDAaLoGiyNs
  38. PEka/siI1F0=
  39. SHA512 : JPDhgoEnNiTaDLqawkGtHplRW8f6zm3g
  40. jDB3E6X6XM4+13yhjwh/pokFAp5BhRSc
  41. 0C4XXibXsS4OYxYiE5hBaw==
  42. End timestamp: 2024-01-26 18:31:45 +0000 (run time: 0m 29s)
  43. BinaryData
  44. \====

Events: <none>

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish operators-cli**

## **Lab: Manage Kubernetes Operators**

Install an operator and verify that it is healthy.

**Outcomes**

* Install the Compliance operator on the command line.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and that the operator that is used in this exercise is not present.

[student@workstation ~]$ **lab start operators-review**

**Instructions**

In this exercise, you install the Compliance operator. For more information, refer to the Compliance Operator chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance/index#compliance-operator>.

1. Log in to your OpenShift cluster as the admin user with the redhatocp password.
   1. Log in to the cluster as the admin user.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Examine the package manifest for the Compliance operator to discover the operator name, catalog name, suggested namespace, and channel.
   1. Use the oc command to list the package manifest resources.
   2. [student@workstation ~]$ **oc get packagemanifest**
   3. NAME CATALOG AGE
   4. lvms-operator do280 Operator Catalog Cs 2d5h
   5. file-integrity-operator do280 Operator Catalog Cs 2d5h
   6. metallb-operator do280 Operator Catalog Cs 2d5h
   7. **compliance-operator do280 Operator Catalog Cs 2d5h**

kubevirt-hyperconverged do280 Operator Catalog Cs 2d5h

* 1. Examine the compliance-operator package manifest.
  2. [student@workstation ~]$ **oc get packagemanifest compliance-operator -o yaml**
  3. apiVersion: packages.operators.coreos.com/v1
  4. kind: PackageManifest
  5. metadata:
  6. creationTimestamp: "2024-01-24T14:05:27Z"
  7. labels:
  8. **catalog: do280-catalog-cs**
  9. **catalog-namespace: openshift-marketplace**
  10. ...output omitted...
  11. name: compliance-operator
  12. namespace: default
  13. spec: {}
  14. status:
  15. ...output omitted...
  16. channels:
  17. - currentCSV: compliance-operator.v1.4.0
  18. currentCSVDesc:
  19. annotations:
  20. alm-examples: |-
  21. ...output omitted...
  22. **operatorframework.io/suggested-namespace: openshift-compliance**
  23. ...output omitted...
  24. version: 1.4.0
  25. name: **stable**
  26. defaultChannel: **stable**
  27. **packageName: compliance-operator**

...output omitted...

The package manifest contains the following information:

| **Field** | **Value** |
| --- | --- |
| catalog | do280-catalog-cs |
| catalog-namespace | openshift-marketplace |
| suggested-namespace | openshift-compliance |
| defaultChannel | stable |
| packageName | compliance-operator |

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the recommended openshift-compliance namespace.
   1. Use the oc command to create the namespace.
   2. [student@workstation ~]$ **oc create namespace openshift-compliance**

namespace/openshift-compliance created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create an operator group with the compliance-operator name in the openshift-compliance namespace. The target namespace of the operator group is the openshift-compliance namespace. You can use the ~/DO280/labs/operators-review/operator-group.yaml file as a template.
   1. Create an operator-group.yaml file with the following content:
   2. apiVersion: operators.coreos.com/v1
   3. kind: OperatorGroup
   4. metadata:
   5. name: **compliance-operator**
   6. namespace: **openshift-compliance**
   7. spec:
   8. targetNamespaces:

- **openshift-compliance**

* 1. Use the oc command to create the operator group:
  2. [student@workstation ~]$ **oc create -f operator-group.yaml**

operatorgroup.operators.coreos.com/compliance-operator created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create a compliance-operator subscription in the openshift-compliance namespace. The subscription has the following parameters:

| **Field** | **Value** |
| --- | --- |
| channel | stable |
| spec.name | compliance-operator |
| source | do280-catalog-cs |
| sourceNamespace | openshift-marketplace |

1. You can use the ~/DO280/labs/operators-review/subscription.yaml file as a template.
2. You can configure automatic install plan approvals.
   1. Create a subscription.yaml file with the following content:
   2. apiVersion: operators.coreos.com/v1alpha1
   3. kind: Subscription
   4. metadata:
   5. name: **compliance-operator**
   6. namespace: **openshift-compliance**
   7. spec:
   8. channel: **stable**
   9. installPlanApproval: Automatic
   10. name: **compliance-operator**
   11. source: **do280-catalog-cs**

sourceNamespace: **openshift-marketplace**

* 1. Use the oc command to create the operator group:
  2. [student@workstation ~]$ **oc create -f subscription.yaml**

subscription.operators.coreos.com/compliance-operator created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Wait until the operator is installed.

The Operator Lifecycle Manager creates a cluster service version in the openshift-compliance namespace. Wait until the cluster service version resource (CSV) is in the Succeeded phase.

Although the CSV defines a single compliance-operator deployment, the operator has two additional deployments. Wait until the compliance-operator, ocp4-openshift-compliance-pp, and rhcos4-openshift-compliance-pp deployments are ready.

* 1. Select the openshift-compliance project.
  2. [student@workstation ~]$ **oc project openshift-compliance**

Now using project "openshift-compliance" on server "https://api.ocp4.example.com:6443".

* 1. Wait until the CSV is in the Succeeded phase.
  2. [student@workstation ~]$ **oc get csv**
  3. NAME DISPLAY VERSION ... PHASE
  4. compliance-operator.v1.4.0 Compliance Operator 1.4.0 ... Succeeded

...output omitted...

The available CSV version in the lab might change. Commands in the following steps require you to replace the available version in the lab.

* 1. Inspect the CSV to view the operator deployment. Replace the version that you obtained in a previous step. The .spec.install.spec.deployments JSONPath expression describes the location of the operator deployments in the CSV resource. Optionally, use the jq command to indent the output.
  2. [student@workstation ~]$ **oc get csv compliance-operator.*v1.4.0* \**
  3. **-o jsonpath={.spec.install.spec.deployments} | jq**
  4. [
  5. {
  6. "name": "compliance-operator",
  7. "spec": {

...output omitted...

The Compliance operator describes a single deployment with the compliance-operator name.

* 1. Use the oc command to list the workloads in the operator namespace.
  2. [student@workstation ~]$ **oc get all**
  3. NAME ...
  4. pod/compliance-operator-... ...
  5. pod/ocp4-openshift-compliance-pp-... ...
  6. pod/rhcos4-openshift-compliance-pp-... ...
  7. ...output omitted...
  8. NAME READY ...
  9. deployment.apps/compliance-operator 1/1 ...
  10. deployment.apps/ocp4-openshift-compliance-pp 1/1 ...
  11. deployment.apps/rhcos4-openshift-compliance-pp 1/1 ...

...output omitted...

Besides the compliance-operator deployment, the Compliance operator creates two other deployments.

Wait until all deployments are ready.

[Hide Solution](https://rol.redhat.com/rol/app/)

1. Verify that the operator works correctly.

This operator watches custom resources of the ScanSettingBinding type and runs file integrity checks on cluster nodes. The operator reports results with custom resources of the ComplianceSuite type.

Create a scan setting binding in the openshift-compliance namespace. You can use the ~/DO280/labs/operators-review/scan-setting-binding.yaml file as a template.

You can also use the web console to create the scan setting binding. The YAML editor in the web console provides the same scan setting binding resource as an example.

Wait until a resource of the ComplianceSuite type in the DONE phase is present in the openshift-compliance namespace.

* 1. Examine the alm-examples annotation in the CSV. Replace the version that you obtained in a previous step.
  2. [student@workstation ~]$ **oc get csv compliance-operator.*v1.4.0* \**
  3. **-o jsonpath={.metadata.annotations.alm-examples} | jq**
  4. [
  5. ...output omitted...
  6. {
  7. "apiVersion": "compliance.openshift.io/v1alpha1",
  8. "kind": "ScanSettingBinding",
  9. "metadata": {
  10. "name": "nist-moderate"
  11. },
  12. "profiles": [
  13. {
  14. "apiGroup": "compliance.openshift.io/v1alpha1",
  15. "kind": "Profile",
  16. "name": "rhcos4-moderate"
  17. }
  18. ],
  19. "settingsRef": {
  20. "apiGroup": "compliance.openshift.io/v1alpha1",
  21. "kind": "ScanSetting",
  22. "name": "default"
  23. }
  24. },
  25. ...output omitted...

]

The annotation contains an example scan setting binding that you can use. The example is in JSON format. When creating a scan setting binding in the web console, the YAML editor loads the same example.

You can also use the oc explain command to describe the scan setting binding resource.

* 1. Create the scan setting binding resource by using the example file in the ~/DO280/labs/operators-review/scan-setting-binding.yaml path.
  2. [student@workstation ~]$ **oc create \**
  3. **-f ~/DO280/labs/operators-review/scan-setting-binding.yaml**

scansettingbinding.compliance.openshift.io/nist-moderate created

* 1. Use the oc command to list compliance suite and pod resources. Execute the command repeatedly until the compliance suite resource is in the DONE phase.
  2. [student@workstation ~]$ **oc get compliancesuite,pod**
  3. NAME PHASE RESULT
  4. compliancesuite.compliance.openshift.io/nist-moderate DONE NON-COMPLIANT
  5. NAME ...
  6. pod/compliance-operator-... ...
  7. pod/ocp4-openshift-compliance-pp-... ...

pod/rhcos4-openshift-compliance-pp-... ...

To execute the scan, the compliance operator creates extra pods. The pods disappear when the scan completes.

[Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade operators-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish operators-review**

# Chapter 8.  Application Security

[Control Application Permissions with Security Context Constraints](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08)

[Guided Exercise: Control Application Permissions with Security Context Constraints](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s02)

[Allow Application Access to Kubernetes APIs](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s03)

[Guided Exercise: Allow Application Access to Kubernetes APIs](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s04)

[Cluster and Node Maintenance with Kubernetes Cron Jobs](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s05)

[Guided Exercise: Cluster and Node Maintenance with Kubernetes Cron Jobs](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s06)

[Lab: Application Security](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s07)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch08s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Run applications that require elevated or special privileges from the host operating system or Kubernetes. |
| **Objectives** | * Create service accounts and apply permissions, and manage security context constraints. * Run an application that requires access to the Kubernetes API of the application's cluster. * Automate regular cluster and application management tasks by using Kubernetes cron jobs. |
| **Sections** | * Control Application Permissions with Security Context Constraints (and Guided Exercise) * Allow Application Access to Kubernetes APIs (and Guided Exercise) * Cluster and Node Maintenance with Kubernetes Cron Jobs (and Guided Exercise) |
| **Lab** | * Application Security |

## **Control Application Permissions with Security Context Constraints**

### **Objectives**

* Create service accounts and apply permissions, and manage security context constraints.

### **Security Context Constraints (SCCs)**

Red Hat OpenShift provides security context constraints (SCCs), a security mechanism that limits the access from a running pod in OpenShift to the host environment. SCCs control the following host resources:

* Running privileged containers
* Requesting extra capabilities for a container
* Using host directories as volumes
* Changing the SELinux context of a container
* Changing the user ID

Some community-developed containers might require relaxed security context constraints to access resources that are forbidden by default, such as file systems or sockets, or to access an SELinux context.

Cluster administrators can run the following command to list the SCCs that OpenShift defines:

[user@host ~]$ **oc get scc**

OpenShift provides the following default SCCs:

* anyuid
* hostaccess
* hostmount-anyuid
* hostnetwork
* hostnetwork-v2
* lvms-topolvm-node
* lvms-vgmanager
* machine-api-termination-handler
* node-exporter
* nonroot
* nonroot-v2
* privileged
* restricted
* restricted-v2

For additional information about an SCC, use the oc describe command:

[user@host ~]$ **oc describe scc anyuid**

Name: anyuid

Priority: 10

Access:

Users: <none>

Groups: system:cluster-admins

Settings:

...output omitted...

Most pods that OpenShift creates use the restricted-v2 SCC, which provides limited access to resources that are external to OpenShift. Use the oc describe command to view the security context constraint that a pod uses.

[user@host ~]$ **oc describe pod console-5df4fcbb47-67c52 \**

**-n openshift-console | grep scc**

openshift.io/scc: restricted-v2

Container images that are downloaded from public container registries, such as Docker Hub, might fail to run when using the restricted-v2 SCC. For example, a container image that requires running as a specific user ID can fail because the restricted-v2 SCC runs the container by using a random user ID. A container image that listens on port 80 or on port 443 can fail for a related reason. The random user ID that the restricted-v2 SCC uses cannot start a service that listens on a privileged network port (port numbers that are less than 1024). Use the scc-subject-review subcommand to list all the security context constraints that can overcome the limitations that hinder the container:

[user@host ~]$ **oc get deployment *deployment-name* -o yaml | \**

**oc adm policy scc-subject-review -f -**

The anyuid SCC defines the run as user strategy to be RunAsAny, which means that the pod can run as any available user ID in the container. With this strategy, containers that require a specific user can run the commands by using a specific user ID.

To change the container to run with a different SCC, you must create a service account that is bound to a pod. Use the oc create serviceaccount command to create the service account, and use the -n option if the service account must be created in a different namespace from the current one:

[user@host ~]$ **oc create serviceaccount *service-account-name***

To associate the service account with an SCC, use the oc adm policy command. Identify a service account by using the -z option, and use the -n option if the service account exists in a different namespace from the current one:

[user@host ~]$ **oc adm policy add-scc-to-user *SCC* -z *service-account***

### IMPORTANT

Only cluster administrators can assign an SCC to a service account or remove an SCC from a service account. Allowing pods to run with a less restrictive SCC can make your cluster less secure. Use with caution.

Change an existing deployment or deployment configuration to use the service account by using the oc set serviceaccount command:

[user@host ~]$ **oc set serviceaccount deployment/*deployment-name* \**

***service-account-name***

If the command succeeds, then the pods that are associated with the deployment or deployment configuration redeploy.

### **Privileged Containers**

Some containers might need to access the runtime environment of the host. For example, the S2I builder class of privileged containers requires access beyond the limits of its own containers. These containers can pose security risks, because they can use any resources on an OpenShift node. Use SCCs to enable access for privileged containers by creating service accounts with privileged access.

### REFERENCES

For more information, refer to the Managing Security Context Constraints chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#managing-pod-security-policies>

## **Guided Exercise: Control Application Permissions with Security Context Constraints**

Deploy applications that require pods with extended permissions.

**Outcomes**

* Create service accounts and assign security context constraints (SCCs) to them.
* Assign a service account to a deployment configuration.
* Run applications that need root privileges.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates some HTPasswd users for the exercise.

[student@workstation ~]$ **lab start appsec-scc**

**Instructions**

1. Log in to the OpenShift cluster and create the appsec-scc project.
   1. Log in to the cluster as the developer user with the developer password.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Create the appsec-scc project.
  2. [student@workstation ~]$ **oc new-project appsec-scc**
  3. Now using project "appsec-scc" on server ...

...output omitted...

1. Deploy an application named gitlab by using the container image at registry.ocp4.example.com:8443/redhattraining/gitlab-ce:8.4.3-ce.0. This image is a copy of the container image at docker.io/gitlab/gitlab-ce:8.4.3-ce.0. Verify that the reason for the pod failure is because the container image needs root privileges.
   1. Deploy the gitlab application.
   2. [student@workstation ~]$ **oc new-app --name gitlab \**
   3. **--image registry.ocp4.example.com:8443/redhattraining/gitlab-ce:8.4.3-ce.0**
   4. ...output omitted...
   5. --> Creating resources ...
   6. imagestream.image.openshift.io "gitlab" created
   7. deployment.apps "gitlab" created
   8. service "gitlab" created
   9. --> Success

...output omitted...

* 1. Determine whether the application is successfully deployed. It should give an error, because this image needs root privileges to deploy.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE

gitlab-d89cd88f8-jwqbp 0/1 **Error** 0 36s

### NOTE

It might take some time for the image to reach the Error state. You might also see the CrashLoopBackOff status when you validate the health of the pod.

* 1. Review the application logs to confirm that insufficient privileges caused the failure.
  2. [student@workstation ~]$ **oc logs pod/gitlab-*d89cd88f8-jwqbp***
  3. ...output omitted...
  4. ================================================================================
  5. Recipe Compile Error in /opt/gitlab/embedded/cookbooks/cache/cookbooks/gitlab/recipes/default.rb
  6. ================================================================================
  7. Chef::Exceptions::InsufficientPermissions
  8. -----------------------------------------
  9. directory[/etc/gitlab] (gitlab::default line 26) had an error: Chef::Exceptions::InsufficientPermissions: **Cannot create directory[/etc/gitlab] at /etc/gitlab due to insufficient permissions**

...output omitted...

The application tries to write to the /etc directory. To allow the application to write to the /etc directory, you can make the application run as the root user. To run the application as the root user, you can grant the anyuid SCC to a service account.

1. Create a service account and assign the anyuid SCC to it.
   1. Log in as the admin user with the redhatocp password.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Verify the appropriate SCC to use with this deployment.
  2. [student@workstation]$ **oc get deploy**
  3. NAME READY UP-TO-DATE AVAILABLE AGE
  4. **gitlab** 0/1 1 0 109s
  5. [student@workstation]$ **oc get deploy/gitlab -o yaml | oc adm policy \**
  6. **scc-subject-review -f -**
  7. RESOURCE ALLOWED BY

Deployment/gitlab **anyuid**

The output confirms that the anyuid SCC allows the gitlab deployment to create and update pods.

* 1. Create a service account named gitlab-sa.
  2. [student@workstation ~]$ **oc create sa gitlab-sa**

serviceaccount/gitlab-sa created

* 1. Assign the anyuid SCC to the gitlab-sa service account.
  2. [student@workstation ~]$ **oc adm policy add-scc-to-user anyuid -z gitlab-sa**

clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "gitlab-sa"

1. Modify the gitlab application to use the newly created service account. Verify that the new deployment succeeds.
   1. Log in as the developer user.
   2. [student@workstation ~]$ **oc login -u developer -p developer**
   3. Login successful.

...output omitted...

* 1. Assign the gitlab-sa service account to the gitlab deployment.
  2. [student@workstation ~]$ **oc set serviceaccount deployment/gitlab gitlab-sa**

deployment.apps/gitlab serviceaccount updated

* 1. Verify that the gitlab redeployment succeeds. You might need to run the oc get pods command multiple times until you see a running application pod.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE

gitlab-86d6d65-zm2fd 1/1 Running 0 55s

1. Verify that the gitlab application works.
   1. Expose the gitlab application. Because the gitlab service listens on ports 22, 80, and 443, you must use the --port option.
   2. [student@workstation ~]$ **oc expose service/gitlab --port 80 \**
   3. **--hostname gitlab.apps.ocp4.example.com**

route.route.openshift.io/gitlab exposed

* 1. Get the exposed route.
  2. [student@workstation ~]$ **oc get routes**
  3. NAME HOST/PORT PATH SERVICES PORT ...

gitlab gitlab.apps.ocp4.example.com gitlab 80 ...

* 1. Verify that the gitlab application is answering HTTP queries.
  2. [student@workstation ~]$ **curl -sL http://gitlab.apps.ocp4.example.com/ | \**
  3. **grep '<title>'**

<title>Sign in · GitLab</title>

1. Delete the appsec-scc project.
2. [student@workstation ~]$ **oc delete project appsec-scc**

project.project.openshift.io "appsec-scc" deleted

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish appsec-scc**

## **Allow Application Access to Kubernetes APIs**

### **Objectives**

* Run an application that requires access to the Kubernetes API of the application's cluster.

### **Securing Kubernetes APIs**

With the Kubernetes APIs, a user or an application can query and modify the cluster state. To protect your cluster from malicious interactions, you must grant access to the different Kubernetes APIs.

Role-based access control (RBAC) authorization is preconfigured in OpenShift. An application requires explicit RBAC authorization to access restricted Kubernetes APIs.

### **Application Authorization with Service Accounts**

A service account is a Kubernetes object within a project. The service account represents the identity of an application that runs in a pod.

To grant an application access to a Kubernetes API, take these actions:

* Create an application service account.
* Grant the service account access to the Kubernetes API.
* Assign the service account to the application pods.

If the pod definition does not specify a service account, then the pod uses the default service account. OpenShift grants no rights to the default service account, which is expected for business workloads. It is not recommended to grant additional permissions to the default service account, because it grants those additional permissions to all pods in the project, which might not be intended.

### **Use Cases for Kubernetes API Access**

Regular business applications can successfully use the default service account, without requiring access to the Kubernetes APIs. On the contrary, infrastructure applications need access to monitor or to modify the cluster resources. These infrastructure applications might be classified into the following use cases:

**Monitoring Applications**

Applications in this category need read access to watch cluster resources or to verify cluster health. For example, a service such as Red Hat Advanced Cluster Security (ACS) needs read access to scan your cluster containers for vulnerabilities.

**Controllers**

Controllers are applications that constantly watch and try to reach the intended state of a resource.

For example, GitOps tools, such as ArgoCD, have controllers that watch cluster resources that are stored in a repository, and update the cluster to react to changes in that repository.

**Operators**

Operators automate creating, configuring, and managing instances of Kubernetes-native applications. Therefore, operators need permissions for configuration and maintenance tasks.

For example, a database operator might create a deployment when it detects a CR that defines a new database.

#### **Application Kubernetes API Authorization with Roles**

To provide the application with the needed permissions only, you can create roles or cluster roles that describe the application requirements. Roles grant permissions to Kubernetes API resources within a single namespace. Cluster roles grant permissions, either within one or more namespaces, or to all the cluster.

For example, you can create a cluster role for an application to read secrets.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: secret-reader

rules:

- apiGroups: [""]

resources: ["secrets"]

verbs: ["get", "watch", "list"]

|  |  |
| --- | --- |
|  | The API groups, where an empty string represents the core API |
|  | The resources that the role refers to |
|  | The verbs or actions that the role allows the application to perform on the resource |

You can also use the default cluster roles that OpenShift defines, which have wider permissions. For example, you can use the edit cluster role to get read access on secrets, as in the previous secret-reader cluster role.

The edit cluster role is less restrictive, and allows the application to create or update most objects.

#### **Binding Roles to Service Accounts**

For an application to use the role permissions, you must bind the role or cluster role to the application service account.

To bind a role or cluster role to a service account in a namespace, you can use the oc adm policy command with the add-role-to-user subcommand.

This command assigns a cluster role to a service account that exists in the current project:

[user@host ~]$ **oc adm policy add-role-to-user *cluster-role* -z *service-account***

You can optionally use -z to avoid specifying the system:serviceaccount:*project* prefix when you assign the role to a service account that exists in the current project.

To create a cluster role binding, you can use the oc adm policy command with the add-cluster-role-to-user subcommand.

The following command assigns a cluster role to a service account with a cluster scope:

[user@host ~]$ **oc adm policy add-cluster-role-to-user *cluster-role* *service-account***

#### **Assigning an Application Service Account to Pods**

OpenShift uses RBAC authorization by using the roles that are associated to the service account to grant or deny access to the resource. You specify the service account name in the spec.serviceAccountName pod definition field.

Applications must use the service account token internally when accessing a Kubernetes API. In earlier OpenShift versions than 4.11, OpenShift generated a secret with a token when creating a service account. Starting from OpenShift 4.11, tokens are no longer generated automatically. You must use the TokenRequest API to generate the service account token. You must mount the token as a pod volume for the application to access it.

### **Scoping Application Access to Kubernetes API Resources**

An application might require access to a resource in the same namespace, or in a different namespace, or in all namespaces.

#### **Accessing API Resources in the Same Namespace**

To grant an application access to resources in the same namespace, you need a role or a cluster role and a service account in that namespace. You then create a role binding that associates to the service account the actions that the role grants on the resource. Using a role binding with a cluster role grants access only to the resource within the namespace.

#### **Accessing API Resources in a Different Namespace**

To give an application access to a resource in a different namespace, you must create the role binding in the project with the resource. The subject for the binding references the application service account that is in a different namespace from the binding.

You can use the following syntax to refer to service accounts from other projects:

system:serviceaccount:***project***:***service-account***

For example, if you have an application pod in the project-1 project that requires access to project-2 secrets, then you must take these actions:

* Create an app-sa service account in the project-1 project.
* Assign the app-sa service account to your application pod.
* Create a role binding on the project-2 project that references the app-sa service account and the secret-reader role or cluster role.

|  |
| --- |
|  |

Figure 8.1: Grant access to a service account to a different project

In this way, you restrict an application's access to a Kubernetes API to specified namespaces.

#### **Accessing API Resources in All Namespaces**

Grant your application service account the cluster role by using a cluster role binding. The cluster role binding grants the application cluster access to the API.

### REFERENCES

For more information, refer to the Using RBAC to Define and Apply Permissions chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#using-rbac>

For more information, refer to the Understanding and Creating Service Accounts chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#understanding-and-creating-service-accounts>

For more information, refer to the Using Service Accounts in Applications chapter in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#using-service-accounts>

For more information, refer to the About Automatically-generated Service Account Token Secrets section in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#auto-generated-sa-token-secrets_using-service-accounts>

## **Guided Exercise: Allow Application Access to Kubernetes APIs**

Configure an application with limited access to Kubernetes API resources.

**Outcomes**

You should be able to grant Kubernetes API access to an application by using a service account that has a role with the required privileges.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

The lab command copies the following files to the lab directory:

* The deployment manifest to install the Stakater Reloader application, at <https://github.com/stakater/Reloader>. This application is a controller that watches for changes in configuration maps and does rolling upgrades on associated deployments.
* The manifests to install the config-app API, which has an endpoint to show its internal configuration. The deployment manifest mounts the API configuration from a configuration map.

In this exercise, you grant permissions on the appsec-api project to the Reloader application, for read access to the configuration map API and edit access to the deployment API.

### WARNING

Using a controller to update a Kubernetes resource by reacting to changes is an alternative to, and might conflict with, using GitOps.

[student@workstation ~]$ **lab start appsec-api**

**Instructions**

1. Change to the lab directory.
   1. Change to the ~/DO280/labs/appsec-api directory.

[student@workstation ~]$ **cd ~/DO280/labs/appsec-api**

1. Log in as the admin user and change to the configmap-reloader project.
   1. Open a terminal window and log in as the admin user with the redhatocp password.
   2. [student@workstation appsec-api]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Use the oc project command to change to the configmap-reloader namespace.
  2. [student@workstation appsec-api]$ **oc project configmap-reloader**

Now using project "configmap-reloader" on server ...

1. Create the configmap-reloader service account to hold the permissions for the Reloader application. Then, assign the configmap-reloader service account to the configmap-reloader deployment.
   1. Create the configmap-reloader service account.
   2. [student@workstation appsec-api]$ **oc create sa configmap-reloader-sa**

serviceaccount/configmap-reloader-sa created

* 1. Add the configmap-reloader-sa service account to the deployment in the reloader-deployment.yaml file.
  2. apiVersion: apps/v1
  3. kind: Deployment
  4. metadata:
  5. labels:
  6. app: configmap-reloader
  7. name: configmap-reloader
  8. namespace: configmap-reloader
  9. spec:
  10. selector:
  11. matchLabels:
  12. app: configmap-reloader
  13. release: "reloader"
  14. template:
  15. metadata:
  16. labels:
  17. app: configmap-reloader
  18. spec:
  19. **serviceAccountName: configmap-reloader-sa**
  20. containers:

...output omitted...

* 1. Use the oc command to create the configmap-reloader deployment from the reloader-deployment.yaml file.
  2. [student@workstation appsec-api]$ **oc apply -f reloader-deployment.yaml**

deployment.apps/configmap-reloader created

1. As the developer user, create the appsec-api project.
   1. Log in to the cluster as the developer user with the developer password.
   2. [student@workstation appsec-api]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Use the oc new-project command to create the appsec-api project.
  2. [student@workstation appsec-api]$ **oc new-project appsec-api**

Now using project "appsec-api" on server ...

1. Grant permissions to the configmap-reloader-sa service account to watch configuration map resources and roll out deployments on the appsec-api project.
   1. Assign the edit`cluster role to the `configmap-reloader-sa service account in the appsec-api project. To assign the cluster role, create a local role binding by using the oc policy add-role-to-user command with the following options:
      * The edit default cluster role.
      * The system:serviceaccount:configmap-reloader:configmap-reloader-sa username to reference the configmap-reloader-sa service account in the configmap-reloader project.
      * The --rolebinding-name option to use the reloader-edit name for the role binding.
      * The -n appsec-api, which is optional because you are already in the appsec-api project.
   2. [student@workstation appsec-api]$ **oc policy add-role-to-user edit \**
   3. **system:serviceaccount:configmap-reloader:configmap-reloader-sa \**
   4. **--rolebinding-name=reloader-edit \**
   5. **-n appsec-api**

clusterrole.rbac.authorization.k8s.io/edit added: "system:serviceaccount:configmap-reloader:configmap-reloader-sa"

### NOTE

1. The edit cluster role with the local role binding allows the configmap-reloader-sa service account to modify most objects in the appsec-api project. In a production scenario, it is best to grant access only to the APIs that your application requires.
2. Install the config-app API by using the manifest files in the config-app directory.
   1. Use the oc apply command with the -f option to create all the manifests in the config-app directory.
   2. [student@workstation appsec-api]$ **oc apply -f ./config-app**
   3. configmap/config-app created
   4. deployment.apps/config-app created
   5. route.route.openshift.io/config-app created

service/config-app created

* 1. Read the config.yaml content from the config-app configuration map by running the oc get command.
  2. [student@workstation appsec-api]$ **oc get configmap config-app \**
  3. **--output="jsonpath={.data.config\.yaml}"**
  4. application:
  5. name: "config-app"

description: "config-app"

* 1. Run the curl command to verify that the exposed route, https://config-app-appsec-api.apps.ocp4.example.com/config, shows the config-app configuration map content.
  2. [student@workstation appsec-api]$ **curl -s \**
  3. **https://config-app-appsec-api.apps.ocp4.example.com/config | jq**
  4. {
  5. "application": {
  6. "description": "config-app",
  7. "name": "config-app"
  8. }

}

1. Configure the config-app deployment with the configmap.reloader.stakater.com/reload: "config-app" annotation so that the controller can roll out deployments automatically when the config-app configuration map changes.
   1. Add the configmap.reloader.stakater.com/reload: "config-app" annotation to the deployment in the config-app/deployment.yaml file.
   2. apiVersion: apps/v1
   3. kind: Deployment
   4. metadata:
   5. name: config-app
   6. namespace: appsec-api
   7. **annotations:**
   8. **configmap.reloader.stakater.com/reload: "config-app"**
   9. spec:

...output omitted...

* 1. Use the oc apply command to update the resource.
  2. [student@workstation appsec-api]$ **oc apply -f config-app/deployment.yaml**

deployment.apps/config-app configured

* 1. Verify that the configmap.reloader.stakater.com/reload: "config-app" annotation is present in the config-app deployment object.
  2. [student@workstation appsec-api]$ **oc get deployment config-app -o yaml**
  3. apiVersion: apps/v1
  4. kind: Deployment
  5. metadata:
  6. annotations:
  7. **configmap.reloader.stakater.com/reload: config-app**
  8. spec:

...output omitted...

1. Update the config-app configuration map description key and query /config endpoint to verify that the Reloader controller upgrades the config-app deployment.
   1. Update the description data in the configuration map in the config-app/configmap.yaml file to the API that exposes its configuration value.
   2. apiVersion: v1
   3. kind: ConfigMap
   4. metadata:
   5. name: config-app
   6. namespace: appsec-api
   7. data:
   8. config.yaml: |
   9. application:
   10. name: "config-app"

description: **"API that exposes its configuration"**

* 1. Use the oc command to apply the changes to the config-app/configmap.yaml file.
  2. [student@workstation appsec-api]$ **oc apply -f config-app/configmap.yaml**

configmap/config-app configured

* 1. Use the watch command to query the API /config endpoint by using the curl command to verify that the API configuration changes. Press **Ctrl**+**C** to exit.
  2. [student@workstation appsec-api]$ **watch \**
  3. **"curl -s https://config-app-appsec-api.apps.ocp4.example.com/config | jq"**
  4. Every 2.0s: curl -s https://config-app-appsec-api.apps.ocp4.example.com/config | jq
  5. workstation: ...
  6. {
  7. "application": {
  8. "description": **"API that exposes its configuration"**,
  9. "name": "config-app"
  10. }

}

Wait until the controller application upgrades the deployment.

1. Change to the home directory to complete the exercise.
   1. Change to the home directory.

[student@workstation appsec-api]$ **cd**

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish appsec-api**

## **Cluster and Node Maintenance with Kubernetes Cron Jobs**

### **Objectives**

* Automate regular cluster and application management tasks with Kubernetes cron jobs.

### **Maintenance Tasks**

Cluster administrators can use scheduled tasks to automate maintenance tasks in the cluster. Other users can create scheduled tasks for regular application maintenance.

Maintenance tasks vary in the privileges that they require. Cluster maintenance tasks require privileged pods, whereas most applications might not require elevated privileges.

### **Kubernetes Batch API Resources**

You can automate tasks in OpenShift by using standard Kubernetes jobs and cron jobs. The automated tasks can be configured to run once or on a regular schedule.

**Job**

Kubernetes jobs specify a task that is executed once.

**Cron Job**

Kubernetes cron jobs have a schedule to execute a task regularly.

When a cron job is due for execution, Kubernetes creates a job resource. Kubernetes creates these jobs from a template in the cron job definition. Other than this relationship, Kubernetes jobs and cron jobs are workload resource types, such as deployments or daemon sets.

#### **Kubernetes Jobs**

The job resource includes a pod template that describes the task to execute. You can use the oc create job --dry-run=client command to get the YAML representation of the Kubernetes job resource:

[user@host ~]$ **oc create job --dry-run=client -o yaml test \**

**--image=registry.access.redhat.com/ubi8/ubi:8.6 \**

**-- curl https://example.com**

A job contains a pod template, and this pod template must specify at least one container. You can add metadata such as labels or annotations to the job definition and pod template.

apiVersion: batch/v1

kind: Job

metadata:

creationTimestamp: null

name: test

spec:

template:

metadata:

creationTimestamp: null

spec:

containers:

- command:

- curl

- https://example.com

image: registry.access.redhat.com/ubi8/ubi:8.6

name: test

resources: {}

restartPolicy: Never

status: {}

|  |  |
| --- | --- |
|  | Job specification |
|  | Pod template |
|  | Pod specification |
|  | Pod containers |
|  | Command |
|  | Container image |

#### **Kubernetes Cron Jobs**

The cron job resource includes a job template that describes the task and a schedule. You can use the oc create cronjob --dry-run=client command to get the YAML representation of the Kubernetes cron job resource:

[user@host ~]$ **oc create cronjob --dry-run=client -o yaml test \**

**--image=registry.access.redhat.com/ubi8/ubi:8.6 \**

**--schedule='0 0 \* \* \*' \**

**-- curl https://example.com**

In Kubernetes, cron job resources are similar to job resources. The jobTemplate key follows the same structure as a job. The schedule key describes when the task runs.

apiVersion: batch/v1

kind: CronJob

metadata:

creationTimestamp: null

name: test

spec:

jobTemplate:

metadata:

creationTimestamp: null

name: test

spec:

template:

metadata:

creationTimestamp: null

spec:

containers:

- command:

- curl

- https://example.com

image: registry.access.redhat.com/ubi8/ubi:8.6

name: test

resources: {}

restartPolicy: OnFailure

schedule: 0 0 \* \* \*

status: {}

|  |  |
| --- | --- |
|  | Cron job specification |
|  | Job template |
|  | Job specification |
|  | Pod template |
|  | Pod specification |
|  | Command |
|  | Container image |
|  | Cron job schedule specification |

#### **Linux Cron Jobs**

The schedule specification for Kubernetes cron jobs is derived from the specification in Linux cron jobs. The crontab file specifies the scheduled tasks for the current user. The schedule specification has five fields to define the date and time when the job is executed. The /etc/crontab file comments include a syntax diagram:

# Example cron job definition:

# ┌───────────────── **minute** (0 - 59)

# │ ┌────────────── **hour** (0 - 23)

# │ │ ┌────────── **day of month** (1 - 31)

# │ │ │ ┌────── **month** (1 - 12) or jan,feb,mar,apr ...

# │ │ │ │ ┌── **day of week** (0 - 7) or sun,mon,tue,wed,thu,fri,sat

# │ │ │ │ │ (Sunday is 0 or 7)

# m h dom mon dow command

**0 \*/2 \* \* \*** **/path/to/task\_executable arguments**

Some examples of cron job specifications are as follows:

| **Schedule specification** | **Description** |
| --- | --- |
| **0 0 \* \* \*** | Run the specified task every day at midnight |
| **0 0 \* \* 7** | Run the specified task every Sunday at midnight |
| **0 \* \* \* \*** | Run the specified task every hour |
| **0 \*/4 \* \* \*** | Run the specified task every four hours |

### NOTE

Refer to the crontab(5) manual page for more information about the cron job schedule specification.

### **Automate Maintenance Tasks with Cron Jobs**

You can automate the maintenance tasks for applications that run inside the cluster, and also execute low-level commands inside privileged debug pods to apply cluster maintenance tasks.

#### **Automating Application Maintenance Tasks**

Regular maintenance tasks might need to run for applications that run in the cluster.

For example, consider creating periodic backups for an application. This application requires the following steps to create the backup:

* Activate maintenance mode.
* Create a compressed database backup.
* Deactivate maintenance mode.
* Copy the database backup to an external location.

The following cron job definition shows a possible implementation of these steps:

apiVersion: batch/v1

kind: CronJob

metadata:

name: wordpress-backup

spec:

**schedule: 0 2 \* \* 7**

jobTemplate:

spec:

template:

spec:

dnsPolicy: ClusterFirst

restartPolicy: Never

containers:

- name: wp-cli

image: ***registry.io/wp-maintenance/wp-cli:2.7***

resources: {}

command:

- **bash**

- **-xc**

args:

- **>**

**wp maintenance-mode activate ;**

**wp db export | gzip > database.sql.gz ;**

**wp maintenance-mode deactivate ;**

**rclone copy database.sql.gz *s3://bucket/backups/* ;**

**rm -v database.sql.gz ;**

|  |  |
| --- | --- |
|  | Schedule for every Sunday at 2 AM |
|  | The Kubernetes job template |
|  | The Kubernetes pod template |
|  | The Kubernetes pod specification |
|  | The pod container configuration |
|  | The container image that runs the maintenance task |
|  | The command to execute inside the pod |
|  | Maintenance commands to execute |

### NOTE

The > symbol uses the YAML folded style, which converts all newlines to spaces when parsing. Each command is separated with a semicolon (;), because the string in the args key is passed as a single argument to the bash -xc command.

This combination of the command and args keys has the same effect as executing the commands in a single line inside the container:

[user@host ~]$ **bash -xc 'wp maintenance-mode activate ; wp db export | gzip > database.sql.gz ; wp maintenance-mode deactivate ; rclone copy database.sql.gz *s3://bucket/backups/* ; rm -v database.sql.gz ;'**

For more information about the YAML folded style, refer to <https://yaml.org/spec/1.2.2/#folded-style>

#### **Automating Cluster Maintenance Tasks**

Cluster maintenance might require executing complex scripts in privileged pods. You can create a shell script with the commands to execute the maintenance task, and mount the script in the pod by using a configuration map.

For example, when images are updated, clusters might accumulate unused images. These images might occupy much space. Executing the crictl rmi --prune command on all nodes of the cluster frees this space.

The following configuration map contains a shell script that cleans images in all cluster nodes by executing a debug pod and running the crictl command with the chroot command to access the root file system of the node:

apiVersion: v1

kind: ConfigMap

metadata:

name: maintenance

app: crictl

data:

maintenance.sh: |

#!/bin/bash

NODES=$(oc get nodes -o=name)

for NODE in ${NODES}

do

echo ${NODE}

oc debug ${NODE} -- \

chroot /host \

/bin/bash -xc 'crictl images ; crictl rmi --prune'

echo $?

done

|  |  |
| --- | --- |
|  | List the nodes in the cluster. |
|  | Iterate over the nodes. |
|  | Run a debug pod on the node. |
|  | Prune the images. |

This task can be scheduled regularly by using a cron job. The quay.io/openshift/origin-cli:4.14 container provides the oc command that runs the debug pod. The pod mounts the configuration map and executes the maintenance script.

apiVersion: batch/v1

kind: CronJob

metadata:

name: image-pruner

spec:

schedule: 0 \* \* \* \*

jobTemplate:

spec:

template:

spec:

dnsPolicy: ClusterFirst

restartPolicy: Never

containers:

- name: image-pruner

image: quay.io/openshift/origin-cli:4.14

resources: {}

command:

- /opt/scripts/maintenance.sh

volumeMounts:

- name: scripts

mountPath: /opt

volumes:

- name: scripts

configMap:

name: maintenance

defaultMode: 0555

|  |  |
| --- | --- |
|  | Path to the script |
|  | Mounting the configuration map as a volume |

Cluster maintenance tasks might require elevated privileges. Administrators can assign service accounts to any workload, including Kubernetes jobs and cron jobs.

You can create a service account with the required privileges, and specify the service account with the serviceAccountName key in the pod definition. You can also use the oc set serviceaccount command to change the service account of an existing workload.

### REFERENCES

[Kubernetes Job](https://kubernetes.io/docs/concepts/workloads/controllers/job/)

[Kubernetes Cron Job](https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/)

[How to Delete Exited Containers and Dangling Images with crictl?](https://access.redhat.com/solutions/5610941)

## **Lab: Application Security**

Deploy an application that requires additional operating system privileges to run.

Deploy an application that requires access to the Kubernetes APIs to perform cluster maintenance tasks.

**Outcomes**

* Deploy a cluster maintenance application that must be executed regularly.
* Grant application access to Kubernetes APIs.
* Run an application with a security context constraint (SCC).

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start appsec-review**

In this exercise, you deploy two applications:

* A legacy payroll application that must run as the fixed 0 UID to open the TCP 80 port.
* A project cleaner deletes projects with the appsec-review-cleaner label and that are longer than 10 seconds. This short expiration time is deliberate for the lab purposes.

You must deploy the project cleaner application to delete obsolete projects every minute.

The lab start command copies the required files for the exercise to the lab directory:

* A deployment manifest with the payroll application.
* A pod manifest that contains a project cleaner application. You can use this pod to test the project cleaner application and copy the pod specification into the cron job to complete the exercise.
* A manifest with the project-cleaner cluster role that grants the application access to find and delete namespaces.
* A cron job template file that you can edit to create cron jobs.
* A script that generates projects to verify that the project cleaner application works.

**Instructions**

1. Log in to your OpenShift cluster as the developer user with the developer password and create the appsec-review project.
   1. Log in as the developer user.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Create the appsec-review project.
  2. [student@workstation ~]$ **oc new-project appsec-review**
  3. Now using project "appsec-review" on server ...

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Change to the ~/DO280/labs/appsec-review directory and deploy the payroll application in the payroll-app.yaml file. Verify that the application cannot run.
   1. Change to the ~/DO280/labs/appsec-review directory to access the lab files.

[student@workstation ~]$ **cd ~/DO280/labs/appsec-review**

* 1. Run the oc apply command to create the payroll deployment.
  2. [student@workstation appsec-review]$ **oc apply -f payroll-app.yaml**

deployment.apps/payroll-api created

* 1. Verify that the application fails to run by reading the deployment logs.
  2. [student@workstation appsec-review]$ **oc logs deployment/payroll-api**
  3. [2023-03-13 08:13:30 +0000] [1] [INFO] Starting gunicorn 20.1.0
  4. [2023-03-13 08:13:30 +0000] [1] [ERROR] Retrying in 1 second.
  5. [2023-03-13 08:13:31 +0000] [1] [ERROR] Retrying in 1 second.
  6. [2023-03-13 08:13:32 +0000] [1] [ERROR] Retrying in 1 second.
  7. [2023-03-13 08:13:33 +0000] [1] [ERROR] Retrying in 1 second.
  8. [2023-03-13 08:13:34 +0000] [1] [ERROR] Retrying in 1 second.

[2023-03-13 08:13:35 +0000] [1] [ERROR] **Can't connect to ('', 80)**

The container in the pod runs as root to listen on port 80.

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. As the admin user, look for an SCC that allows the workload in the payroll-app.yaml deployment to run.
   1. Log in as the admin user with the redhatocp password.
   2. [student@workstation appsec-review]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Run the oc adm policy scc-subject-review command to get an SCC that allows the application to run.
  2. [student@workstation appsec-review]$ **oc adm policy scc-subject-review \**
  3. **-f payroll-app.yaml**
  4. RESOURCE ALLOWED BY

Deployment/payroll-api **anyuid**

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the payroll-sa service account and assign to it the SCC that the application requires. Then, assign the payroll-sa service account to the payroll-api deployment.
   1. Run the oc create command to create the payroll-sa service account.
   2. [student@workstation appsec-review]$ **oc create sa payroll-sa**

serviceaccount/payroll-sa created

* 1. Assign the anyuid SCC to the payroll-sa service account.
  2. [student@workstation appsec-review]$ **oc adm policy \**
  3. **add-scc-to-user anyuid -z payroll-sa**

clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "payroll-sa"

* 1. Use the oc set serviceaccount command to add the payroll-sa service account to the payroll-api deployment.
  2. [student@workstation appsec-review]$ **oc set serviceaccount deployment \**
  3. **payroll-api payroll-sa**

deployment.apps/payroll-api serviceaccount updated

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Verify that the payroll API is accessible by running the curl command from the payroll-api deployment. Use the http://localhost/payments/status URL to verify that the API is working.
   1. Use the oc exec command with the payroll-api deployment to run the curl command. Provide the -sS option to hide progress output and show errors.
   2. [student@workstation appsec-review]$ **oc exec deployment/payroll-api \**
   3. **-- curl -sS http://localhost/payments/status**

[{"id":240,"status":"Paid","userId":1003},{"id":241,"status":"Pending","userId":1003}]

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the project-cleaner-sa service account and assign it to the project-cleaner.yaml pod manifest to configure the application permissions.
   1. Create the project-cleaner-sa service account.
   2. [student@workstation appsec-review]$ **oc create sa project-cleaner-sa**

serviceaccount/project-cleaner-sa created

* 1. Edit the project-cleaner.yaml pod manifest file to use the project-cleaner-sa service account.
  2. apiVersion: v1
  3. kind: Pod
  4. metadata:
  5. name: project-cleaner
  6. namespace: appsec-review
  7. spec:
  8. restartPolicy: Never
  9. **serviceAccountName: project-cleaner-sa**
  10. containers:
  11. - name: project-cleaner

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the project-cleaner role in the cluster-role.yaml file and assign it to the project-cleaner-sa service account.
   1. Create the project-cleaner cluster role by applying the cluster-role.yaml manifest file.
   2. [student@workstation appsec-review]$ **oc apply -f cluster-role.yaml**

clusterrole.rbac.authorization.k8s.io/project-cleaner created

* 1. Use the oc adm policy add-clusterrole-to-user command to add the project-cleaner role to the project-cleaner-sa service account.
  2. [student@workstation appsec-review]$ **oc adm policy add-cluster-role-to-user \**
  3. **project-cleaner -z project-cleaner-sa**

clusterrole.rbac.authorization.k8s.io/project-cleaner added: "project-cleaner-sa"

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Edit the cron-job.yaml file to create the appsec-review-cleaner cron job by using the project-cleaner.yaml pod manifest as the job template. Create the cron job and configure it to run every minute. You can use the solution file in the ~/DO280/solutions/appsec-review/cron-job.yaml path.
   1. Edit the cron-job.yaml file to replace the CHANGE\_ME string with the "\*/1 \* \* \* \*" schedule to execute the job every minute.
   2. apiVersion: batch/v1
   3. kind: CronJob
   4. metadata:
   5. name: appsec-review-cleaner
   6. namespace: appsec-review
   7. spec:
   8. schedule: **"\*/1 \* \* \* \*"**
   9. concurrencyPolicy: Forbid
   10. jobTemplate:

...output omitted...

* 1. Replace the CHANGE\_ME label in the jobTemplate definition with the spec definition from the project-cleaner.yaml pod manifest. Although the long image name might show across two lines, you must add it as one line.
  2. apiVersion: batch/v1
  3. kind: CronJob
  4. metadata:
  5. name: appsec-review-cleaner
  6. namespace: appsec-review
  7. spec:
  8. schedule: "\*/1 \* \* \* \*"
  9. concurrencyPolicy: Forbid
  10. jobTemplate:
  11. spec:
  12. template:
  13. spec:
  14. **restartPolicy: Never**
  15. **serviceAccountName: project-cleaner-sa**
  16. **containers:**
  17. **- name: project-cleaner**
  18. **image: registry.ocp4.example.com:8443/redhattraining/do280-project-cleaner:v1.1**
  19. **imagePullPolicy: Always**
  20. **env:**
  21. **- name: "PROJECT\_TAG"**
  22. **value: "appsec-review-cleaner"**
  23. **- name: "EXPIRATION\_SECONDS"**

**value: "10"**

* 1. Create the cron job.
  2. [student@workstation appsec-review]$ **oc apply -f cron-job.yaml**

cronjob.batch/appsec-review-cleaner created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Optionally, verify that the project cleaner executed correctly. Use the generate-projects.sh script from the lab directory to generate projects for deletion. Wait for the next job execution and print the logs from that job's pod.
   1. Run the generate-projects.sh script to create test projects that the project cleaner will delete the next time that it runs.
   2. [student@workstation appsec-review]$ **./generate-projects.sh**
   3. obsolete-appsec-review-1 created at 15:29:14
   4. obsolete-appsec-review-2 created at 15:29:15
   5. obsolete-appsec-review-3 created at 15:29:16
   6. namespace/obsolete-appsec-review-1 labeled
   7. namespace/obsolete-appsec-review-2 labeled
   8. namespace/obsolete-appsec-review-3 labeled
   9. Last appsec-review-cleaner label applied at 15:29:20

...output omitted...

* 1. List the pods in the appsec-review project until you see a pod with the Completed status that is later than the last label that the script applied.
  2. [student@workstation appsec-review]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE
  4. appsec-review-cleaner-27909204-g49gr 0/1 Completed 0 2m37s
  5. appsec-review-cleaner-27909205-q2f2t 0/1 Completed 0 97s

**appsec-review-cleaner-27909206-xcswb** 0/1 **Completed** 0 **37s**

* 1. Print the logs from the last completed job, to verify that it deleted the obsolete projects.
  2. [student@workstation appsec-review]$ **oc logs pod/appsec-review-cleaner-*27909206-xcswb***
  3. ...output omitted...
  4. Namespace 'obsolete-appsec-review-1' deleted
  5. Namespace 'obsolete-appsec-review-2' deleted

Namespace 'obsolete-appsec-review-3' deleted

* 1. Change to the home directory to prepare for the next exercise.

[student@workstation appsec-review]$ **cd**

1. [Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade appsec-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish appsec-review**

# Chapter 9.  OpenShift Updates

[The Cluster Update Process](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09)

[Quiz: The Cluster Update Process](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s02)

[Detect Deprecated Kubernetes API Usage](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s03)

[Quiz: Detect Deprecated Kubernetes API Usage](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s04)

[Update Operators with the OLM](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s05)

[Quiz: Update Operators with the OLM](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s06)

[Quiz: OpenShift Updates](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s07)

[Summary](https://rol.redhat.com/rol/app/courses/do280-4.14/pages/ch09s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Update an OpenShift cluster and minimize disruption to deployed applications. |
| **Objectives** | * Describe the cluster update process. * Identify applications that use deprecated Kubernetes APIs. * Update OLM-managed operators by using the web console and CLI. |
| **Sections** | * The Cluster Update Process (and Quiz) * Detect Deprecated Kubernetes API Usage (and Quiz) * Update Operators with the OLM (and Quiz) * OpenShift Updates (Quiz) |

## **The Cluster Update Process**

### **Objectives**

* Describe the cluster update process.

### **Introducing Cluster Updates**

Red Hat OpenShift Container Platform 4 adds many new features by using Red Hat Enterprise Linux CoreOS. Red Hat released a new software distribution system that provides the upgrade path to update your cluster and the underlying operating system. With this new distribution system, OpenShift clusters can perform Over-the-Air updates (OTA).

This software distribution system for OTA manages the controller manifests, cluster roles, and any other resources to update a cluster to a particular version. With this feature, a cluster can run the 4.14.x version seamlessly. With OTA, a cluster can use new features as they become available, including the latest bug fixes and security patches. OTA substantially decreases downtime due to upgrades.

Red Hat hosts and manages this service at <https://console.redhat.com/openshift>, and hosts cluster images at [https://quay.io](https://quay.io/). You use a single interface to manage the lifecycle of all your OpenShift clusters. With OTA, you can update faster by skipping intermediate versions. For example, you can update from 4.14.1 to 4.14.3, and thus bypass 4.14.2.

### IMPORTANT

Starting with OpenShift 4.10, the OTA system requires a persistent connection to the internet. For more information about how to update disconnected clusters, consult the Updating a Restricted Network Cluster chapter in the references section.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.1: Managing clusters at cloud.redhat.com

The service defines upgrade paths that correspond to cluster eligibility for certain updates. Upgrade paths belong to update channels. Consider a channel as a representation of the upgrade path. The channel controls the frequency and stability of updates. The OTA policy engine represents channels as a series of pointers to particular versions within the upgrade path.

A channel name consists of the following parts: the tier (release candidate, fast, stable, and extended update support), the major version (4), and the minor version (.12). Example channel names include: candidate-4.14, fast-4.14, stable-4.14, and eus-4.14. Each channel delivers patches for a given cluster version.

#### **The Candidate Channel**

The candidate channel delivers updates for testing feature acceptance in the next version of OpenShift Container Platform. The release candidate versions are subject to further checks, and are promoted to the fast or stable channels when they meet the quality standards.

### IMPORTANT

Red Hat does not support the updates that are listed only in the candidate channel.

#### **The Fast Channel**

The fast channel delivers updates as soon as Red Hat declares the given version as a general availability release. Red Hat supports the updates that are released in this channel, and it is best suited to development and QA environments.

### NOTE

Customers can help to improve OpenShift by joining the Red Hat connected customers program. If you join this program, then your cluster is registered to the fast channel.

#### **The Stable Channel**

Red Hat support and site reliability engineering (SRE) teams monitor operational clusters with the updates from the fast channel. If operational clusters pass additional testing and validation, then updates in the fast channel are enabled in the stable channel. Red Hat supports the updates that are released in this channel, and it is best suited to production environments.

If Red Hat observes operational issues from a fast channel update, then that update is skipped in the stable channel. The stable channel delay provides time to observe any unforeseen problems in OpenShift clusters that testing did not reveal.

#### **The Extended Update Support Channel**

Starting with OpenShift Container Platform 4.8, Red Hat denotes all even-numbered minor releases (for example, 4.8, 4.10, 4.12, and 4.14) as Extended Update Support (EUS) releases.

EUS releases have no difference between stable-4.*x* and eus-4.*x* channels (where *x* denotes the even-numbered minor release) until OpenShift Container Platform moves to the EUS phase. You can switch to the EUS channel as soon as it becomes available.

#### **Support Status for Update Channels**

Red Hat offers support for all released updates in the fast, stable, and eus update channels. Red Hat supports the released updates in the candidate channel only if they are also listed in the fast or stable channels.

| **Update channel** | **Support status** |
| --- | --- |
| candidate-4.*x* | Supported if the update is also listed in the fast or stable channels. |
| fast-4.*x* | Supported |
| stable-4.*x* | Supported |
| eus-4.*x* | Supported |

### NOTE

The *x* in the channel name denotes the minor version.

### **Upgrade Paths**

You can apply each of the upgrade channels to a Red Hat OpenShift Container Platform version 4.14 cluster in different environments. The following paragraphs describe an example scenario where the 4.14.3 version has a defect.

**Stable channel**

When using the stable-4.14 channel, you can upgrade your cluster from 4.14.0 to 4.14.1 or to 4.14.2. If an issue is discovered in the 4.14.3 release, then you cannot upgrade to that version. When a patch becomes available in the 4.14.4 release, you can update your cluster to that version.

This channel is suited to production environments, because the Red Hat SRE teams and support services test the releases in that channel.

**Fast channel**

The fast-4.14 channel can deliver 4.14.1 and 4.14.2 updates but not 4.14.3. Red Hat also supports this channel, and you can apply it to development, QA, or production environments.

Administrators must specifically choose a different minor version channel, such as fast-4.14, to upgrade to a new release in a new minor version when it becomes available.

**Candidate channel**

You can use the candidate-4.14 channel to install the latest features of OpenShift. With this channel, you can upgrade to all z-stream releases, such as 4.14.1, 4.14.2, and 4.14.3.

You use this channel to access the latest features of the product as they get released. This channel is suited to development and pre-production environments.

**EUS channel**

When switching to the eus-4.14 channel, the stable-4.14 channel does not receive z-stream updates until the next EUS version becomes available.

### NOTE

Starting with OpenShift Container Platform 4.8, Red Hat denotes all even-numbered minor releases as Extended Update Support (EUS) releases.

The following graphic describes the update graphs for the stable and candidate channels:

|  |
| --- |
|  |

Figure 9.2: Update graphs for stable and candidate channels

Red Hat provides support for the General Availability (GA) updates that are released in the stable and fast channels. Red Hat does not support updates that are listed only in the candidate channel.

To ensure the stability of the cluster and the proper level of support, switch only from a stable channel to a fast channel. Although it is possible to switch from a stable channel or a fast channel to a candidate channel, it is not recommended. The candidate channel is best suited to testing feature acceptance and to assist in qualifying the next version of OpenShift Container Platform.

### NOTE

The release of updates for patch and security fixes ranges from several hours to a day. This delay provides time to assess any operational impacts to OpenShift clusters.

### **Changing the Update Channel**

You can change the update channel to eus-4.14, stable-4.14, fast-4.14, or candidate-4.14 by using the web console or the OpenShift CLI client:

**Web console**

Navigate to the **Administration** → **Cluster Settings** page on the details tab, and then click the pencil icon.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.3: Current update channel in the web console

A window displays options to select an update channel.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.4: Changing the update channel in the web console

**Command line**

Execute the following command to switch to another update channel by using the oc client. You can also switch to another update channel, such as stable-4.14, to update to the next minor version of OpenShift Container Platform.

[user@host ~]$ **oc patch clusterversion version --type="merge" \**

**--patch '{"spec":{"channel":"fast-4.14"}}'**

clusterversion.config.openshift.io/version patched

### **Pausing the Machine Health Check Resource**

During the upgrade process, nodes in the cluster might become temporarily unavailable. In the case of worker nodes, the machine health check might identify such nodes as unhealthy and reboot them. To avoid rebooting such nodes, pause all the machine health check resources before updating the cluster.

### NOTE

The prerequisite to pause the machine health check resources is not required on single-node installations.

Run the following command to list all the available machine health check resources.

[user@host ~]$ **oc get machinehealthcheck -n openshift-machine-api**

NAME MAXUNHEALTHY EXPECTEDMACHINES CURRENTHEALTHY

**machine-api-termination-handler** 100%

Add the cluster.x-k8s.io/paused annotation to the machine health check resource to pause it before updating the cluster.

[user@host ~]$ **oc annotate machinehealthcheck -n openshift-machine-api \**

**machine-api-termination-handler cluster.x-k8s.io/paused=""**

machinehealthcheck.machine.openshift.io/machine-api-termination-handler annotated

Remove the annotation after the cluster is updated.

[user@host ~]$ **oc annotate machinehealthcheck -n openshift-machine-api \**

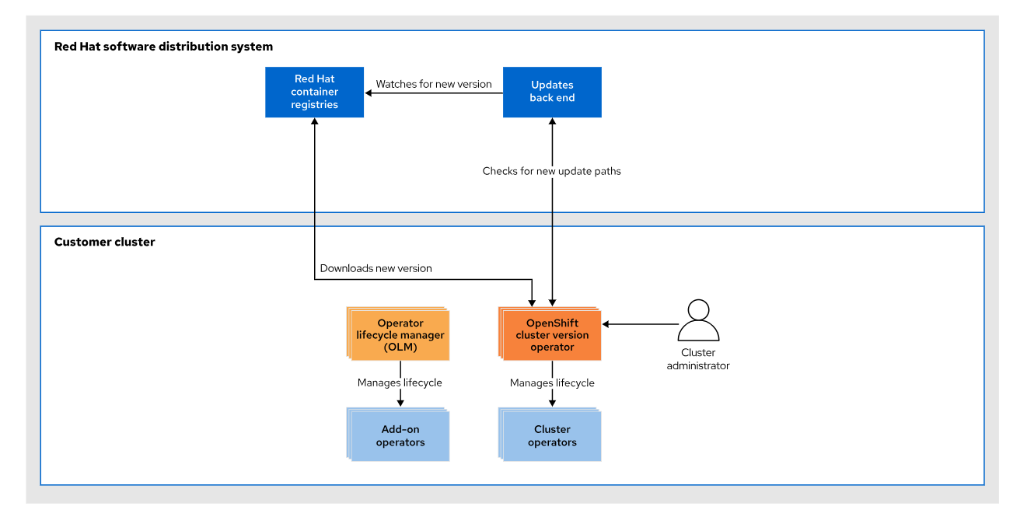
**machine-api-termination-handler cluster.x-k8s.io/paused-**

machinehealthcheck.machine.openshift.io/machine-api-termination-handler annotated

### **Over-the-air Updates**

OTA follows a client-server approach. Red Hat hosts the cluster images and the update infrastructure. OTA generates all possible update paths for your cluster. OTA also gathers information about the cluster and your entitlement to determine the available upgrade paths. The web console sends a notification when a new update is available.

The following diagram describes the updates architecture: Red Hat hosts both the cluster images and a "watcher", which automatically detects new images that are pushed to Quay. The Cluster Version Operator (CVO) receives its update status from that watcher. The CVO starts by updating the cluster components via their operators, and then updates any extra components that the Operator Lifecycle Manager (OLM) manages.



|  |
| --- |
|  |

Figure 9.5: OpenShift Container Platform updates architecture

With telemetry, Red Hat can determine the update path. The cluster uses a Prometheus-based Telemeter component to report on the state of each cluster operator. The data is anonymized and sent back to Red Hat servers that advise cluster administrators about potential new releases.

### NOTE

Red Hat values customer privacy. For a complete list of the data that Telemeter gathers, consult the Data Collection and Telemeter Sample Metrics documents in the references section.

In the future, Red Hat intends to extend the list of updated operators that are included in the upgrade path to include independent software vendor (ISV) operators.

A diagram of a company

Description automatically generated

|  |
| --- |
|  |

Figure 9.6: Managing cluster updates by using telemetry

### **The Update Process**

The following components are involved in the cluster update process:

**Machine Config Operator**

The Machine Config Operator applies the desired machine state to each of the nodes. This component also handles the rolling upgrade of nodes in the cluster, and uses CoreOS Ignition as the configuration format.

**Operator Lifecycle Manager**

The OLM orchestrates updates to any operators that are running in the cluster.

### **Updating the Cluster**

You can update the cluster via the web console or from the command line. The **Administration** → **Cluster Settings** page displays an update status of Available updates when a new update is available. From this page, click **Select a version**, and then select the version and the cluster update option that you want to install:

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.7: Update the cluster by using the web console

### IMPORTANT

Rolling back your cluster to an earlier version is not supported. **If your update is failing to complete, contact Red Hat support.**

The update process also updates the underlying operating system when updates are available. The updates use the rpm-ostree technology for managing transactional upgrades. Updates are delivered via container images and are part of the OpenShift update process. When the update deploys, the nodes pull the new image, extract it, write the packages to the disk, and then modify the bootloader to boot into the new version. The machine reboots and implements a rolling update to ensure that the cluster capacity is minimally impacted.

#### **Update the Cluster by Using the Command Line**

The following steps describe the procedure for updating a cluster as a cluster administrator by using the command-line interface:

* Be sure to update all operators that are installed through the OLM to the 4.14 version before updating the OpenShift cluster.
* Retrieve the cluster version and review the current update channel information. If you are running the cluster in production, then ensure that the channel reads stable.
* [user@host ~]$ **oc get clusterversion**
* NAME VERSION AVAILABLE PROGRESSING SINCE STATUS
* version **4.14.0** True False 2d Cluster version is **4.14.0**
* [user@host ~]$ **oc get clusterversion -o jsonpath='{.items[0].spec.channel}{"\n"}'**

stable-4.14

* View the available updates and note the version number of the update to apply.
* [user@host ~]$ **oc adm upgrade**
* Cluster version is **4.14.0**
* Upstream is unset, so the cluster will use an appropriate default.
* Channel: stable-4.14 (available channels: candidate-4.14, candidate-4.15, eus-4.14, fast-4.14, stable-4.14)
* Recommended updates:
* VERSION IMAGE
* **4.14.10** quay.io/openshift-release-dev/ocp-release@sha256:...

...output omitted...

* Apply the latest update to your cluster, or update to a specific version:
  + Run the following command to install the latest available update for your cluster.

[user@host ~]$ **oc adm upgrade --to-latest=true**

* + Run the following command to install a specific version. VERSION corresponds to one of the available versions that the oc adm upgrade command returns.

[user@host ~]$ **oc adm upgrade --to=*VERSION***

* The previous command initializes the update process. Run the following command to review the status of the Cluster Version Operator (CVO) and the installed cluster operators.
* [user@host ~]$ **oc get clusterversion**
* NAME VERSION AVAILABLE PROGRESSING SINCE STATUS
* version 4.14.0 True **True** 1m Working towards **4.14.10** ...
* [user@host ~]$ **oc get clusteroperators**
* NAME VERSION AVAILABLE PROGRESSING DEGRADED ...
* authentication 4.14.0 True False False ...
* baremetal **4.14.10** **False** True False ...
* cloud-controller-manager **4.14.10** True False **True** ...

...output omitted...

* Use the following command to review the cluster version history and monitor the status of the update. It might take some time for all the objects to finish updating.

The history contains a list of the most recent versions that were applied to the cluster. This list is updated when the CVO applies an update. The list is ordered by date, where the newest update is first in the list.

If the rollout completed successfully, then updates in the history have a Completed state. Otherwise, the update has a Partial state if it failed or did not complete.

[user@host ~]$ **oc describe clusterversion**

...output omitted...

History:

Completion Time: 2024-02-10T04:38:12Z

Image: quay.io/openshift-release-dev/ocp-release@sha256:...

Started Time: 2024-02-10T03:35:05Z

State: **Partial**

Verified: true

Version: **4.14.10**

Completion Time: 2024-02-10T12:39:02Z

Image: quay.io/openshift-release-dev/ocp-release@sha256:...

Started Time: 2024-02-10T12:23:14Z

State: **Completed**

Verified: false

Version: **4.14.10**

### IMPORTANT

When an update is failing to complete, the Cluster Version Operator (CVO) reports the status of any blocking components and attempts to reconcile the update.

Rolling back your cluster to a previous version is not supported. **If your update is failing to complete, contact Red Hat support.**

* After the process completes, you can confirm that the cluster is updated to the new version.
* [user@host ~]$ **oc get clusterversion**
* NAME VERSION AVAILABLE PROGRESSING SINCE STATUS

version **4.14.10** True False 30m Cluster version is **4.14.10**

### REFERENCES

For more information about update channels, update prerequisites, and updating clusters in disconnected environments, refer to the Updating a Restricted Network Cluster and Updating a Cluster Between Minor Versions chapters in the Red Hat OpenShift Container Platform 4.14 Updating Clusters documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-restricted-network-cluster>

For more information about updating operators that are installed through the Operator Lifecycle Manager, refer to the Upgrading Installed Operators section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Working with Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-upgrading-operators>

For more information about performing an EUS-to-EUS update, refer to the Preparing to Perform an EUS-to-EUS Update chapter in the Red Hat OpenShift Container Platform 4.14 Updating Clusters documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-eus-to-eus-upgrade_eus-to-eus-upgrade>

For more information about the OpenShift Container Platform upgrade paths, visit the following page in the customer portal: <https://access.redhat.com/solutions/4583231>

For more information about the OpenShift Container Platform update graph, visit the following page in the customer portal: <https://access.redhat.com/labs/ocpupgradegraph/update_path>

For more information about OpenShift Extended Update Support (EUS), visit the following page in the customer portal: <https://access.redhat.com/support/policy/updates/openshift-eus>

For more information about the OpenShift Container Platform lifecycle policy, visit the following page in the customer portal: <https://access.redhat.com/support/policy/updates/openshift>

[OpenShift 4 Data Collection](https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/data-collection.md)

[OpenShift 4 Telemeter Sample Metrics](https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/sample-metrics.md)

## **Detect Deprecated Kubernetes API Usage**

### **Objectives**

* Identify applications that use deprecated Kubernetes APIs.

### **OpenShift Versions**

Kubernetes is an open source container orchestration engine for automating the deployment, scaling, and management of containerized applications. The OpenShift Container Platform foundation is based on Kubernetes and therefore shares the underlying technology. The following table lists the OpenShift version and the Kubernetes version that it is based on:

| **OpenShift version** | **Kubernetes version** |
| --- | --- |
| 4.12 | 1.25 |
| 4.13 | 1.26 |
| 4.14 | 1.27 |

### **Kubernetes API Deprecation Policy**

The Kubernetes API versions are categorized based on feature maturity (experimental, pre-release, and stable).

| **API version** | **Category** | **Description** |
| --- | --- | --- |
| v1alpha1 | Alpha | Experimental features |
| v1beta1 | Beta | Pre-release features |
| v1 | Stable | Stable features, generally available |

Use the following command to view the current version of a resource:

[user@host ~]$ **oc api-resources | egrep '^NAME|cronjobs'**

NAME SHORTNAMES APIVERSION NAMESPACED KIND

cronjobs cj batch/v1 true CronJob

When a stable version of a feature is released, the beta versions are marked as deprecated and are removed after three Kubernetes releases. If a request uses a deprecated API version, then the API server returns a deprecation warning that includes the name of the current version of the cluster.

[user@host ~]$ **egrep 'kind|apiVersion' cronjob-beta.yaml**

kind: **CronJob**

apiVersion: **batch/v1beta1**

[user@host ~]$ **oc create -f cronjob-beta.yaml**

**Warning**: **batch/v1beta1 CronJob** is deprecated in v1.21+, unavailable in v1.25+; use **batch/v1 CronJob**

cronjob.batch/hello created

If a request uses an API version that Kubernetes removed, then the API server returns an error, because that API version is not supported in the cluster.

[user@host ~]$ **egrep 'kind|apiVersion' cronjob-alpha.yaml**

apiVersion: **batch/v1alpha1**

kind: **CronJob**

[user@host ~]$ **oc create -f cronjob-alpha.yaml**

**error**: resource mapping not found for name: "hello" namespace: "" from "cronjob-alpha.yaml": no matches for kind "**CronJob**" in version "**batch/v1beta1**"

**ensure CRDs are installed first**

#### **Deprecated and Removed Features in Kubernetes**

The Kubernetes 1.27 release stopped serving some API versions that were marked as deprecated in previous releases. The following table contains a short list of the deprecated and removed API versions.

| **Resource** | **Removed API Group** | **Current API Group** |
| --- | --- | --- |
| CSIStorageCapacity | storage.k8s.io/​v1beta1 | storage.k8s.io/​v1 |

### NOTE

For more information about the API versions that are deprecated and removed in Kubernetes, consult Kubernetes Deprecated API Migration Guide in the references section.

#### **Identifying Deprecated APIs**

You can identify from the API request count whether a workload uses a deprecated API version. The API request count output contains four columns. A value in the REMOVEDINRELEASE column indicates that the API version is deprecated and specifies the Kubernetes version that will remove it.

[user@host ~]$ **oc get apirequestcounts | awk '{if(NF==4){print $0}}'**

NAME

**REMOVEDINRELEASE** REQUESTSINCURRENTHOUR REQUESTSINLAST24H

...output omitted...

cronjobs.v1beta1.batch

**1.25** 15 44

horizontalpodautoscalers.v2beta2.autoscaling

**1.26** 6 30

podsecuritypolicies.v1beta1.policy

**1.25** 28 77

...output omitted...

If the REMOVEDINRELEASE column is blank, then it indicates that the current API version is not deprecated, and that version will be kept in future releases.

### NOTE

You can use a JSONPath filter to retrieve the results. The FILTER variable is written on a single line.

[user@host ~]$ **FILTER='{range .items[?(@.status.removedInRelease!="")]}{.status.removedInRelease}{"\t"}{.status.requestCount}{"\t"}{.metadata.name}{"\n"}{end}'**

[user@host ~]$ **oc get apirequestcounts -o jsonpath="${FILTER}" | \**

**column -t -N "RemovedInRelease,RequestCount,Name"**

RemovedInRelease RequestCount Name

1.25 44 cronjobs.v1beta1.batch

...output omitted...

If the command does not retrieve any information, then it indicates that none of the installed APIs are deprecated.

You can use a JSONPath filter for a list of actions for that resource and who did them.

[user@host ~]$ **FILTER='{range .status.currentHour..byUser[\*]}{..byVerb[\*].verb}{","}{.username}{","}{.userAgent}{"\n"}{end}'**

[user@host ~]$ **TYPE=apirequestcount.apiserver.openshift.io/cronjobs.v1.batch**

[user@host ~]$ **echo ${TYPE} ; oc get ${TYPE} -o jsonpath="${FILTER}" | \**

**column -t -s ',' -N "Verbs,Username,UserAgent"**

apirequestcount.apiserver.openshift.io/cronjobs.v1.batch

Verbs Username UserAgent

get update system:serviceaccount:kube-system:cronj... kube-controller-manager/v1...

watch system:kube-controller-manager kube-controller-manager/v1...

...output omitted...

### **Deprecated and Removed Features in OpenShift**

Red Hat OpenShift Container Platform (RHOCP) is a set of modular components and services that are built on top of a Kubernetes container infrastructure.

Some features that were available in previous OpenShift releases are deprecated or removed. A deprecated feature is not recommended for new deployments, because a future release will remove it. The following table contains a short list of the deprecated and removed features in OpenShift.

| **OpenShift 4.12** | **OpenShift 4.13** | **OpenShift 4.14** | **Feature** |
| --- | --- | --- | --- |
| General Availability | General Availability | Deprecated | Operator lifecycle and development deprecated |
| Deprecated | Deprecated | Deprecated | CoreDNS wildcard queries for the cluster.local domain |
| Deprecated | Deprecated | Deprecated | Persistent storage that uses FlexVolume |
| Not Available | General Availability | **Removed** | --include-local-oci-catalogs parameter for oc-mirror |
| General Availability | General Availability | **Deprecated** | DeploymentConfig objects |

### NOTE

For more information about the deprecated and removed API versions in Kubernetes, consult the OpenShift Container Platform 4.14 release notes in the references section.

#### **Deprecated API Alerts in OpenShift**

OpenShift includes two alerts that are triggered when a workload uses a deprecated API version:

**APIRemovedInNextReleaseInUse**

This alert is triggered for APIs that OpenShift Container Platform will remove in the next release.

**APIRemovedInNextEUSReleaseInUse**

This alert is triggered for APIs that OpenShift Container Platform Extended Update Support (EUS) will remove in the next release.

The alert describes the situation with context to identify the affected workload.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.8: Deprecated API alert

You can extract the alerts in JSON format from the Prometheus stateful set, and then filter the result to retrieve the deprecated API alerts.

[user@host ~]$ **oc exec -it statefulset/prometheus-k8s -c prometheus \**

**-n openshift-monitoring -- \**

**curl -fsSL 'http://localhost:9090/api/v1/alerts' | jq . > alerts.json**

[user@host ~]$ **jq '[.data.alerts[] |**

**select(.labels.alertname=="APIRemovedInNextReleaseInUse" or**

**.labels.alertname=="APIRemovedInNextEUSReleaseInUse")]' < alerts.json**

[

{

"labels": {

"alertname": **"APIRemovedInNextReleaseInUse"**,

...output omitted...

},

"state": "firing",

...output omitted...

},

{

"labels": {

"alertname": **"APIRemovedInNextEUSReleaseInUse"**,

...output omitted...

},

"state": "firing",

...output omitted...

}

]

### NOTE

If the output of the jq command is an empty JSON array [], then the alerts were not reported.

#### **Explicit Acknowledgment Before Cluster Updates**

OpenShift Container Platform 4.14 uses Kubernetes 1.27, which removed deprecated v1beta1 APIs.

OpenShift Container Platform requires an administrator to provide a manual acknowledgment before the cluster can be upgraded from version 4.13 to 4.14. This requirement helps to prevent issues after upgrading to OpenShift Container Platform 4.14, where workloads, tools, or other components that run on or interact with the cluster still use removed APIs.

Administrators must evaluate their cluster for workloads that use removed APIs, and migrate the affected components to the appropriate new API version. After migration, the administrator can provide an acknowledgment.

[user@host ~]$ **oc patch configmap admin-acks -n openshift-config --type=merge \**

**--patch '{"data":{"ack-4.13-kube-1.27-api-removals-in-4.14":"true"}}'**

configmap/admin-acks patched

### REFERENCES

For more information about the removed features in OpenShift, refer to the Deprecated and Removed Features section in the Red Hat OpenShift Container Platform 4.14 release notes at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/release_notes/index#ocp-4-14-deprecated-removed-features>

For more information about what version of the Kubernetes API is included with each OpenShift 4.x release, visit the following page in the customer portal: <https://access.redhat.com/solutions/4870701>

For more information about the Kubernetes API deprecations and removals, visit the following page in the customer portal: <https://access.redhat.com/articles/6955985>

For more information about the deprecated APIs in OpenShift Container Platform 4.14, visit the following page in the customer portal: <https://access.redhat.com/articles/6955381>

For more information about how to get fired alerts on OpenShift by using the command-line, visit the following page in the customer portal: <https://access.redhat.com/solutions/4250221>

[What's New in Red Hat OpenShift 4.14](https://www.redhat.com/en/whats-new-red-hat-openshift)

[Preparing to Update to OpenShift Container Platform 4.14](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-cluster-prepare)

[Kubernetes Deprecation Policy](https://kubernetes.io/docs/reference/using-api/deprecation-policy/)

[Kubernetes Deprecated API Migration Guide](https://kubernetes.io/docs/reference/using-api/deprecation-guide/)

[Kubernetes Removals and Deprecations in 1.27](https://kubernetes.io/blog/2023/03/17/upcoming-changes-in-kubernetes-v1-27/)

[Kubernetes 1.27 release announcement](https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/)

## **Update Operators with the OLM**

### **Objectives**

* Update OLM-managed operators by using the web console and CLI.

### **Operator Updates**

For operators that are installed in an OpenShift cluster, operator providers can release new versions. These new versions can contain bug fixes and new features. The Operator Lifecycle Manager (OLM) can update these operators.

However, new operator versions can introduce bugs and incompatibilities.

Cluster administrators should define operator update policies to ensure that bug fixes and new functions are adopted, with the cluster continuing to operate correctly.

OpenShift provides features to help to implement such policies.

* For each installed operator, you can decide whether the OLM automatically applies updates, or whether the updates require administrator approval.
* Operator providers can create multiple channels for an operator. The provider can follow different policies to push updates to each channel, so that each channel contains different versions of the operator. When installing an operator, you choose the channel to follow for updates.
* You can create custom catalogs, and decide which versions of operators to include in the catalog. For example, in a multicluster environment, you configure operators to update automatically, but add only tested versions to the catalog.

Providers can publish operators by other means than the OLM and operator catalogs. For example, a provider can publish operators as Helm charts or YAML resource files. The OLM does not manage operators that are installed by other means.

### **Operator Update Channels**

Each operator provider can create multiple channels for an operator.

For example, a provider can create stable and preview channels for an operator. The provider publishes each new version of the operator to the preview channel. You can use the preview channel to test new features and to validate that the new versions fix bugs. If the provider receives feedback for preview versions of the operator and finds no serious issues with the latest version, then the provider publishes the version to the stable channel. You can use the stable channel for environments with higher reliability requirements, and trade off slower adoption of new features for improved stability.

Additionally, operators might have new features that introduce significant changes or incompatibilities with earlier versions. Operator providers might adopt a versioning scheme for the operator that separates major updates from minor updates, depending on the adoption cost of the new version. In this scenario, providers can create channels for different major versions of the operator.

For example, a provider creates an operator that installs an application. The provider creates version-1 and version-2 channels, to correspond to different major versions of the application. Users of the operator can stay on the version-1 channel in the production environment, and test and design an update process to adopt the version-2 channel in a staging environment.

When you install an operator, determine the most suitable channel for your requirements. Clusters with varying reliability requirements might use different channels.

You can edit an operator subscription to switch channels. Switching channels does not cause any operator update, unless switching channel makes a later version available and the operator is configured for automatic updates. Switching channels might cause unwanted results; always refer to the operator documentation to learn about possible issues.

### **Automatic and Manual Updates**

When you install an operator, you can decide whether the OLM automatically applies updates, or whether the OLM requires an administrator to approve the update. On the operator installation wizard, you can choose between automatic or manual approval. When you create a subscription by using the oc command, the resource specification contains an installPlanApproval property that requires an Automatic or Manual value.

If the publishing policies of an operator suit your requirements, then you can configure automatic approvals. Click **Operators** → **Installed Operators** on the web console, or examine cluster service versions with the oc command, to review the version of installed operators.

If you install an operator and configure manual approvals, then you must approve updates before the OLM updates the operator.

The **Installed Operators** page in the web console displays available upgrades.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.9: The Installed Operators page with an available upgrade

The subscription resources and the install plan resources contain information about upgrades. You can use the oc command to examine those resources to find available upgrades.

[user@host ~]$ **oc get sub -n openshift-operators web-terminal -o yaml**

...output omitted...

spec:

channel: fast

installPlanApproval: Manual

name: web-terminal

source: do280-catalog-redhat

sourceNamespace: openshift-marketplace

startingCSV: web-terminal.v1.5.1

status:

...output omitted...

conditions:

...output omitted...

- lastTransitionTime: "2022-11-24T13:46:21Z"

reason: RequiresApproval

status: "True"

type: InstallPlanPending

currentCSV: web-terminal.v1.6.0

installPlanGeneration: 2

installPlanRef:

apiVersion: operators.coreos.com/v1alpha1

kind: InstallPlan

name: install-72vnw

namespace: openshift-operators

resourceVersion: "194989"

uid: 8dc979fe-936f-475a-8977-36d210c4da98

installedCSV: web-terminal.v1.5.1

...output omitted...

state: UpgradePending

|  |  |
| --- | --- |
|  | The currentCSV key shows the latest available version in the channel. |
|  | The installPlanRef section contains a reference to the install plan resource. |
|  | The installedCSV key shows the current version. |

The OLM also creates an install plan resource when the operator channel contains a later version of an operator.

[user@host ~]$ **oc get installplan -n openshift-operators install-72vnw -o yaml**

apiVersion: operators.coreos.com/v1alpha1

kind: InstallPlan

...output omitted...

spec:

approval: Manual

approved: false

clusterServiceVersionNames:

- web-terminal.v1.6.0

generation: 2

status:

...output omitted...

phase: RequiresApproval

...output omitted...

|  |  |
| --- | --- |
|  | The approval key indicates whether updates must be approved. |
|  | The approved key shows whether the update is approved. |
|  | The clusterServiceVersionNames shows the updated version. |

To install the update, edit the specification of the install plan to change the approved key value to true.

[user@host ~]$ **oc patch installplan install-72vnw --type merge \**

**--patch '{"spec":{"approved":true}}'**

installplan.operators.coreos.com/install-72vnw patched

You can also use the web console to approve an update. In the **Installed Operators**, click **Upgrade available**, and then click **Preview InstallPlan** to view the install plan. Review the install plan, and then click **Approve** to update the operator.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.10: Reviewing an install plan

### **Operator Updates and Cluster Updates**

Operators might be incompatible with later versions of OpenShift. For example, an operator that uses an API that is removed from later versions of OpenShift does not work correctly when the cluster is updated. Operators can define a list of compatible OpenShift versions.

When updating a cluster, you might need to update operators if the installed version of the operator is not compatible with the updated OpenShift version. Before you update a cluster, review and install any operator updates that are needed for compatibility fixes. If no compatible updates are available, then you must update the cluster by uninstalling incompatible operators.

#### **Uninstalling Operators**

You can uninstall operators by using the web console or the oc command.

In the console, click **Operators** → **Installed operators** and locate the operator. Click the vertical ellipsis (**⋮**) menu, and then click **Uninstall Operator**.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 9.11: The uninstall operator button

After confirming the operation by clicking **Uninstall**, the OLM uninstalls the operator.

Alternatively, delete the subscription and cluster service versions by using the oc command.

### IMPORTANT

Uninstalling an operator can leave operator resources on the cluster. Always review the operator documentation to learn about cleanup processes that you must follow to completely remove an operator.

### REFERENCES

Refer to the Upgrading Installed Operators section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-upgrading-operators>

Refer to the Deleting Operators from a Cluster section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-deleting-operators-from-a-cluster>

For more information about creating custom catalogs with controlled operator versions, refer to the Managing Custom Catalogs section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-managing-custom-catalogs>

## **Lab: Cluster Self-service Setup**

Configure a cluster with default settings for self-service projects.

**Outcomes**

* Create a project template that sets quotas, ranges, and network policies.
* Restrict access to the self-provisioners cluster role.
* Create groups and assign users to groups.
* Use role-based access control (RBAC) to grant permissions to groups.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start compreview-review**

The lab command copies the exercise files to the ~/DO280 directory and creates the following users:

* do280-support
* do280-platform
* do280-presenter
* do280-attendee

The goal, as the cluster administrator, is to configure a dedicated cluster to host workshops on different topics.

Each workshop requires a project, so that workshops are isolated from each other.

You must set up the cluster so that when the presenter creates a workshop project, the project gets a base configuration.

The presenter must be mostly self-sufficient to administer a workshop with little help from the workshop support team.

The workshop support team must deploy applications that administer workshops and that enhance the workshop experience. You set up a project and the applications for this purpose on a second lab.

**Specifications**

Use the following values to access the OpenShift cluster:

| **Item** | **Value** |
| --- | --- |
| Dev user/password | developer/developer |
| Admin user/password | admin/redhatocp |
| API URL | https://api.ocp4.example.com:6443 |

The following workshop groups are required:

* Create the groups with the specified users in the following table:

| **Group** | **User** |
| --- | --- |
| platform | do280-platform |
| presenters | do280-presenter |
| workshop-support | do280-support |

* The lab start command creates the users with the redhat password.
* The platform group administers the cluster.
* The presenters group consists of the people who deliver the workshops.
* The workshop-support group maintains the needed applications to support the workshops and the workshop presenters.
* Ensure that only users from the following groups can create projects:

| **Group** |
| --- |
| platform |
| presenters |
| workshop-support |

* An attendee must not be able to create projects. Because this exercise requires steps that restart the Kubernetes API server, this configuration must persist across API server restarts.
* The workshop-support group requires the following roles in the cluster:
  + The admin role to administer projects
  + A custom role that is provided in the groups-role.yaml file You must create this custom role to enable support members to create workshop groups and to add workshop attendees.
* The platform group must be able to administer the cluster without restrictions.
* The workshop-support group must perform the following tasks for the workshop project:
  + Create a workshop-specific attendees group.
  + Assign the edit role to the attendees group.
  + Add users to the attendees group.
* Each workshop must be hosted in an independent project.
* All the resources that the cluster creates with a new workshop project must use workshop as the name for grading purposes.
* Each workshop must enforce the following maximum constraints:
  + The project uses up to 2 CPUs.
  + The project uses up to 1 Gi of RAM.
  + The project requests up to 1.5 CPUs.
  + The project requests up to 750 Mi of RAM.
* Each workshop must enforce constraints to prevent an attendee's workload from consuming all the allocated resources for the workshop:
  + A workload uses up to 750m CPUs.
  + A workload uses up to 750 Mi.
* Each workshop must have a resource specification for workloads:
  + A default limit of 500m CPUs.
  + A default limit of 500 Mi of RAM.
  + A default request of 0.1 CPUs.
  + A default request of 250 Mi of RAM.

You can use the templates that are provided in the quota.yaml, limitrange.yaml, and networkpolicy.yaml files.

* Each workshop project must have this additional default configuration:
  + A local binding for the presenter user to the admin cluster role with the workshop name
  + The workshop=*project\_name* label to help to identify the workshop workload
  + Must accept traffic only from within the same workshop or from the ingress controller.
* Use the registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 image, which listens on the 8080 port, to simulate a workshop workload.
* As the do280-presenter user, you must create a workshop with the do280 name.
* As the do280-support user, you must create the do280-attendees group with the do280-attendee user, and assign the edit cluster role to the do280-attendees group.

1. Change to the ~/DO280/labs/compreview-review directory and log in to the cluster as the admin user.
   1. Change to the lab directory.

[student@workstation ~]$ **cd ~/DO280/labs/compreview-review**

* 1. Open a terminal window and log in as the admin user with the redhatocp password.
  2. [student@workstation compreview-review]$ **oc login -u admin -p redhatocp \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

...output omitted...

1. Create the following groups and add a user as specified in the following table.

| **Group** | **User** |
| --- | --- |
| workshop-support | do280-support |
| presenters | do280-presenter |
| platform | do280-platform |

* 1. Create the workshop-support group.
  2. [student@workstation compreview-review]$ **oc adm groups new workshop-support**

group.user.openshift.io/workshop-support created

* 1. Add the do280-support user to the workshop-support group.
  2. [student@workstation compreview-review]$ **oc adm groups add-users \**
  3. **workshop-support do280-support**

group.user.openshift.io/workshop-support added: "do280-support"

* 1. Create the presenters group.
  2. [student@workstation compreview-review]$ **oc adm groups new presenters**

group.user.openshift.io/presenters created

* 1. Add the do280-presenter user to the presenters group.
  2. [student@workstation compreview-review]$ **oc adm groups add-users \**
  3. **presenters do280-presenter**

group.user.openshift.io/presenters added: "do280-presenter"

* 1. Create the platform group.
  2. [student@workstation compreview-review]$ **oc adm groups new platform**

group.user.openshift.io/platform created

* 1. Add the do280-platform user to the platform group.
  2. [student@workstation compreview-review]$ **oc adm groups add-users \**
  3. **platform do280-platform**

group.user.openshift.io/platform added: "do280-platform"

* 1. Use the oc get groups command to verify that the group configuration is correct.
  2. [student@workstation compreview-review]$ **oc get groups**
  3. NAME USERS
  4. ...output omitted...
  5. **platform do280-platform**
  6. **presenters do280-presenter**

**workshop-support do280-support**

1. Grant to the workshop-support group the admin and the custom manage-groups cluster roles. You must create the manage-groups custom cluster role from the groups⁠-⁠role⁠.⁠yaml file.
   1. Grant the admin cluster role to the workshop-support group.
   2. [student@workstation compreview-review]$ **oc adm policy \**
   3. **add-cluster-role-to-group admin workshop-support**

clusterrole.rbac.authorization.k8s.io/admin added: "workshop-support"

* 1. Run the oc create command to create the manage-groups cluster role in the groups-role.yaml file.
  2. [student@workstation compreview-review]$ **oc create -f groups-role.yaml**

clusterrole.rbac.authorization.k8s.io/manage-groups created

* 1. Grant the manage-groups cluster role to the workshop-support group.
  2. [student@workstation compreview-review]$ **oc adm policy \**
  3. **add-cluster-role-to-group manage-groups workshop-support**

clusterrole.rbac.authorization.k8s.io/manage-groups added: "workshop-support"

1. Create a cluster role binding to assign the cluster-admin cluster role to the platform group.
2. [student@workstation compreview-review]$ **oc adm policy \**
3. **add-cluster-role-to-group cluster-admin platform**

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "platform"

1. Allow only the platform, workshop-support and presenters groups to create projects, by editing the self-provisioner cluster role. Enforce that only users from these groups can create projects. Also, make this change permanent by setting the rbac.authorization.kubernetes.io/autoupdate annotation with the false value.
   1. Use the oc edit command to edit the self-provisioners cluster role binding.
   2. [student@workstation compreview-review]$ **oc edit clusterrolebinding \**

**self-provisioners**

Replace the subject of the role binding for the system:authenticated:oauth group with the platform, workshop-support, and presenters groups.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

annotations:

**rbac.authorization.kubernetes.io/autoupdate: "false"**

creationTimestamp: "2023-01-24T23:31:00Z"

name: self-provisioners

resourceVersion: "250330"

uid: a6053896-f68f-41ff-9bb3-5da579a701bc

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: self-provisioner

subjects:

**- apiGroup: rbac.authorization.k8s.io**

**kind: Group**

**name: platform**

**- apiGroup: rbac.authorization.k8s.io**

**kind: Group**

**name: workshop-support**

**- apiGroup: rbac.authorization.k8s.io**

**kind: Group**

**name: presenters**

* 1. As the do280-attendee user, verify that you cannot create a project.

Log in as the do280-attendee user with the redhat password.

[student@workstation compreview-review]$ **oc login -u do280-attendee -p redhat**

Login successful.

You don't have any projects. Contact your system administrator to request a project.

Use the oc new-project command to try to create a template-test project.

[student@workstation compreview-review]$ **oc new-project template-test**

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

1. As the admin user, create a template-test namespace to design the project template.
   1. Log in as the admin user with the redhatocp password.
   2. [student@workstation compreview-review]$ **oc login -u admin -p redhatocp**
   3. Login successful.

...output omitted...

* 1. Use the oc new-project command to create the template-test project.
  2. [student@workstation compreview-review]$ **oc new-project template-test**
  3. Now using project "template-test" on server...

...output omitted...

1. Create a template resource quota with the following specification.

| **Quota** | **Value** |
| --- | --- |
| limits.cpu | 2 |
| limits.memory | 1Gi |
| requests.cpu | 1500m |
| requests.memory | 750Mi |

* 1. Edit the quota.yaml file and replace the CHANGE\_ME label to match the following definition.
  2. apiVersion: v1
  3. kind: ResourceQuota
  4. metadata:
  5. name: workshop
  6. namespace: **template-test**
  7. spec:
  8. **hard:**
  9. **limits.cpu: 2**
  10. **limits.memory: 1Gi**
  11. **requests.cpu: 1500m**

**requests.memory: 750Mi**

* 1. Use the oc create command to create the quota in the template-test project.
  2. [student@workstation compreview-review]$ **oc create -f quota.yaml**

resourcequota/workshop created

1. Create the workshop limit range with the following specification.

| **Limit type** | **Value** |
| --- | --- |
| max.cpu | 750m |
| max.mem | 750Mi |
| default.cpu | 500m |
| default.memory | 500Mi |
| defaulRequest.cpu | 100m |
| defaulRequest.memory | 250Mi |

* 1. Edit the limitrange.yaml file and replace the CHANGE\_ME label to match the following definition.
  2. apiVersion: v1
  3. kind: LimitRange
  4. metadata:
  5. name: workshop
  6. namespace: **template-test**
  7. spec:
  8. **limits:**
  9. **- max:**
  10. **cpu: 750m**
  11. **memory: 750Mi**
  12. **default:**
  13. **cpu: 500m**
  14. **memory: 500Mi**
  15. **defaultRequest:**
  16. **cpu: 100m**
  17. **memory: 250Mi**

**type: Container**

* 1. Use the oc create command to create the limit range in the template-test project.
  2. [student@workstation compreview-review]$ **oc create -f limitrange.yaml**

limitrange/workshop created

1. Create a network policy to accept traffic from within the workshop project or from outside the cluster. To identify the workshop project traffic, label the template-test namespace with the workshop=template-test label.
   1. Use the oc create deployment command to create a deployment without resource specifications.
   2. [student@workstation compreview-review]$ **oc create deployment test-workload \**
   3. **--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0**

deployment.apps/test-workload created

* 1. Get the IP address of one of the NGINX pods.
  2. [student@workstation compreview-review]$ **oc get pod -o wide**
  3. NAME READY STATUS ... IP ...

test-workload-***56bf7dc6fc-mshn9*** 1/1 Running ... **10.8.0.138** ...

* 1. Use the oc debug command to run the curl command from a pod in the default project.

Use the curl command from the default namespace to query the NGINX server that runs in the test workload.

[student@workstation compreview-review]$ **oc debug --to-namespace="default" \**

**-- curl -s http://*10.8.0.138:8080***

Starting pod/image-debug ...

<html>

<body>

<h1>Hello, world from nginx!</h1>

</body>

</html>

Removing debug pod ...

* 1. Use the oc label command to add the label to the template-test namespace.
  2. [student@workstation compreview-review]$ **oc label ns template-test \**
  3. **workshop=template-test**

namespace/template-test labeled

* 1. Edit the network policy from the networkpolicy.yaml file. Replace the CHANGE\_ME labels according to the following specification.
  2. kind: NetworkPolicy
  3. apiVersion: networking.k8s.io/v1
  4. metadata:
  5. name: workshop
  6. namespace: template-test
  7. spec:
  8. podSelector: **{}**
  9. ingress:
  10. - from:
  11. - namespaceSelector:
  12. matchLabels:
  13. **workshop: template-test**
  14. - namespaceSelector:
  15. matchLabels:

**policy-group.network.openshift.io/ingress: ""**

* 1. Run the oc create command to create the policy in the template-test project.
  2. [student@workstation compreview-review]$ **oc create -f networkpolicy.yaml**

networkpolicy.networking.k8s.io/workshop created

* 1. Verify that you cannot connect to the workshop pod from the default project.
  2. [student@workstation compreview-review]$ **oc debug --to-namespace="default" \**
  3. **-- curl -sS --connect-timeout 5 http://*10.8.0.138:8080***
  4. Starting pod/image-debug ...
  5. curl: (28) Connection timed out after 5000 milliseconds

Removing debug pod ...

* 1. Verify that you can connect to the workshop pod from the workshop project.
  2. [student@workstation compreview-review]$ **oc debug \**
  3. **--to-namespace="template-test" \**
  4. **-- curl -sS http://*10.8.0.138:8080***
  5. Warning: would violate PodSecurity "restricted:latest": ...output omitted...
  6. Starting pod/image-debug ...
  7. <html>
  8. <body>
  9. <h1>Hello, world from nginx!</h1>
  10. </body>
  11. </html>

Removing debug pod ...

1. Create the workshop project template by using the previously created template resources.
   1. Run the oc adm create-bootstrap-project-template command to create the project-template.yaml file to use as the template for new projects.
   2. [student@workstation compreview-review]$ **oc adm \**
   3. **create-bootstrap-project-template \**

**-o yaml > project-template.yaml**

* 1. Use the oc get command to create a YAML list with the following resources:
     + resourcequota/workshop
     + limitrange/workshop
     + networkpolicy/workshop

Redirect the output to append to the project-template.yaml file.

[student@workstation compreview-review]$ **oc get resourcequota/workshop \**

**limitrange/workshop \**

**networkpolicy/workshop \**

**-o yaml >> project-template.yaml**

* 1. Edit the project-template.yaml file to perform the following operations:
     + Cut the contents of the items stanza and paste them immediately before the parameters stanza. Keep the original indentation, because every YAML item of the list must appear at the beginning of the line.
     + Remove any left-over content after the parameters block.
     + Remove the following keys from the limit range and quota definitions:
       - creationTimestamp
       - resourceVersion
       - uid
       - status
       - generation
     + Replace the template-test text with the ${PROJECT\_NAME} text.
     + Add the workshop=${PROJECT\_NAME} label.
     + Rename the admin role binding with the workshop name.

Use the search-and-replace editor function to replace the template-test string with the ${PROJECT\_NAME} template parameter. Optionally, you can use the sed command if it is available.

The solution file is in the ~/DO280/solutions/compreview-review/project-template.yaml path.

[student@workstation compreview-review]$ **sed -i \**

**'s/template-test/${PROJECT\_NAME}/g' project-template.yaml**

Then, move the resource list to the objects key. The project-template.yaml file has the following expected content.

apiVersion: template.openshift.io/v1

kind: Template

metadata:

name: project-request

objects:

- apiVersion: project.openshift.io/v1

kind: Project

metadata:

annotations:

openshift.io/description: ${PROJECT\_DESCRIPTION}

openshift.io/display-name: ${PROJECT\_DISPLAYNAME}

openshift.io/requester: ${PROJECT\_REQUESTING\_USER}

name: ${PROJECT\_NAME}

**labels:**

**workshop: ${PROJECT\_NAME}**

spec: {}

- apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: **workshop**

namespace: ${PROJECT\_NAME}

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: admin

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: User

name: ${PROJECT\_ADMIN\_USER}

- apiVersion: v1

kind: ResourceQuota

metadata:

annotations:

**name: workshop**

namespace: ${PROJECT\_NAME}

spec:

hard:

limits.cpu: "2"

limits.memory: 1Gi

requests.cpu: 1500m

requests.memory: 750Mi

- apiVersion: v1

kind: LimitRange

metadata:

annotations:

**name: workshop**

namespace: ${PROJECT\_NAME}

spec:

limits:

- default:

cpu: 500m

memory: 500Mi

defaultRequest:

cpu: 100m

memory: 250Mi

max:

cpu: 750m

memory: 750Mi

type: Container

- apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

annotations:

**name: workshop**

namespace: ${PROJECT\_NAME}

spec:

ingress:

- from:

- namespaceSelector:

matchLabels:

workshop: ${PROJECT\_NAME}

- namespaceSelector:

matchLabels:

policy-group.network.openshift.io/ingress: ""

podSelector: {}

policyTypes:

- Ingress

parameters:

- name: PROJECT\_NAME

- name: PROJECT\_DISPLAYNAME

- name: PROJECT\_DESCRIPTION

- name: PROJECT\_ADMIN\_USER

- name: PROJECT\_REQUESTING\_USER

* 1. Create the project template in the project-template.yaml file by using the oc create command in the openshift-config namespace.
  2. [student@workstation compreview-review]$ **oc create -f project-template.yaml \**
  3. **-n openshift-config**

template.template.openshift.io/project-request created

* 1. Use the oc edit command to change the cluster project configuration.
  2. [student@workstation compreview-review]$ **oc edit \**

**projects.config.openshift.io cluster**

Edit the resource to match the following content:

apiVersion: config.openshift.io/v1

kind: Project

metadata:

...output omitted...

name: cluster

...output omitted...

**spec:**

**projectRequestTemplate:**

**name: project-request**

To edit the file, you use the default vi editor.

* 1. Use the watch command to view the API server pods.
  2. [student@workstation compreview-review]$ **watch oc get \**

**pod -n openshift-apiserver**

Wait until new pods are created. Press **Ctrl**+**C** to exit the watch command.

1. As the do280-presenter, create the do280 workshop project.
   1. Log in as the do280-presenter user with the redhat password.
   2. [student@workstation compreview-review]$ **oc login -u do280-presenter -p redhat**
   3. Login successful.

...output omitted...

* 1. Use the oc new-project command to create the do280 project.
  2. [student@workstation compreview-review]$ **oc new-project do280**
  3. Now using project "do280" on server ...

...output omitted...

* 1. Verify that the oc new-project command creates the following resources from the template:
     + Quota
     + Limit range
     + Network policy
     + [student@workstation compreview-review]$ **oc get resourcequota/workshop \**
     + **limitrange/workshop \**
     + **networkpolicy/workshop**
     + NAME AGE REQUEST LIMIT
     + resourcequota/workshop 95s requests.cpu: 0/1500m ... limits.cpu: 0/2 ...
     + NAME CREATED AT
     + limitrange/workshop 2023-03-03T10:37:28Z
     + NAME POD-SELECTOR AGE

networkpolicy.networking.k8s.io/workshop <none> 95s

* 1. Verify that the do280 project definition has the workshop=do280 label.
  2. [student@workstation compreview-review]$ **oc get project do280 -o yaml**
  3. apiVersion: project.openshift.io/v1
  4. kind: Project
  5. metadata:
  6. ...output omitted...
  7. labels:
  8. **workshop: do280**
  9. ...output omitted...
  10. name: do280
  11. resourceVersion: "1293438"

...output omitted...

1. As the do280-support user, create the do280-attendees group. Then, assign the edit cluster role to the do280-attendees group, and add the do280-attendee user to the group.
   1. Log in as the do280-support user with the redhat password.
   2. [student@workstation compreview-review]$ **oc login -u do280-support -p redhat**
   3. Login successful.

...output omitted...

* 1. Create the do280-attendees group.
  2. [student@workstation compreview-review]$ **oc adm groups new do280-attendees**

group.user.openshift.io/do280-attendees created

* 1. Assign the edit role to the do280-attendees group in the do280-workshop project.

Add the edit cluster role to the do280-attendees group in the do280 project.

[student@workstation compreview-review]$ **oc adm policy \**

**add-role-to-group edit do280-attendees -n do280**

clusterrole.rbac.authorization.k8s.io/edit added: "do280-attendees"

* 1. As the do280-attendee user, verify that you cannot access the do280 project.

Log in as the do280-attendee user with the redhat password.

[student@workstation compreview-review]$ **oc login -u do280-attendee -p redhat**

Login successful.

**You don't have any projects.** ...

* 1. As the do280-support user, add the do280-attendee user to the do280-attendees group.

Log in as the do280-support user with the redhat password.

[student@workstation compreview-review]$ **oc login -u do280-support -p redhat**

Login successful.

...output omitted...

Use the oc adm groups command to add the do280-attendee user to the workshop-do280-attendees group.

[student@workstation compreview-review]$ **oc adm groups add-users \**

**do280-attendees do280-attendee**

group.user.openshift.io/do280-attendees added: "do280-attendee"

* 1. As the do280-attendee user, verify that you can create workloads in the do280 project.

Log in as the do280-attendee user with the redhat password.

[student@workstation compreview-review]$ **oc login -u do280-attendee -p redhat**

Login successful.

**You have one project on this server: "do280"**

**Using project "do280".**

Use the oc create deployment command to create a deployment without resource specifications.

[student@workstation compreview-review]$ **oc create deployment \**

**attendee-workload \**

**--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0**

deployment.apps/attendee-workload created

1. Change to the home directory to prepare for the next exercise.

[student@workstation appsec-review]$ **cd**

[Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade compreview-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish compreview-review**

## **Lab: Secure Applications**

Configure a project that requires custom settings.

Secure applications by encrypting and restricting network traffic.

Automate cluster maintenance tasks.

**Outcomes**

* Create a project quota.
* Create a limit range.
* Use role-based access control to grant permissions to service accounts and groups.
* Encrypt the traffic end-to-end with TLS by using a signed certificate.
* Restrict cluster internal traffic to pods by using network policies.
* Grant application access to Kubernetes APIs.
* Configure a cluster maintenance application to run periodically.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

[student@workstation ~]$ **lab start compreview-apps**

The lab command copies the exercise files into the ~/DO280/labs/compreview-apps directory and creates the workshop-support group with the do280-support user. The lab command also restores the project template configuration from the previous exercise.

The goal, as a cluster administrator, is to prepare the workshop-support namespace for the support team. Create a namespace instead of a project to avoid using the project template. The project template applies a default configuration for workshop projects, and does not apply the configuration to the workshop-support namespace. Then, as a support team member, you configure and deploy the applications that maintain the cluster and support the workshop experience.

You must set up an application that automatically deletes completed workshops, and set up a social media API that attendees from all workshops use.

**Specifications**

* Create the workshop-support namespace with the category: support label.
* Grant to the workshop-support group the admin role in the cluster.
* Workloads from the workshop-support namespace must enforce the following constraints:
  + The project uses up to 4 CPUs.
  + The project uses up to 4 Gi of RAM.
  + The project requests up to 3.5 CPUs.
  + The project requests up to 3 Gi of RAM.
* Define the default resource specification for workloads:
  + A default limit of 300m CPUs.
  + A default limit of 400 Mi of RAM.
  + A default request of 100m CPUs.
  + A default request of 250 Mi of RAM.
* Any quota or limit range must have the workshop-support name for grading purposes.
* As the do280-support user, deploy the project-cleaner application from the project-cleaner/example-pod.yaml file to the workshop-support namespace by using a project-cleaner cron job that runs every minute.

The project cleaner deletes projects with the workshop label that exist for more than 10 seconds. This short expiration time is deliberate for this lab.

* You must create a project-cleaner-sa service account to use in the project cleaner application.
* The role that the project cleaner needs is defined in the project-cleaner/cluster-role.yaml file.
* Deploy the beeper-db database in the beeper-api/beeper-db.yaml file to the workshop-support namespace.
* Deploy the beeper-api application in the beeper-api/deployment.yaml file to the workshop-support namespace.
* You must configure this application to use TLS end-to-end by using the following specification:
  + Use the beeper-api.pem certificate and the beeper-api.key in the certs directory.
  + Configure the /etc/pki/beeper-api/ path as the mount point for the certificate and key.
  + Set the TLS\_ENABLED environment variable to the true value.
* Update the startup, readiness, and liveness probes to use TLS.
* Create a passthrough route with the beeper-api.apps.ocp4.example.com hostname.
* The database pods, which are pods in the workshop-support namespace with the app=beeper-db label, must accept only TCP traffic from the beeper-api pods in the workshop-support namespace on the 5432 port. You can use the category=support label to identify the pods that belong to the workshop-support namespace.
* Configure the cluster network so that the workshop-support namespace accepts only external ingress traffic to pods that listen on the 8080 port, and blocks traffic from other projects.

1. Change to the ~/DO280/labs/compreview-apps directory and log in to the cluster as the admin user.
   1. Open a terminal window and change to the lab directory.

[student@workstation ~]$ **cd ~/DO280/labs/compreview-apps**

* 1. Log in as the admin user with the redhatocp password.
  2. [student@workstation compreview-apps]$ **oc login -u admin -p redhatocp \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

...output omitted...

1. Create and prepare the workshop-support namespace with the following actions:
   1. Add the category=support label.
   2. Grant the admin cluster role to the workshop-support group.
   3. Create the workshop-support namespace.
   4. [student@workstation compreview-apps]$ **oc create namespace workshop-support**

namespace/workshop-support created

* 1. Use the oc label command to add the category=support label to the workshop-support namespace.
  2. [student@workstation beeper-api]$ **oc label namespace \**
  3. **workshop-support category=support**

namespace/workshop-support labeled

* 1. Change to the workshop-support namespace by using the oc project command.
  2. [student@workstation beeper-api]$ **oc project workshop-support**

Now using project "workshop-support" on server...

* 1. Create a cluster role binding to assign the admin cluster role to the workshop-support group.
  2. [student@workstation compreview-apps]$ **oc adm policy \**
  3. **add-cluster-role-to-group admin workshop-support**

clusterrole.rbac.authorization.k8s.io/admin added: "workshop-support"

1. Create the resource quota for the workshop-support namespace with the following specification.

| **Quota** | **Value** |
| --- | --- |
| limits.cpu | 4 |
| limits.memory | 4Gi |
| requests.cpu | 3500m |
| requests.memory | 3Gi |

* 1. Run the oc create quota command to create the quota.
  2. [student@workstation compreview-apps]$ **oc create quota workshop-support \**
  3. **--hard=limits.cpu=4,limits.memory=4Gi,requests.cpu=3500m,requests.memory=3Gi**

resourcequota/workshop-support created

1. Create the workshop limit range with the following specification.

| **Limit type** | **Value** |
| --- | --- |
| default.cpu | 300m |
| default.memory | 400Mi |
| defaulRequest.cpu | 100m |
| defaulRequest.memory | 250Mi |

* 1. Edit the limitrange.yaml file and replace the CHANGE\_ME label to match the following definition.
  2. apiVersion: v1
  3. kind: LimitRange
  4. metadata:
  5. name: workshop-support
  6. namespace: workshop-support
  7. spec:
  8. limits:
  9. **- default:**
  10. **cpu: 300m**
  11. **memory: 400Mi**
  12. **defaultRequest:**
  13. **cpu: 100m**
  14. **memory: 250Mi**

**type: Container**

* 1. Use the oc apply command to create the limit range in the workshop-support project.
  2. [student@workstation compreview-apps]$ **oc apply -f limitrange.yaml**

limitrange/workshop-support created

1. Create the project-cleaner-sa service account in the workshop-support namespace. Then, assign the role from the project-cleaner/cluster-role.yaml file to the project-cleaner-sa service account.
   1. Create the project-cleaner-sa service account.
   2. [student@workstation compreview-apps]$ **oc create sa project-cleaner-sa**

serviceaccount/project-cleaner-sa created

* 1. Change to the ~/DO280/labs/compreview-apps/project-cleaner directory to access the application files.
  2. [student@workstation compreview-apps]$ **cd \**

**~/DO280/labs/compreview-apps/project-cleaner**

* 1. Create the project-cleaner cluster role by applying the cluster-role.yaml manifest file.
  2. [student@workstation project-cleaner]$ **oc apply -f cluster-role.yaml**

clusterrole.rbac.authorization.k8s.io/project-cleaner created

* 1. Use the oc adm policy add-cluster-role-to-user command to add the project-cleaner role to the project-cleaner-sa service account.
  2. [student@workstation project-cleaner]$ **oc adm policy add-cluster-role-to-user \**
  3. **project-cleaner -z project-cleaner-sa**

clusterrole.rbac.authorization.k8s.io/project-cleaner added: "project-cleaner-sa"

1. As the do280-support user, create the project-cleaner cron job by editing the cron⁠-⁠job⁠.⁠yaml file and by using the example-pod.yaml pod manifest as the job template. Configure the cron job to run every minute.
   1. Log in as the do280-support user with the redhat password.
   2. [student@workstation project-cleaner]$ **oc login -u do280-support -p redhat**
   3. Login successful.

...output omitted...

* 1. Edit the cron-job.yaml file:
     + Replace the CHANGE\_ME label with the "\*/1 \* \* \* \*" schedule to execute the job every minute.
     + Replace the CHANGE\_ME label in the jobTemplate definition with the spec definition from the example-pod.yaml pod manifest.
     + Replace the CHANGE\_ME label in the serviceAccountName key with the project-cleaner-sa service account.

Although the long image name might show across two lines, you must add it as one line.

A solution file is in the ~/DO280/solutions/compreview-apps/project-cleaner/cron-job.yaml path.

apiVersion: batch/v1

kind: CronJob

metadata:

name: project-cleaner

namespace: workshop-support

spec:

schedule: **"\*/1 \* \* \* \*"**

concurrencyPolicy: Forbid

jobTemplate:

spec:

template:

spec:

**restartPolicy: Never**

**serviceAccountName: project-cleaner-sa**

**containers:**

**- name: project-cleaner**

**image: registry.ocp4.example.com:8443/redhattraining/do280-project-cleaner:v1.1**

**imagePullPolicy: Always**

**env:**

**- name: "PROJECT\_TAG"**

**value: "workshop"**

**- name: "EXPIRATION\_SECONDS"**

**value: "10"**

**resources:**

**limits:**

**cpu: 100m**

**memory: 200Mi**

* 1. Create the cron job.
  2. [student@workstation project-cleaner]$ **oc apply -f cron-job.yaml**

cronjob.batch/project-cleaner created

### NOTE

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

* 1. Verify that the project cleaner application is deployed correctly, by creating a clean⁠-⁠test project.
  2. [student@workstation project-cleaner]$ **oc new-project clean-test**
  3. Now using project "clean-test" on server...

...output omitted...

Change to the workshop-support namespace.

[student@workstation project-cleaner]$ **oc project workshop-support**

Now using project "workshop-support" on server...

Wait for a successful job run. Then, get the pod name from the last job run.

[student@workstation project-cleaner]$ **oc get jobs,pods**

NAME COMPLETIONS DURATION AGE

job.batch/project-cleaner-27949859 1/1 7s 2m40s

job.batch/project-cleaner-27949860 1/1 7s 100s

job.batch/project-cleaner-27949861 1/1 6s 40s

NAME READY STATUS RESTARTS AGE

pod/project-cleaner-27949859-f98vj 0/1 Completed 0 2m40s

pod/project-cleaner-27949860-j8td5 0/1 Completed 0 100s

**pod/project-cleaner-27949861-p262t 0/1 Completed 0 40s**

Read the logs of the pod that completed the job.

[student@workstation project-cleaner]$ **oc logs \**

**pod/project-cleaner-*27949861-p262t***

Listing namespaces with label workshop:

- namespace: clean-test, created 55.327453 seconds ago...

Deleting namespaces: clean-test

Namespace 'clean-test' deleted

### NOTE

You might see deleted projects from other exercises in the course.

* 1. Verify that the cron job deletes the clean-test project, by using the oc get project command.
  2. [student@workstation project-cleaner]$ **oc get project clean-test**

Error from server (NotFound): namespaces "clean-test" not found

1. Create the beeper database by applying the beeper-api/beeper-db.yaml file.
   1. Change to the ~/DO280/labs/compreview/beeper-api directory to access the application files.

[student@workstation project-cleaner]$ **cd ~/DO280/labs/compreview-apps/beeper-api**

* 1. Use the oc apply command to create the database in the workshop-support namespace.
  2. [student@workstation beeper-api]$ **oc apply -f beeper-db.yaml**
  3. secret/beeper-db created
  4. service/beeper-db created
  5. persistentvolumeclaim/beeper-db created

deployment.apps/beeper-db created

* 1. Verify that the database pod is running by using the oc get pod command to get the pods with the app=beeper-db label.
  2. [student@workstation beeper-api]$ **oc get pod -l app=beeper-db**
  3. NAME READY STATUS RESTARTS AGE

beeper-db-688756744f-rgxpg 1/1 Running 0 3m51s

1. Configure TLS on the beeper-api deployment by using a signed certificate by a corporate CA to accept TLS connections from outside the cluster.

You have the CA certificate and the signed certificate for the beeper-api.apps.ocp4.example.com domain in the beeper-api/certs directory of the lab.

Use the following settings in the deployment to configure TLS:

* 1. Set the path for the certificate and key to /etc/pki/beeper-api/.
  2. Set the TLS\_ENABLED environment variable to the true value.
  3. Update the startup, readiness, and liveness probes to use TLS.
  4. Create the beeper-api-cert secret by using the beeper-api.pem certificate and the beeper-api.key key from the lab directory.
  5. [student@workstation beeper-api]$ **oc create secret tls beeper-api-cert \**
  6. **--cert certs/beeper-api.pem --key certs/beeper-api.key**

secret/beeper-api-cert created

* 1. Edit the beeper-api deployment in the deployment.yaml file to mount the beeper-api-cert secret on the /etc/pki/beeper-api/ path.
  2. apiVersion: apps/v1
  3. kind: Deployment
  4. metadata:
  5. name: beeper-api
  6. namespace: workshop-support
  7. spec:
  8. ...output omitted...
  9. spec:
  10. containers:
  11. - name: beeper-api
  12. ...output omitted...
  13. env:
  14. - name: TLS\_ENABLED
  15. value: "false"
  16. **volumeMounts:**
  17. **- name: beeper-api-cert**
  18. **mountPath: /etc/pki/beeper-api/**
  19. **volumes:**
  20. **- name: beeper-api-cert**
  21. **secret:**
  22. **defaultMode: 420**

**secretName: beeper-api-cert**

* 1. Edit the beeper-api deployment in the deployment.yaml file to configure TLS for the application and for the startup, readiness, and liveness probes.
  2. apiVersion: apps/v1
  3. kind: Deployment
  4. metadata:
  5. name: beeper-api
  6. namespace: workshop-support
  7. spec:
  8. ...output omitted...
  9. spec:
  10. containers:
  11. - name: beeper-api
  12. ...output omitted...
  13. ports:
  14. - containerPort: 8080
  15. readinessProbe:
  16. httpGet:
  17. port: 8080
  18. path: /readyz
  19. **scheme: HTTPS**
  20. livenessProbe:
  21. httpGet:
  22. port: 8080
  23. path: /livez
  24. **scheme: HTTPS**
  25. startupProbe:
  26. httpGet:
  27. path: /readyz
  28. port: 8080
  29. **scheme: HTTPS**
  30. failureThreshold: 30
  31. periodSeconds: 3
  32. env:
  33. - name: TLS\_ENABLED
  34. **value: "true"**

...output omitted...

* 1. Use the oc apply command to create the beeper-api deployment.
  2. [student@workstation beeper-api]$ **oc apply -f deployment.yaml**

deployment.apps/beeper-api created

* 1. Edit the service.yaml file to configure the beeper-api service to listen on the standard HTTPS 443 port and to forward connections to pods with the app: beeper-api label on port 8080.
  2. apiVersion: v1
  3. kind: Service
  4. metadata:
  5. name: beeper-api
  6. namespace: workshop-support
  7. spec:
  8. selector:
  9. **app: beeper-api**
  10. ports:
  11. - port: **443**
  12. targetPort: **8080**

name: https

* 1. Use the oc apply command to create the beeper-api service.
  2. [student@workstation beeper-api]$ **oc apply -f service.yaml**

service/beeper-api created

1. Expose the beeper API to outer cluster access by using the FQDN in the signed certificate by the corporate CA.
   1. Create a passthrough route for the beeper-api service by using the beeper-api.apps.ocp4.example.com hostname.
   2. [student@workstation beeper-api]$ **oc create route \**
   3. **passthrough beeper-api-https \**
   4. **--service beeper-api \**
   5. **--hostname beeper-api.apps.ocp4.example.com**

route.route.openshift.io/beeper-api-https created

* 1. Use the curl command to the https://beeper-api.apps.ocp4.example.com/api/beeps URL to verify that the beeper API is accessible from outside the cluster. Add the --cacert option to accept the certs/ca.pem CA.
  2. [student@workstation beeper-api]$ **curl -s --cacert \**
  3. **certs/ca.pem https://beeper-api.apps.ocp4.example.com/api/beeps; echo**

[]

1. Optionally, open a web browser and verify that you can access the API by navigating to the https://beeper-api.apps.ocp4.example.com/swagger-ui.html URL. When you see the warning about the security risk, click **Advanced…​** and then click **Accept the Risk and Continue**.
2. Configure network policies to allow only TCP ingress traffic on port 5432 to database pods from the beeper-api pods.
   1. Verify that you can access the beeper-db service from the workshop-support namespace by testing TCP connectivity to the database service. Use the oc debug command to create a pod with the nc command with the -z option to test TCP access.
   2. [student@workstation beeper-api]$ **oc debug --to-namespace="workshop-support" -- \**
   3. **nc -z -v beeper-db.workshop-support.svc.cluster.local 5432**
   4. Starting pod/image-debug ...
   5. Ncat: Version 7.70 ( https://nmap.org/ncat )
   6. Ncat: **Connected to 172.30.219.94:5432.**
   7. Ncat: 0 bytes sent, 0 bytes received in 0.02 seconds.

Removing debug pod ...

* 1. Create an entry in the database by using the following curl command.
  2. [student@workstation beeper-api]$ **curl -s --cacert certs/ca.pem -X 'POST' \**
  3. **'https://beeper-api.apps.ocp4.example.com/api/beep' \**
  4. **-H 'Content-Type: application/json' \**

**-d '{ "username": "user1", "content": "first message" }'**

* 1. Edit the db-networkpolicy.yaml file so that only pods with the app: beeper-api label can connect to database pods.
  2. apiVersion: networking.k8s.io/v1
  3. kind: NetworkPolicy
  4. metadata:
  5. name: database-policy
  6. namespace: workshop-support
  7. spec:
  8. podSelector:
  9. matchLabels:
  10. **app: beeper-db**
  11. ingress:
  12. - from:
  13. **- namespaceSelector:**
  14. **matchLabels:**
  15. **category: support**
  16. **podSelector:**
  17. **matchLabels:**
  18. **app: beeper-api**
  19. ports:
  20. **- protocol: TCP**

**port: 5432**

* 1. Create the network policy.
  2. [student@workstation beeper-api]$ **oc apply -f db-networkpolicy.yaml**

networkpolicy.networking.k8s.io/beeper-api-ingresspolicy created

* 1. Verify that you cannot connect to the database, by running the previous nc command.
  2. [student@workstation beeper-api]$ **oc debug --to-namespace="workshop-support" -- \**
  3. **nc -z -v beeper-db.workshop-support.svc.cluster.local 5432**
  4. Starting pod/image-debug ...
  5. Ncat: Version 7.70 ( https://nmap.org/ncat )
  6. Ncat: **Connection timed out.**

Removing debug pod ...

* 1. Verify that the API pods have access to the database pods, by running the curl command to query the API by using the external route.
  2. [student@workstation beeper-api]$ **curl -s --cacert \**
  3. **certs/ca.pem https://beeper-api.apps.ocp4.example.com/api/beeps; echo**

[{"id":1,"username":"user1","content":"first message","votes":0}]

1. Configure network policies in the workshop-support namespace to accept only ingress connections from the OpenShift router pods to port 8080.
   1. Verify that you can access the API service from the workshop-support namespace by testing TCP connectivity. Use the oc debug command to create a pod with the nc command with the -z option to test TCP access.
   2. [student@workstation beeper-api]$ **oc debug --to-namespace="workshop-support" -- \**
   3. **nc -z -v beeper-api.workshop-support.svc.cluster.local 443**
   4. Starting pod/image-debug ...
   5. Ncat: Version 7.70 ( https://nmap.org/ncat )
   6. Ncat: Connected to 172.30.32.28:443.
   7. Ncat: 0 bytes sent, 0 bytes received in 0.02 seconds.

Removing debug pod ...

* 1. Edit the beeper-api-ingresspolicy.yaml file to accept ingress connections from router pods by adding a namespace selector with the policy-group.network.openshift.io/ingress label.
  2. apiVersion: networking.k8s.io/v1
  3. kind: NetworkPolicy
  4. metadata:
  5. name: beeper-api-ingresspolicy
  6. namespace: workshop-support
  7. spec:
  8. podSelector: **{}**
  9. ingress:
  10. - from:
  11. **- namespaceSelector:**
  12. **matchLabels:**
  13. **policy-group.network.openshift.io/ingress: ""**
  14. ports:
  15. **- protocol: TCP**

**port: 8080**

* 1. Create the network policy.
  2. [student@workstation beeper-api]$ **oc apply -f beeper-api-ingresspolicy.yaml**

networkpolicy.networking.k8s.io/beeper-api created

* 1. Verify that you cannot access the API service from the workshop-support namespace. Use the oc debug command to create a pod with the nc command with the -z option to test TCP access.
  2. [student@workstation beeper-api]$ **oc debug --to-namespace="workshop-support" -- \**
  3. **nc -z -v beeper-api.workshop-support.svc.cluster.local 443**
  4. Starting pod/image-debug ...
  5. Ncat: Version 7.70 ( https://nmap.org/ncat )
  6. Ncat: **Connection timed out.**

Removing debug pod ...

* 1. Verify that the API pods are accessible from outside the cluster by running the curl command to query the API external route.
  2. [student@workstation beeper-api]$ **curl -s --cacert \**
  3. **certs/ca.pem https://beeper-api.apps.ocp4.example.com/livez; echo**

{"status":"UP"}

1. Change to the home directory.

[student@workstation appsec-review]$ **cd**

[Hide Solution](https://rol.redhat.com/rol/app/)

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade compreview-apps**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish compreview-apps**