Chapter 2.  Kubernetes and OpenShift Command-line Interfaces and APIs

[**The Kubernetes and OpenShift Command-line Interfaces**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: The Kubernetes and OpenShift Command-line Interfaces**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s02/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Inspect Kubernetes Resources**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s03/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Inspect Kubernetes Resources**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s04/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Assess the Health of an OpenShift Cluster**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s05/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Assess the Health of an OpenShift Cluster**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s06/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Lab: Kubernetes and OpenShift Command-line Interfaces and APIs**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s07/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Quiz: Kubernetes and OpenShift Command-line Interfaces and APIs**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s08/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Summary**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s09/c2358540-87d5-48de-b49e-6f23bdcd629c)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Access an OpenShift cluster by using the command line and query its Kubernetes API resources to assess the health of a cluster. |
| **Objectives** | * Access an OpenShift cluster by using the Kubernetes and OpenShift command-line interfaces. * Query, format, and filter attributes of Kubernetes resources. * Query the health of essential cluster services and components. |
| **Sections** | * The Kubernetes and OpenShift Command-line Interfaces (and Guided Exercise) * Inspect Kubernetes Resources (and Guided Exercise) * Assess the Health of an OpenShift Cluster (and Guided Exercise) |
| **Lab** | * Kubernetes and OpenShift Command-line Interfaces and APIs |

The Kubernetes and OpenShift Command-line Interfaces

Objectives

* Access an OpenShift cluster by using the Kubernetes and OpenShift command-line interfaces.

The Kubernetes and OpenShift Command-line Interfaces

You can manage an OpenShift cluster from the web console or by using the kubectl or oc command-line interfaces (CLI). The kubectl commands are native to Kubernetes, and are a thin wrapper over the Kubernetes API. The OpenShift oc commands are a superset of the kubectl commands, and add commands for the OpenShift-specific features. In this course, examples of both the kubectl and the oc commands are shown, to highlight the differences between the commands.

With the oc command, you can create applications and manage Red Hat OpenShift Container Platform (RHOCP) projects from a terminal. The OpenShift CLI is ideal in the following situations:

* Working directly with project source code.
* Scripting OpenShift Container Platform operations.
* Managing projects that are restricted by bandwidth.
* When the web console is unavailable.
* Working with OpenShift resources, such as routes and deployment configs.

Kubernetes Command-line Tool

The oc CLI installation also includes an installation of the kubectl CLI, which is the recommended method for installing the kubectl CLI for OpenShift users.

You can also install the kubectl CLI independently of the oc CLI. You must use a kubectl CLI version that is within one minor version difference of your cluster. For example, a v1.26 client can communicate with v1.25, v1.26, and v1.27 control planes. Using the latest compatible version of the kubectl CLI can help to avoid unforeseen issues.

To perform a manual installation of the kubectl binary for a Linux installation, you must first download the latest release by using the curl command.

[user@host ~]$ **curl -LO "https://dl.k8s.io/release/$(curl -L \**

**-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"**

Then, you must download the kubectl checksum file and then validate the kubectl binary against the checksum file.

[user@host ~]$ **curl -LO "https://dl.k8s.io/$(curl -L \**

**-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"**

[user@host ~]$ **echo "$(cat kubectl.sha256) kubectl" | sha256sum --check**

kubectl: OK

If the check fails, then the sha256sum command exits with nonzero status, and prints a kubectl: FAILED message.

You can then install the kubectl CLI.

[user@host ~]$ **sudo install -o root -g root -m 0755 kubectl \**

**/usr/local/bin/kubectl**

**Note**

If you do not have root access on the target system, you can still install the kubectl CLI to the ~/.local/bin directory. For more information, refer to <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>.

Finally, use the kubectl version command to verify the installed version. This command prints the client and server versions. Use the --client option to view the client version only.

[user@host ~]$ **kubectl version --client**

Alternatively, a distribution that is based on Red Hat Enterprise Linux (RHEL) can install the kubectl CLI with the following command:

[user@host ~]$ **cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo**

**[kubernetes]**

**name=Kubernetes**

**baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch**

**enabled=1**

**gpgcheck=1**

**gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg**

**EOF**

[user@host ~]$ **sudo yum install -y kubectl**

To view a list of the available kubectl commands, use the kubectl --help command.

[user@host ~]$ **kubectl --help**

kubectl controls the Kubernetes cluster manager.

Find more information at:

https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):

create Create a resource from a file or from stdin

expose Take a replication controller, service, deployment or pod and

expose it as a new Kubernetes Service

run Run a particular image on the cluster

set Set specific features on objects

Basic Commands (Intermediate):

*...output omitted...*

You can also use the --help option on any command to view detailed information about the command, including its purpose, examples, available subcommands, and options. For example, the following command provides information about the kubectl create command and its usage.

[user@host ~]$ **kubectl create --help**

Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

Examples:

# Create a pod using the data in pod.json

kubectl create -f ./pod.json

# Create a pod based on the JSON passed into stdin

cat pod.json | kubectl create -f -

# Edit the data in registry.yaml in JSON then create the resource using the edited data

kubectl create -f registry.yaml --edit -o json

Available Commands:

clusterrole Create a cluster role

clusterrolebinfing Create a cluster role binding for a particular cluster role

*...output omitted...*

Kubernetes uses many resource components to support applications. The kubectl explain command provides detailed information about the attributes of a given resource. For example, use the following command to learn more about the attributes of a pod resource.

[user@host ~]$ **kubectl explain pod**

KIND: Pod

VERSION: v1

DESCRIPTION:

Pod is a collection of containers that can run on a host. This resource is

created by clients and scheduled onto hosts.

FIELDS:

apiVersion <string>

APIVersion defines the versioned schema of this representation of an

object.

*...output omitted...*

Refer to [Kubernetes Documentation - Getting Started](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/) for further information about the kubectl commands.

The OpenShift Command-line Tool

The main method of interacting with an RHOCP cluster is by using the oc command.

You can download the oc CLI from the OpenShift web console to ensure that the CLI tools are compatible with the RHOCP cluster. From the OpenShift web console, navigate to **Help** → **Command line tools**. The **Help** menu is represented by a ? icon. The web console provides several installation options for the oc client, such as downloads for the following operating systems:

* x86\_64 Windows, Mac, and Linux systems
* ARM 64 Linux and Mac systems
* Linux for IBM Z, IBM Power, and little endian

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

The basic usage of the oc command is through its subcommands in the following syntax:

[user@host ~]$ **oc *command***

Because the oc CLI is a superset of the kubectl CLI, the version, --help, and explain commands are the same for both CLIs. However, the oc CLI includes additional commands that are not included in the kubectl CLI, such as the oc login and oc new-project commands.

Managing Resources at the Command Line

Developers who are familiar with Kubernetes can use the kubectl utility to manage a RHOCP cluster. This course uses the oc command-line utility, to take advantage of additional RHOCP features. The oc commands manage resources that are exclusive to RHOCP, such as projects, deployment configurations, routes, and image streams.

Before you can interact with your RHOCP cluster, you must authenticate your requests. Use the oc login command to authenticate your requests. The oc login command provides role-based authentication and authorization that protects the RHOCP cluster from unauthorized access. The syntax to log in is shown below:

[user@host ~]$ **oc login *cluster-url***

For example, in this course, you can use the following command:

[user@host ~]$ **oc login https://api.ocp4.example.com:6443**

Username: **developer**

Password: **developer**

Login successful.

You don't have any projects. You can try to create a new project, by running

$ oc new-project <projectname>

Welcome to OpenShift! See 'oc help' to get started.

After authenticating to the RHOCP cluster, you can create a project with the oc new-project command. Projects provide isolation between your application resources. Projects are Kubernetes namespaces with additional annotations that provide multitenancy scoping for applications.

[user@host ~]$ **oc new-project myapp**

Several essential commands can manage RHOCP and Kubernetes resources, as described here. Unless otherwise specified, the following commands are compatible with both the oc and kubectl CLIs.

Some commands require a user with cluster administrator access. The following list includes several useful oc commands for cluster administrators.

**oc cluster-info**

The cluster-info command prints the address of the control plane and other cluster services. The oc cluster-info dump command expands the output to include helpful details for debugging cluster problems.

[user@host ~]$ **oc cluster-info**

Kubernetes control plane is running at https://api.ocp4.example.com:6443

*...output omitted...*

**oc api-versions**

The structure of cluster resources has a corresponding API version, which the oc api-versions command displays. The command prints the supported API versions on the server, in the form of "group/version".

In the following example, the group is admissionregistration.k8s.io and the version is v1:

[user@host ~]$ **oc api-versions**

admissionregistration.k8s.io/v1

*...output omitted...*

**oc get clusteroperator**

The cluster operators that Red Hat ships serve as the architectural foundation for RHOCP. RHOCP installs cluster operators by default. Use the oc get clusteroperator command to see a list of the cluster operators:

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

NAME VERSION AVAILABLE PROGRESSING DEGRADED SINCE ...

authentication 4.14.0 True False False 18d

baremetal 4.14.0 True False False 18d

*...output omitted...*

Other useful commands are available to both regular and administrator users:

**oc get**

Use the get command to retrieve information about resources in the selected project. Generally, this command shows only the most important characteristics of the resources, and omits more detailed information.

The oc get *RESOURCE\_TYPE* command displays a summary of all resources of the specified type.

For example, the following command returns the list of the pod resources in the current project:

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

NAME READY STATUS RESTARTS AGE

quotes-api-6c9f758574-nk8kd 1/1 Running 0 39m

quotes-ui-d7d457674-rbkl7 1/1 Running 0 67s

You can use the oc get *RESOURCE\_TYPE* *RESOURCE\_NAME* command to export a resource definition. Typical use cases include creating a backup or modifying a definition. The -o yaml option prints the object representation in YAML format. You can change to JSON format by providing a -o json option.

**oc get all**

Use the oc get all command to retrieve a summary of the most important components of a cluster. This command iterates through the major resource types for the current project, and prints a summary of their information:

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

NAME DOCKER REPO TAGS UPDATED

is/nginx 172.30.1.1:5000/basic-kubernetes/nginx latest About an hour ago

NAME REVISION DESIRED CURRENT TRIGGERED BY

dc/nginx 1 1 1 config,image(nginx:latest)

NAME DESIRED CURRENT READY AGE

rc/nginx-1 1 1 1 1h

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

svc/nginx 172.30.72.75 <none> 80/TCP,443/TCP 1h

NAME READY STATUS RESTARTS AGE

po/nginx-1-ypp8t 1/1 Running 0 1h

**oc describe**

If the summaries from the get command are insufficient, then you can use the oc describe *RESOURCE\_TYPE* *RESOURCE\_NAME* command to retrieve additional information. Unlike the get command, you can use the describe command to iterate through all the different resources by type. Although most major resources can be described, this function is not available across all resources. The following example demonstrates describing a pod resource:

[user@host ~]$ **oc describe mysql-openshift-1-glgrp**

Name: mysql-openshift-1-glqrp

Namespace: mysql-openshift

Priority: 0

Node: cluster-worker-1/172.25.250.52

Start Time: Fri, 15 Feb 2019 02:14:34 +0000

Labels: app=mysql-openshift

deployment=mysql-openshift-1

*...output omitted...*

Status: Running

IP: 10.129.0.85

**oc explain**

To learn about the fields of an API resource object, use the oc explain command. This command describes the purpose and the fields that are associated with each supported API resource. You can also use this command to print the documentation of a specific field of a resource. Fields are identified via a JSONPath identifier. The following example prints the documentation for the .spec.containers.resources field of the pod resource type:

[user@host ~]$ **oc explain pods.spec.containers.resources**

KIND: Pod

VERSION: v1

FIELD: resources <ResourceRequiremnts>

DESCRIPTION:

Compute Resources required by this container. Cannot be updated. More info:

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

ResourceRequirements describes the compute resource requirements.

FIELDS:

claims <[]ResourceClaim>

Claims lists the names of resources, defined in spec.resourceClaims, that

are used by this container.

This is an alpha field nd requires enabling the DynamicResourceAllocation

feature gate.

This field is immutable. It can only be set for containers.

limits <map[string]Quantity>

Limits describes the maximum amount of compute resources allowed. More

info:

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

requests <map[string]Quantity>

Requests describes the minimum amount of compute resources required. If

Requests is omitted for a container, it defaults to Limits if that is

explicitly specified, otherwise to an implementation-defined value. Requests

cannot exceed Limits. More info:

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Add the --recursive flag to display all fields of a resource without descriptions. Information about each field is retrieved from the server in OpenAPI format.

**oc create**

Use the create command to create a RHOCP resource in the current project. This command creates resources from a resource definition. Typically, this command is paired with the oc get *RESOURCE\_TYPE* *RESOURCE\_NAME* -o yaml command for editing definitions. Developers commonly use the -f flag to indicate the file that contains the JSON or YAML representation of an RHOCP resource.

For example, to create resources from the pod.yaml file, use the following command:

[user@host ~]$ **oc create -f pod.yaml**

pod/quotes-pod created

RHOCP resources in the YAML format are discussed later.

**oc status**

The oc status command provides a high-level overview of the current project. The command shows services, deployments, build configurations, and active deployments. Information about any misconfigured components is also shown. The --suggest option shows additional details for any identified issues.

**oc delete**

Use the delete command to delete an existing RHOCP resource from the current project. You must specify the resource type and the resource name.

For example, to delete the quotes-ui pod, use the following command:

[user@host ~]$ **oc delete pod quotes-ui**

pod/quotes-ui deleted

A fundamental understanding of the RHOCP architecture is needed here, because deleting managed resources, such as pods, results in the automatic creation of new instances of those resources. When a project is deleted, it deletes all the resources and applications within it.

Each of these commands is executed in the current selected project. To execute commands in a different project, you must include the --namespace or -n options.

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

NAME READY STATUS RESTARTS AGE

apiserver-68c9485699-ndqlc 2/2 Running 2 18d

Refer to the references for a complete list of oc commands.

Authentication with OAuth

For users to interact with RHOCP, they must first authenticate to the cluster. The authentication layer identifies the user that is associated with requests to the RHOCP API. After authentication, the authorization layer then uses information about the requesting user to determine whether the request is allowed.

A user in OpenShift is an entity that can make requests to the RHOCP API. An RHOCP User object represents an actor that can be granted permissions in the system by adding roles to the user or to the user's groups. Typically, this represents the account of a developer or an administrator.

Several types of users can exist.

**Regular users**

Most interactive RHOCP users are represented by this user type. An RHOCP User object represents a regular user.

**System users**

Infrastructure uses system users to interact with the API securely. Some system users are automatically created, including the cluster administrator, with access to everything. By default, unauthenticated requests use an anonymous system user.

**Service accounts**

ServiceAccount objects represent service accounts. RHOCP creates service accounts automatically when a project is created. Project administrators can create additional service accounts to define access to the contents of each project.

Each user must authenticate to access a cluster. After authentication, policy determines what the user is authorized to do.

**Note**

Authentication and authorization are covered in greater detail in the "DO280: Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster" course.

The RHOCP control plane includes a built-in OAuth server. To authenticate themselves to the API, users obtain OAuth access tokens. Token authentication is the only guaranteed method to work with any OpenShift cluster, because enterprise Single Sign-On (SSO) might replace the login form of the web console.

When a person requests a new OAuth token, the OAuth server uses the configured identity provider to determine the identity of the person who makes the request. The OAuth server then determines the user that the identity maps to; creates an access token for that user; and then returns the token for use.

To retrieve an OAuth token by using the OpenShift web console, navigate to **Help** → **Command line tools**. The **Help** menu is represented by a ? icon.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

On the Command Line Tools page, navigate to **Copy login Command**. The following page requires you to log in with your OpenShift user credentials. Next, navigate to **Display token**. Use the command under the Log in with this token label to log in to the OpenShift API.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Copy the command from the web console and paste it on the command line. The copied command uses the --token and --server options, similar to the following example.

[user@host ~]$ **oc login --token=*sha256-BW...rA8* \**

**--server=https://api.ocp4.example.com:6443**

Guided Exercise: The Kubernetes and OpenShift Command-line Interfaces

Access an OpenShift cluster by using the command-line to get information about cluster services and nodes.

**Outcomes**

* Use the OpenShift web console to locate the installation file for the oc OpenShift command-line interface.
* Get and use a token from the web console to access the cluster from the command line.
* Identify key differences between the kubectl and oc command-line tools.
* Identify the main components of OpenShift and Kubernetes.

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

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

[student@workstation ~]$ **lab start cli-interfaces**

**Instructions**

1. Log in to the OpenShift web console as the developer user. Locate the installation file for the oc OpenShift command-line interface (CLI).
   1. Open a web browser and then navigate to https://console-openshift-console.apps.ocp4.example.com.
   2. Click **Red Hat Identity Management** and log in as the developer user with the developer password.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* 1. Locate the installation file for the oc CLI. From the OpenShift web console, select **Help** → **Command line tools**. The **Help** menu is represented by a ? icon.

A screenshot of a computer

Description automatically generated

The oc binary is available for multiple operating systems and architectures. For each operating system and architecture, the oc binary also includes the kubectl binary.

**Note**

You do not need to download or install the oc and kubectl binaries, which are already installed on the workstation machine.

1. Download an authorization token from the web console. Then, use the token and the oc command to log in to the OpenShift cluster.
   1. From the **Command Line Tools** page, click the **Copy login command** link.
   2. The link opens a login page. Click **Red Hat Identity Management** and log in as the developer user with the developer password.
   3. A web page is displayed. Click the **Display token** link to show your API token and the login command.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

* 1. Copy the oc login command to your clipboard. Open a terminal window and then use the copied command to log in to the cluster with your token.
  2. [student@workstation ~]$ **oc login --token=*sha256-fypX...Ot6A* \**
  3. **--server=https://api.ocp4.example.com:6443**
  4. Logged into "https://api.ocp4.example.com:6443" as "developer" using the token provided.

*...output omitted...*

1. Compare the available commands for the kubectl and oc commands.
   1. Use the help command to list and review the available commands for the kubectl command.
   2. [student@workstation ~]$ **kubectl help**
   3. kubectl controls the Kubernetes cluster manager.
   4. Find more information at: https://kubernetes.io/docs/reference/kubectl/
   5. Basic Commands (Beginner):
   6. create Create a resource from a file or from stdin
   7. expose Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service
   8. run Run a particular image on the cluster
   9. set Set specific features on objects
   10. Basic Commands (Intermediate):
   11. explain Get documentation for a resource
   12. get Display one or many resources
   13. edit Edit a resource on the server
   14. delete Delete resources by file names, stdin, resources and names, or by resources and label selector

*...output omitted...*.

Notice that the kubectl command does not provide a login command.

* 1. Examine the available subcommands and options for the kubectl create command by using the --help option.
  2. [student@workstation ~]$ **kubectl create --help**
  3. Create a resource from a file or from stdin.
  4. JSON and YAML formats are accepted.
  5. Examples:
  6. # Create a pod using the data in pod.json
  7. kubectl create -f ./pod.json
  8. *...output omitted...*.
  9. Available Commands:
  10. clusterrole Create a cluster role
  11. clusterrolebinding Create a cluster role binding for a particular cluster role
  12. configmap Create a config map from a local file, directory or literal value
  13. cronjob Create a cron job with the specified name
  14. deployment Create a deployment with the specified name
  15. *...output omitted...*
  16. Options:
  17. --allow-missing-template-keys=true:
  18. If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to
  19. golang and jsonpath output formats.
  20. --dry-run='none':
  21. Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without
  22. sending it. If server strategy, submit server-side request without persisting the resource.
  23. *...output omitted...*.
  24. Usage:
  25. kubectl create -f FILENAME [options]
  26. Use "kubectl create <command> --help" for more information about a given command.

Use "kubectl options" for a list of global command-line options (applies to all commands).

You can use the --help option with any kubectl command. The --help option provides information about a command, including the available subcommands and options, and the command syntax.

* 1. List and review the available commands for the oc binary by using the help command.
  2. [student@workstation ~]$ **oc help**
  3. OpenShift Client
  4. This client helps you develop, build, deploy, and run your applications on any
  5. OpenShift or Kubernetes cluster. It also includes the administrative
  6. commands for managing a cluster under the 'adm' subcommand.
  7. Basic Commands:
  8. login Log in to a server
  9. new-project Request a new project
  10. new-app Create a new application
  11. status Show an overview of the current project
  12. project Switch to another project
  13. projects Display existing projects
  14. explain Get documentation for a resource

*...output omitted...*.

The oc command supports the same capabilities as the kubectl command. The oc command provides additional commands to natively support an OpenShift cluster. For example, the new-project command creates a project, which is a Kubernetes namespace, in the OpenShift cluster. The new-app command is unique to the oc command. It creates applications by using existing source code or prebuilt images.

* 1. Use the --help option with the oc create command to view the available subcommands and options.
  2. [student@workstation ~]$ **oc create --help**
  3. Create a resource from a file or from stdin.
  4. JSON and YAML formats are accepted.
  5. Examples:
  6. # Create a pod using the data in pod.json
  7. oc create -f ./pod.json
  8. *...output omitted...*
  9. Available Commands:
  10. build Create a new build
  11. clusterresourcequota Create a cluster resource quota
  12. clusterrole Create a cluster role
  13. clusterrolebinding Create a cluster role binding for a particular cluster role
  14. configmap Create a config map from a local file, directory or literal value
  15. cronjob Create a cron job with the specified name
  16. deployment Create a deployment with the specified name
  17. deploymentconfig Create a deployment config with default options that uses a given image
  18. *...output omitted...*.
  19. Options:
  20. --allow-missing-template-keys=true:
  21. If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to
  22. golang and jsonpath output formats.
  23. --dry-run='none':
  24. Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without
  25. sending it. If server strategy, submit server-side request without persisting the resource.
  26. *...output omitted...*
  27. Usage:
  28. oc create -f FILENAME [options]

*....output omitted...*.

The oc create command includes the same subcommands and options as the kubectl create command, and provides additional subcommands for OpenShift resources. For example, you can use the oc create command to create OpenShift resources such as a deployment configuration, a route, and an image stream.

1. Identify the components and Kubernetes resources of an OpenShift cluster by using the terminal. Unless otherwise noted, all commands are available for the oc and kubectl commands.
   1. In a terminal, use the oc login command to log in to the cluster as the admin user with the redhatocp password. Regular cluster users, such as the developer user, cannot list resources at a cluster scope.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp**
   3. Login successful

*...output omitted...*

* 1. Identify the cluster version with the version command.
  2. [student@workstation ~]$ **oc version**
  3. Client Version: 4.14.0
  4. Kustomize Version: v5.0.1
  5. **Server Version: 4.14.0**

Kubernetes Version: v1.27.6+f67aeb3

* 1. Use the cluster-info command to identify the URL for the Kubernetes control plane.
  2. [student@workstation ~]$ **oc cluster-info**
  3. Kubernetes control plane is running at https://api.ocp4.example.com:6443

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

* 1. Identify the supported API versions by using the api-versions command.
  2. [student@workstation ~]$ **oc api-versions**
  3. admissionregistration.k8s.io/v1
  4. apiextensions.k8s.io/v1
  5. apiregistration.k8s.io/v1
  6. apiserver.openshift.io/v1
  7. apps.openshift.io/v1
  8. apps/v1

*...output omitted...*.

* 1. List cluster operators with the get clusteroperator command.
  2. [student@workstation ~]$ **oc get clusteroperator**
  3. NAME VERSION AVAILABLE PROGRESSING DEGRADED SINCE ...
  4. authentication 4.14.0 True False False 18d
  5. baremetal 4.14.0 True False False 18d
  6. cloud-controller-manager 4.14.0 True False False 18d
  7. cloud-credential 4.14.0 True False False 18d
  8. cluster-autoscaler 4.14.0 True False False 18d
  9. config-operator 4.14.0 True False False 18d
  10. console 4.14.0 True False False 18d
  11. control-plane-machine-set 4.14.0 True False False 18d
  12. csi-snapshot-controller 4.14.0 True False False 18d
  13. dns 4.14.0 True False False 5h24m
  14. etcd 4.14.0 True False False 18d
  15. image-registry 4.14.0 True False False 18d
  16. ingress 4.14.0 True False False 18d

*...output omitted...*

* 1. Use the get command to list pods in the openshift-api project. Specify the project with the -n option.
  2. [student@workstation ~]$ **oc get pods -n openshift-apiserver**
  3. NAME READY STATUS RESTARTS AGE

apiserver-68c9485699-ndqlc 2/2 Running 6 18d

* 1. Use the oc status command to retrieve the status of resources in the openshift-authentication project.
  2. [student@workstation ~]$ **oc status -n openshift-authentication**
  3. Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
  4. In project openshift-authentication on server https://api.ocp4.example.com:6443
  5. https://oauth-openshift.apps.ocp4.example.com (passthrough) to pod port 6443 (svc/oauth-openshift)
  6. deployment/oauth-openshift deploys quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:64e6...de42
  7. deployment #7 running for 2 weeks - 1 pod
  8. deployment #6 deployed 2 weeks ago
  9. deployment #4 deployed 2 weeks ago
  10. deployment #5 deployed 2 weeks ago
  11. deployment #3 deployed 2 weeks ago
  12. deployment #2 deployed 2 weeks ago
  13. deployment #1 deployed 2 weeks ago

*...output omitted...*

* 1. Use the explain command to list the description and available fields for services resources.
  2. [student@workstation ~]$ **oc explain services**
  3. KIND: Service
  4. VERSION: v1
  5. DESCRIPTION:
  6. Service is a named abstraction of software service (for example, mysql)
  7. consisting of local port (for example 3306) that the proxy listens on, and
  8. the selector that determines which pods will answer requests sent through
  9. the proxy.
  10. FIELDS:
  11. apiVersion <string>
  12. APIVersion defines the versioned schema of this representation of an
  13. object. Servers should convert recognized schemas to the latest internal
  14. value, and may reject unrecognized values.

*...output omitted...*

* 1. Use the get command to list cluster nodes.
  2. [student@workstation ~]$ **oc get nodes**
  3. NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master,worker 18d v1.27.6+f67aeb3

A single node exists in the cluster.

Inspect Kubernetes Resources

Objectives

* Query, format, and filter attributes of Kubernetes resources.

Kubernetes and OpenShift Resources

Kubernetes uses API resource objects to represent the intended state of everything in the cluster. All administrative tasks require creating, viewing, and changing the API resources. Use the oc api-resources command to view the Kubernetes resources.

**Note**

The oc commands in the examples are identical to the equivalent kubectl commands.

[user@host ~]$ **oc api-resources**

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

componentstatuses cs v1 false ComponentStatus

configmaps cm v1 true ConfigMap

endpoints ep v1 true Endpoints

*...output omitted...*

daemonsets ds apps/v1 true DaemonSet

deployments deploy apps/v1 true Deployment

replicasets rs apps/v1 true ReplicaSet

statefulsets sts apps/v1 true StatefulSet

*...output omitted...*

The SHORTNAME for a component helps to minimize typing long CLI commands. For example, you can use oc get cm instead of oc get configmaps.

The APIVERSION column divides the objects into API groups. The column uses the <API-Group>/<API-Version> format. The API-Group object is blank for Kubernetes core resource objects.

Many Kubernetes resources exist within the context of a Kubernetes namespace. Kubernetes namespaces and OpenShift projects are broadly similar. A 1:1 relationship always exists between a namespace and an OpenShift project.

The KIND is the formal Kubernetes resource schema type.

The oc api-resources command can further filter the output with options that operate on the data.

**Table 2.1. The api-resources Command Options**

| **Option Example** | **Description** |
| --- | --- |
| --namespaced=true | If false, return non-namespaced resources, otherwise return namespaced resources |
| --api-group apps | Limit to resources in the specified API group. Use --api-group='' to show core resources. |
| --sort-by name | If non-empty, sort list of resources using specified field. The field can be either 'name' or 'kind'. |

For example, use the following oc api-resources command to see all the namespaced resources in the apps API group, sorted by name.

[user@host ~]$ **oc api-resources --namespaced=true --api-group apps --sort-by name**

NAME SHORTNAMES APIVERSION NAMESPACED KIND

controllerrevisions apps/v1 true ControllerRevision

daemonsets ds apps/v1 true DaemonSet

deployments deploy apps/v1 true Deployment

replicasets rs apps/v1 true ReplicaSet

statefulsets sts apps/v1 true StatefulSet

Each resource contains fields that identify the resource or that describe the intended configuration of the resource. Use the oc explain command to get information about valid fields for an object. For example, execute the oc explain pod command to get information about possible pod object fields.

[user@host ~]$ **oc explain pod**

KIND: Pod

VERSION: v1

DESCRIPTION:

Pod is a collection of containers that can run on a host. This resource is

created by clients and scheduled onto hosts.

FIELDS:

apiVersion <string>

APIVersion defines the versioned schema of this representation of an

...

kind <string>

Kind is a string value representing the REST resource this object

...

metadata <ObjectMeta>

Standard object's metadata. More info:

...

spec <PodSpec>

Specification of the desired behavior of the pod. More info:

*...output omitted...*

Every Kubernetes resource contains the kind, apiVersion, spec, and status fields. However, when you create an object definition, you do not need to provide the status field. Instead, Kubernetes generates the status field, and it lists information such as runtime status and readiness. The status field is useful for troubleshooting an error or for verifying the current state of a resource.

You can use the YAML path to a field and dot-notation to get information about a particular field. For example, the following oc explain command shows details for the pod.spec fields.

[user@host ~]$ **oc explain pod.spec**

KIND: Pod

VERSION: v1

FIELD: spec <PodSpec>

DESCRIPTION:

Specification of the desired behavior of the pod. More info:

https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

PodSpec is a description of a pod.

FIELDS:

activeDeadlineSeconds <integer>

Optional duration in seconds the pod may be active on the node relative to

*...output omitted...*

The following Kubernetes main resource types can be created and configured by using a YAML or a JSON manifest file, or by using OpenShift management tools:

**Pods (pod)**

Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the primary unit of work for Kubernetes.

**Services (svc)**

Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.

**ReplicaSet (rs)**

Ensure that a specified number of pod replicas are running at any given time.

**Persistent Volumes (pv)**

Define storage areas for Kubernetes pods to use.

**Persistent Volume Claims (pvc)**

Represent a request for storage by a pod. PVCs link a PV to a pod so that its containers can use the provisioned storage, usually by mounting the storage into the container's file system.

**ConfigMaps (cm) and Secrets**

Contain a set of keys and values that other resources can use. ConfigMaps and Secrets centralize configuration values that several resources use. Secrets differ from ConfigMaps in that the values of Secrets are always encoded (not encrypted), and their access is restricted to fewer authorized users.

**Deployment (deploy)**

A representation of a set of containers that are included in a pod, and the deployment strategies to use. A deployment object contains the configuration to apply to all containers of each pod replica, such as the base image, tags, storage definitions, and the commands to execute when the containers start. Although Kubernetes replicas can be created stand-alone in OpenShift, they are usually created by higher-level resources such as deployment controllers.

Red Hat OpenShift Container Platform (RHOCP) adds the following main resource types to Kubernetes:

**BuildConfig (bc)**

Defines a process to execute in the OpenShift project. The OpenShift Source-to-Image (S2I) feature uses a BuildConfig to build a container image from application source code that is stored in a Git repository. A bc works together with a dc to provide an extensible continuous integration and continuous delivery workflows.

**DeploymentConfig (dc)**

OpenShift 4.5 introduced the Deployment resource concept to replace the DeploymentConfig default configuration for pods. Both concepts represent a set of containers that are included in a pod, and the deployment strategies to use.

The Deployment object serves as the improved version of the DeploymentConfig object. Some functional replacements between both objects are as follows:

* Deployment objects no longer support automatic rollback or lifecyle hooks.
* Every change in the pod template that Deployment objects use triggers a new rollout automatically.
* The deployment process of a Deployment object can be paused at any time without affecting the deployer process.
* A Deployment object can have as many active replica sets as the user wants, and can scale down previous replicas. In contrast, the DeploymentConfig object can have only two active replication sets at a time.

Although Deployment objects are the default replacement of the deprecated DeploymentConfig objects in OpenShift 4.12 and later versions, you can still use DeploymentConfig objects if you need a specific feature that they provide, but they are not recommended for new installations. In this case, you must specify the type of object when creating a new application by including the --as-deployment-config flag.

**Routes**

Represent a DNS hostname that the OpenShift router recognizes as an ingress point for applications and microservices.

Structure of Resources

Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the object spec and the object status. The spec object describes the intended state of the resource, and the status object describes the current state. You specify the spec section of the resource when you create the object. Kubernetes controllers continuously update the status of the object throughout the existence of the object. The Kubernetes control plane continuously and actively manages every object's actual state to match the desired state you supplied.

The status field uses a collection of condition resource objects with the following fields.

**Table 2.2. Condition Resource Fields**

| **Field** | **Example** | **Description** |
| --- | --- | --- |
| Type | ContainersReady | The type of the condition |
| Status | False | The state of the condition |
| Reason | RequirementsNotMet | An optional field to provide extra information |
| Message | 2/3 containers are running | An optional textual description for the condition |
| LastTransitionTime | 2023-03-07T18:05:28Z | The last time that conditions were changed |

For example, in Kubernetes, a Deployment object can represent an application that is running on your cluster. When you create a Deployment object, you might configure the deployment spec object to specify that you want three replicas of the application to be running. Kubernetes reads the deployment spec object and starts three instances of your chosen application, and updates the status field to match your spec object. If any of those instances fails, then Kubernetes responds to the difference between the spec and status objects by making a correction, in this case to start a replacement instance.

Other common fields provide base information in addition to the spec and status fields of a Kubernetes object.

**Table 2.3. API Resource Fields**

| **Field** | **Description** |
| --- | --- |
| apiVersion | Identifier of the object schema version. |
| kind | Schema identifier. |
| metadata.name | Creates a label with a name key that other resources in Kubernetes can use to find it. |
| metadata.namespace | The namespace, or the RHOCP project where the resource is. |
| metadata.labels | Key-value pairs that can connect identifying metadata with Kubernetes objects. |

Resources in Kubernetes consist of multiple objects. These objects define the intended state of a resource. When you create or modify an object, you make a persistent record of the intended state. Kubernetes reads the object and modifies the current state accordingly.

All RHOCP and Kubernetes objects can be represented as a JSON or YAML structure. Consider the following pod object in the YAML format:

apiVersion: v1

kind: Pod

metadata:

name: wildfly

namespace: my\_app

labels:

name: wildfly

spec:

containers:

- resources:

limits:

cpu: 0.5

image: quay.io/example/todojee:v1

name: wildfly

ports:

- containerPort: 8080

name: wildfly

env:

- name: MYSQL\_DATABASE

value: items

- name: MYSQL\_USER

value: user1

- name: MYSQL\_PASSWORD

value: mypa55

*...object omitted...*

status:

conditions:

- lastProbeTime: null

lastTransitionTime: "2023-08-19T12:59:22Z"

status: "True"

type: PodScheduled

|  |  |
| --- | --- |
|  | Identifier of the object schema version. |
|  | Schema identifier. In this example, the object conforms to the pod schema. |
|  | Metadata for a given resource, such as annotations, labels, name, and namespace. |
|  | A unique name for a pod in Kubernetes that enables administrators to run commands on it. |
|  | The namespace, or the RHOCP project that the resource resides in. |
|  | Creates a label with a name key that other resources in Kubernetes, usually a service, can use to find it. |
|  | Defines the pod object configuration, or the intended state of the resource. |
|  | Defines the container image name. |
|  | Name of the container inside a pod. Container names are important for oc commands when a pod contains multiple containers. |
|  | A container-dependent attribute to identify the port that the container uses. |
|  | Defines a collection of environment variables. |
|  | Current state of the object. Kubernetes provides this field, which lists information such as runtime status, readiness, and container images. |

Labels are key-value pairs that you define in the .metadata.labels object path, for example:

kind: Pod

apiVersion: v1

metadata:

name: example-pod

labels:

app: example-pod

group: developers

*...object omitted...*

The preceding example contains the app=example-pod and group=developers labels. Developers often use labels to target a set of objects by using the -l or the --selector option. For example, the following oc get command lists pods that contain the group=developers label:

[user@host ~]$ **oc get pod --selector group=developers**

NAME READY STATUS RESTARTS AGE

example-pod-6c9f758574-7fhg 1/1 Running 5 11d

Command Outputs

The kubectl and oc CLI commands provide many output formatting options. By default, many commands display a small subset of the most useful fields for the given resource type in a tabular output. Many commands support a -o wide option that shows additional fields.

**Table 2.4. Tabular Fields**

| **oc get pods** | **oc get pods -o wide** | **Example value** |
| --- | --- | --- |
| NAME | NAME | example-pod |
| READY | READY | 1/1 |
| STATUS | STATUS | Running |
| RESTARTS | RESTARTS | 5 |
| AGE | AGE | 11d |
|  | IP | 10.8.0.60 |
|  | NODE | master01 |
|  | NOMINATED NODE | <none> |
|  | READINESS GATES | <none> |

To view all the fields that are associated with a resource, the describe subcommand shows a detailed description of the selected resource and related resources. You can select a single object by name, or all objects of that type, or provide a name prefix, or a label selector.

For example, the following command first looks for an exact match on the TYPE object and the NAME-PREFIX object. If no such resource exists, then the command outputs details for every resource of that type with a name with a NAME\_PREFIX prefix.

[user@host ~]$ **oc describe *TYPE* *NAME-PREFIX***

The describe subcommand provides detailed human-readable output. However, the format of the describe output might change between versions, and thus is not recommended for script development. Any scripts that rely on the output of the describe subcommand might break after a version update.

Kubernetes provides YAML and JSON-formatted output options that are suitable for parsing or scripting.

YAML Output

The -o yaml option provides a YAML-formatted output that is parsable and still human-readable.

[user@host ~]$ **oc get pods -o yaml**

apiVersion: v1

items:

- apiVersion: v1

kind: Pod

metadata:

annotations:

*...object omitted...*

**Note**

The reference documentation provides a more detailed introduction to YAML.

You can use any tool that can process YAML documents to filter the YAML output for your chosen field. For example, you can use the yq tool at <https://mikefarah.gitbook.io/yq/> to process YAML and JSON files.

The yq processor uses a dot notation to separate field names in a query. The following example pipes the YAML output to the yq command to parse the podIP field.

[user@host ~]$ **oc get pods -o yaml | yq r - 'items[0].status.podIP'**

10.8.0.60

The [0] in the example specifies the first index in the items array.

**Note**

The lab environment includes version 3.3.0 of the yq command, which these examples use. Later versions of the yq command introduce incompatibilities with earlier versions. The content in this course might not work with other versions.

**Note**

Another tool named yq is at <https://kislyuk.github.io/yq/>. The two yq tools are not compatible; commands that are designed for one of them do not work with the other.

JSON Output

Kubernetes uses the JSON format internally to process resource objects. Use the -o json option to view a resource in the JSON format.

[user@host ~]$ **oc get pods -o json**

{

"apiVersion": "v1",

"items": [

{

"apiVersion": "v1",

"kind": "Pod",

"metadata": {

"annotations": {

You can use other tools to process JSON documents, such as the jq tool at <https://jqlang.github.io/jq/>. Similar to the yq processor, use the jq processor and dot notation on the fields to query specific information from the JSON-formatted output.

[user@host ~]$ **oc get pods -o json | jq '.items[0].status.podIP'**

"10.8.0.60"

Alternatively, the example might have used .items[].status.podIP for the query string. The empty brackets instruct the jq tool to query all items.

Custom Output

Kubernetes provides a custom output format that combines the convenience of extracting data via jq styled queries with a tabular output format. Use the -o custom-columns option with comma-separated *<column name> : <jq query string>* pairs.

[user@host ~]$ **oc get pods \**

**-o custom-columns=PodName:".metadata.name",\**

**ContainerName:"spec.containers[].name",\**

**Phase:"status.phase",\**

**IP:"status.podIP",\**

**Ports:"spec.containers[].ports[].containerPort"**

PodName ContainerName Phase IP Ports

myapp-77fb5cd997-xplhz myapp Running 10.8.0.60 <none>

Kubernetes also supports the use of JSONPath expressions. JSONPath is a query language for JSON. JSONPath expressions refer to a JSON data structure; they filter and extract formatted fields from JSON objects.

In the following example, the JSONPath expression uses the range operator to iterate over the list of pods to extract the name of the pod, its IP address, and the assigned ports.

[user@host ~]$ **oc get pods** \

**-o jsonpath='{range .items[]}{"Pod Name: "}{.metadata.name}**

**{"IP: "}{.status.podIP}**

**{"Ports: "}{.spec.containers[].ports[].containerPort}{"\n"}{end}'**

Pod Name: myapp-77fb5cd997-xplhz

IP: 10.8.0.60

Ports:

You can customize the format of the output with Go templates, which the Go programming language uses. Use the -o go-template option followed by a Go template, where Go expressions are inside double braces, {{ }}.

[user@host ~]$ **oc get pods** \

**-o go-template='{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}'**

myapp-77fb5cd997-xplhz

Guided Exercise: Inspect Kubernetes Resources

Verify the state of an OpenShift cluster by querying its recognized resource types, their schemas, and extracting information from Kubernetes resources that are related to to OpenShift cluster services.

**Outcomes**

* List and explain the supported API resources for a cluster.
* Identify resources from specific API groups.
* Format command outputs in the YAML and JSON formats.
* Use filters to parse command outputs.
* Use JSONPath and custom columns to extract information from resources.

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 accessible and that all resources are available for this exercise. It also creates a myapp application in the cli-resources project.

[student@workstation ~]$ **lab start cli-resources**

**Instructions**

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

*...output omitted...*

* 1. Set the cli-resources project as the active project.
  2. [student@workstation ~]$ **oc project cli-resources**

*...output omitted...*

1. List the available cluster resource types with the api-resources command. Then, use filters to list namespaced and non-namespaced resources.
   1. List the available resource types with the api-resources command.
   2. [student@workstation ~]$ **oc api-resources**
   3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
   4. bindings v1 true Binding
   5. componentstatuses cs v1 false ComponentStatus
   6. configmaps cm v1 true ConfigMap
   7. endpoints ep v1 true Endpoints
   8. events ev v1 true Event
   9. limitranges limits v1 true LimitRange
   10. namespaces ns v1 false Namespace
   11. nodes no v1 false Node
   12. persistentvolumeclaims pvc v1 true PersistentVolumeClaim
   13. persistentvolumes pv v1 false PersistentVolume
   14. pods po v1 true Pod
   15. *...output omitted...*
   16. controllerrevisions apps/v1 true ControllerRevision
   17. daemonsets ds apps/v1 true DaemonSet
   18. *...output omitted...*
   19. cronjobs cj batch/v1 true CronJob
   20. jobs batch/v1 true Job

*...output omitted...*

The api-resources command prints the supported API resources, including resource names, available shortnames, and the API versions.

You can use the APIVERSIONS field to determine which API group provides the resource. The field lists the group followed by the API version of the resource. For example, the jobs resource type is provided by the batch API group, and v1 is the API version of the resource.

* 1. Use the --namespaced option to limit the output of the api-resources command to namespaced resources.

Then, determine the number of available namespaced resources. Use the -o name option to list the resource names, and then pipe the output to the wc -l command.

[student@workstation ~]$ **oc api-resources --namespaced**

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

configmaps cm v1 true ConfigMap

endpoints ep v1 true Endpoints

events ev v1 true Event

limitranges limits v1 true LimitRange

persistentvolumeclaims pvc v1 true PersistentVolumeClaim

pods po v1 true Pod

podtemplates v1 true PodTemplate

replicationcontrollers rc v1 true ReplicationController

resourcequotas quota v1 true ResourceQuota

secrets v1 true Secret

*...output omitted...*

[student@workstation ~]$ **oc api-resources --namespaced -o name | wc -l**

113

The cluster has 113 namespaced cluster resource types, such as the pods, deployments, and services resources.

* 1. Limit the output of the api-resources command to non-namespaced resources.

Then, determine the number of available non-namespaced resources. To list the resource names, use the -o name option and then pipe the output to the wc -l command.

[student@workstation ~]$ **oc api-resources --namespaced=false**

NAME SHORTNAMES APIVERSION ...

componentstatuses cs v1 ...

namespaces ns v1 ...

nodes no v1 ...

persistentvolumes pv v1 ...

mutatingwebhookconfigurations admissionregistration.k8s.io/v1 ...

validatingwebhookconfigurations admissionregistration.k8s.io/v1 ...

customresourcedefinitions crd,crds apiextensions.k8s.io/v1 ...

*...output omitted...*

[student@workstation ~]$ **oc api-resources --namespaced=false -o name | wc -l**

118

The cluster has 118 non-namespaced cluster resource types, such as the nodes, images, and project resources.

1. Identify and explain the available cluster resource types that the core API group provides. Then, describe a resource from the core API group in the cli-resources project.
   1. Filter the output of the api-resources command to show only resources from the core API group. Use the --api-group option and set '' as the value.
   2. [student@workstation ~]$ **oc api-resources --api-group ''**
   3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
   4. bindings v1 true Binding
   5. componentstatuses cs v1 false ComponentStatus
   6. configmaps cm v1 true ConfigMap
   7. endpoints ep v1 true Endpoints
   8. events ev v1 true Event
   9. limitranges limits v1 true LimitRange
   10. namespaces ns v1 false Namespace
   11. nodes no v1 false Node
   12. persistentvolumeclaims pvc v1 true PersistentVolumeClaim
   13. persistentvolumes pv v1 false PersistentVolume
   14. pods po v1 true Pod
   15. podtemplates v1 true PodTemplate
   16. replicationcontrollers rc v1 true ReplicationController
   17. resourcequotas quota v1 true ResourceQuota
   18. secrets v1 true Secret
   19. serviceaccounts sa v1 true ServiceAccount

services svc v1 true Service

The core API group provides several resource types, such as nodes, events, and pods.

* 1. Use the explain command to list a description and the available fields for the pods resource type.
  2. [student@workstation ~]$ **oc explain pods**
  3. KIND: Pod
  4. VERSION: v1
  5. DESCRIPTION:
  6. Pod is a collection of containers that can run on a host. This resource is
  7. created by clients and scheduled onto hosts.
  8. FIELDS:
  9. apiVersion <string>
  10. APIVersion defines the versioned schema of this representation of an
  11. object. Servers should convert recognized schemas to the latest internal
  12. value, and may reject unrecognized values. More info:

*...output omitted...*

* 1. List all pods in the cli-resources project.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE

myapp-54fcdcd9d7-2h5vx 1/1 Running 0 4m25s

A single pod exists in the cli-resources project. The pod name might differ in your output.

* 1. Use the describe command to view the configuration and events for the pod. Specify the pod name from the previous step.
  2. [student@workstation ~]$ **oc describe pod *myapp-54fcdcd9d7-2h5vx***
  3. Name: myapp-54fcdcd9d7-2h5vx
  4. Namespace: cli-resources
  5. *...output omitted...*
  6. Status: Running
  7. IP: 10.8.0.127
  8. IPs:
  9. IP: 10.8.0.127
  10. Controlled By: ReplicaSet/myapp-54fcdcd9d7
  11. Containers:
  12. myapp:
  13. Container ID: cri-o://e0da...669d
  14. Image: registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
  15. Image ID: registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:91ad...fd83
  16. *...output omitted...*
  17. Limits:
  18. cpu: 500m
  19. memory: 128Mi
  20. Requests:
  21. cpu: 500m
  22. memory: 128Mi
  23. Environment: <none>
  24. *...output omitted...*
  25. Events:
  26. Type Reason Age From Message
  27. ---- ------ ---- ---- -------
  28. Normal Scheduled 10m default-scheduler Successfully assigned cli-resources/myapp-54fcdcd9d7-2h5vx to master01

*....output omitted...*

* 1. Retrieve the details of the pod in a structured format. Use the get command and specify the output as the YAML format. Compare the results of the describe command versus the get command.
  2. [student@workstation ~]$ **oc get pod *myapp-54fcdcd9d7-2h5vx* -o yaml**
  3. apiVersion: v1
  4. kind: Pod
  5. metadata:
  6. annotations:
  7. *...output omitted...*
  8. labels:
  9. app: myapp
  10. pod-template-hash: 54fcdcd9d7
  11. name: myapp-54fcdcd9d7-2h5vx
  12. namespace: cli-resources
  13. *...output omitted...*
  14. spec:
  15. containers:
  16. - image: registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
  17. imagePullPolicy: Always
  18. name: myapp
  19. resources:
  20. limits:
  21. cpu: 500m
  22. memory: 128Mi
  23. requests:
  24. cpu: 500m
  25. memory: 128Mi

*...output omitted...*

Using a structured format with the get command provides more details about a resource than the describe command.

1. Identify and explain the available cluster resource types that the Kubernetes apps API group provides. Then, describe a resource from the apps API group in the cli-resources project.
   1. List the resource types that the apps API group provides.
   2. [student@workstation ~]$ **oc api-resources --api-group apps**
   3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
   4. controllerrevisions apps/v1 true ControllerRevision
   5. daemonsets ds apps/v1 true DaemonSet
   6. deployments deploy apps/v1 true Deployment
   7. replicasets rs apps/v1 true ReplicaSet

statefulsets sts apps/v1 true StatefulSet

* 1. Use the explain command to list a description and fields for the deployments resource type.
  2. [student@workstation ~]$ **oc explain deployments**
  3. GROUP: apps
  4. KIND: Deployment
  5. VERSION: apps/v1
  6. DESCRIPTION:
  7. Deployment enables declarative updates for Pods and ReplicaSets.
  8. FIELDS:
  9. apiVersion <string>
  10. APIVersion defines the versioned schema of this representation of an
  11. object. Servers should convert recognized schemas to the latest internal
  12. value, and may reject unrecognized values. More info:

*...output omitted...*

* 1. Use the get command to identify any deployment resources in the cli-resources project.
  2. [student@workstation ~]$ **oc get deploy**
  3. NAME READY UP-TO-DATE AVAILABLE AGE

myapp 1/1 1 1 25m

* 1. The myapp deployment exists in the cli-resources project. Use the get command and the -o wide option to identify the container name and the container image in the deployment.
  2. [student@workstation ~]$ **oc get deploy myapp -o wide**
  3. NAME ... CONTAINERS IMAGES SELECTOR

myapp ... myapp registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 app=myapp

The myapp deployment uses the registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 container image for the myapp container.

* 1. Describe the myapp deployment to view more details about the resource.
  2. [student@workstation ~]$ **oc describe deployment myapp**
  3. Name: myapp
  4. Namespace: cli-resources
  5. CreationTimestamp: Wed, 01 Mar 2023 18:41:39 -0500
  6. Labels: my-app
  7. Annotations: deployment.kubernetes.io/revision: 1
  8. Selector: app=myapp
  9. Replicas: 1 desired | 1 updated | 1 total | 1 available | 0 unavailable
  10. StrategyType: RollingUpdate
  11. MinReadySeconds: 0
  12. RollingUpdateStrategy: 25% max unavailable, 25% max surge
  13. Pod Template:
  14. Labels: app=myapp
  15. Containers:
  16. myapp:
  17. Image: registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
  18. Port: 8080
  19. Host Port: 8080
  20. Limits:
  21. cpu: 500m
  22. memory: 128Mi
  23. Environment: <none>
  24. Mounts: <none>
  25. Volumes: <none>
  26. Conditions:
  27. Type Status Reason
  28. ---- ------ ------
  29. Available True MinimumReplicasAvailable
  30. Progressing True NewReplicaSetAvailable
  31. OldReplicaSets: <none>
  32. NewReplicaSet: myapp-54fcdcd9d7 (1/1 replicas created)
  33. Events:
  34. Type Reason Age From Message
  35. ---- ------ ---- ---- -------

Normal ScalingReplicaSet 30m deployment-controller Scaled up replica set myapp-54fcdcd9d7 to 1

1. Identify and explain the available cluster resource types that the OpenShift configuration API group provides. Then, describe a resource from the OpenShift configuration API group.
   1. List the resource types that the OpenShift configuration API group provides.
   2. [student@workstation ~]$ **oc api-resources --api-group config.openshift.io**
   3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
   4. apiservers config.openshift.io/v1 false APIServer
   5. authentications config.openshift.io/v1 false Authentication
   6. builds config.openshift.io/v1 false Build
   7. clusteroperators co config.openshift.io/v1 false ClusterOperator
   8. clusterversions config.openshift.io/v1 false ClusterVersion
   9. consoles config.openshift.io/v1 false Console
   10. dnses config.openshift.io/v1 false DNS
   11. featuregates config.openshift.io/v1 false FeatureGate
   12. imagecontentpolicies config.openshift.io/v1 false ImageContentPolicy
   13. imagedigestmirrorsets idms config.openshift.io/v1 false ImageDigestMirrorSet
   14. images config.openshift.io/v1 false Image
   15. imagetagmirrorsets itms config.openshift.io/v1 false ImageTagMirrorSet
   16. infrastructures config.openshift.io/v1 false Infrastructure
   17. ingresses config.openshift.io/v1 false Ingress
   18. networks config.openshift.io/v1 false Network
   19. nodes config.openshift.io/v1 false Node
   20. oauths config.openshift.io/v1 false OAuth
   21. operatorhubs config.openshift.io/v1 false OperatorHub
   22. projects config.openshift.io/v1 false Project
   23. proxies config.openshift.io/v1 false Proxy

schedulers config.openshift.io/v1 false Scheduler

The config.openshift.io API group provides multiple, non-namespaced resource types.

* 1. Use the explain command to list a description and fields for the projects resource type.
  2. [student@workstation ~]$ **oc explain projects**
  3. GROUP: project.openshift.io
  4. KIND: Project
  5. VERSION: v1
  6. DESCRIPTION:
  7. Projects are the unit of isolation and collaboration in OpenShift. A
  8. project has one or more members, a quota on the resources that the project
  9. may consume, and the security controls on the resources in the project.
  10. Within a project, members may have different roles - project administrators
  11. can set membership, editors can create and manage the resources, and
  12. viewers can see but not access running containers. In a normal cluster
  13. project administrators are not able to alter their quotas - that is
  14. restricted to cluster administrators.
  15. Listing or watching projects will return only projects the user has the
  16. reader role on.

*...output omitted...*

* 1. Describe the cli-resources project.
  2. [student@workstation ~]$ **oc describe project cli-resources**
  3. Name: cli-resources
  4. Created: 10 minutes ago
  5. Labels: kubernetes.io/metadata.name=cli-resources
  6. pod-security.kubernetes.io/audit=restricted
  7. pod-security.kubernetes.io/audit-version=v1.24
  8. pod-security.kubernetes.io/warn=restricted
  9. pod-security.kubernetes.io/warn-version=v1.24
  10. Annotations: openshift.io/description=
  11. openshift.io/display-name=
  12. openshift.io/requester=system:admin
  13. openshift.io/sa.scc.mcs=s0:c26,c25
  14. openshift.io/sa.scc.supplemental-groups=1000710000/10000
  15. openshift.io/sa.scc.uid-range=1000710000/10000
  16. Display Name: <none>
  17. Description: <none>
  18. Status: Active
  19. Node Selector: <none>
  20. Quota: <none>

Resource limits: <none>

* 1. Retrieve more details of the cli-resources project. Use the get command, and format the output to use JSON.
  2. [student@workstation ~]$ **oc get project cli-resources -o json**
  3. {
  4. "apiVersion": "project.openshift.io/v1",
  5. "kind": "Project",
  6. "metadata": {
  7. *...output omitted...*.
  8. "labels": {
  9. "kubernetes.io/metadata.name": "cli-resources",
  10. "pod-security.kubernetes.io/audit": "restricted",
  11. "pod-security.kubernetes.io/audit-version": "v1.24",
  12. "pod-security.kubernetes.io/warn": "restricted",
  13. "pod-security.kubernetes.io/warn-version": "v1.24"
  14. },
  15. "name": "cli-resources",
  16. "resourceVersion": "705313",
  17. "uid": "53cbbe45-31ea-4b41-93a9-4ba5c2c4c1f3"
  18. },
  19. *...output omitted...*
  20. "status": {
  21. "phase": "Active"
  22. }

}

The get command provides additional details, such as the kind and apiVersion attributes, of the project resource.

1. Verify the cluster status by inspecting cluster services. Format command outputs by using filters.
   1. Retrieve the list of pods for the Etcd operator. The Etcd operator is available in the openshift-etcd namespace. Specify the namespace with the --namespace or -n option.
   2. [student@workstation ~]$ **oc get pods -n openshift-etcd**

Error from server (Forbidden): pods is forbidden: User "developer" cannot list resource "pods" in API group "" in the namespace "openshift-etcd"

The developer user cannot access resources in the openshift-etcd namespace. Regular cluster users, such as the developer user, cannot query resources in the openshift- namespaces.

Log in as the admin user with the redhatocp password. Then, retrieve the list of pods in the openshift-etcd namespace.

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

Login successful

*...output omitted...*

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

NAME READY STATUS RESTARTS AGE

etcd-master01 4/4 Running 36 25d

installer-3-master01 0/1 Completed 0 25d

* 1. Retrieve the image of the etcd-master01 pod in the openshift-etcd namespace. Use filters to limit the output to the .spec.containers attribute of the pod to get the first element. Compare the outputs of the JSONPath, the jq filters, and the Go template format.
  2. [student@workstation ~]$ **oc get pods etcd-master01 -n openshift-etcd \**
  3. **-o=jsonpath='{.spec.containers[0].image}'{"\n"}**

quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:24d9...2020

[student@workstation ~]$ **oc get pods -n openshift-etcd etcd-master01 \**

**-o json | jq .spec.containers[0].image**

"quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:24d9...2020"

[student@workstation ~]$ **oc get pods -n openshift-etcd etcd-master01 \**

**-o go-template='{{(index .spec.containers 0).image}}'**

quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:24d9b9d9d7fadacbc505c849a1e4b390b2f0fcd452ad851b7cce21e8cf

* 1. Retrieve the condition status of the prometheus-k8s-0 pod in the openshift-monitoring namespace. Configure the output to use the YAML format, and then filter the output with the yq filter.
  2. [student@workstation ~]$ **oc get pods -n openshift-monitoring prometheus-k8s-0 \**
  3. **-o yaml | yq r - 'status.conditions'**
  4. - lastProbeTime: null
  5. lastTransitionTime: "2023-12-12T18:07:17Z"
  6. status: "True"
  7. type: Initialized
  8. - lastProbeTime: null
  9. lastTransitionTime: "2023-12-15T18:07:45Z"
  10. status: "True"
  11. type: Ready
  12. - lastProbeTime: null
  13. lastTransitionTime: "2023-12-15T18:07:45Z"
  14. status: "True"
  15. type: ContainersReady
  16. - lastProbeTime: null
  17. lastTransitionTime: "2023-12-12T22:39:52Z"
  18. status: "True"

type: PodScheduled

The r - option tells the yq command to read the standard input (STDIN) for the YAML output of the get command.

* 1. Use the get command to retrieve detailed information for the pods in the openshift-storage namespace. Use the YAML format and custom columns to filter the output according to the following table:

| **Column title** | **Object** |
| --- | --- |
| PodName | metadata.name |
| ContainerName | spec.containers[].name |
| Phase | status.phase |
| IP | status.podIP |
| Ports | spec.containers[].ports[].containerPort |

* 1. [student@workstation ~]$ **oc get pods -n openshift-storage \**
  2. **-o custom-columns=PodName:".metadata.name",\**
  3. **ContainerName:"spec.containers[].name",\**
  4. **Phase:"status.phase",\**
  5. **IP:"status.podIP",\**
  6. **Ports:"spec.containers[].ports[].containerPort"**
  7. PodName ContainerName Phase IP Ports
  8. lvms-operator-7fcd897cb-... manager Running 10.8.0.97 9443
  9. topolvm-controller-.... topolvm-controller Running 10.8.0.98 9808
  10. topolvm-node-9spzf lvmd Running 10.8.0.100 <none>
  11. vg-manager-z8g5k vg-manager Running 10.8.0.101 <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 cli-resources**

Assess the Health of an OpenShift Cluster

Objectives

* Query the health of essential cluster services and components.

Query Operator Conditions

Operators are important components of Red Hat OpenShift Container Platform (RHOCP). Operators automate the required tasks to maintain a healthy RHOCP cluster that would otherwise require human intervention. Operators are the preferred method of packaging, deploying, and managing services on the control plane.

Operators integrate with Kubernetes APIs and CLI tools such as kubectl and oc commands. Operators provide the means of monitoring applications, performing health checks, managing over-the-air (OTA) updates, and ensuring that applications remain in your specified state.

Because CRI-O and the Kubelet run on every node, almost every other cluster function can be managed on the control plane by using Operators. Components that are added to the control plane by using operators include critical networking and credential services.

Operators in RHOCP are managed by two different systems, depending on the purpose of the operator.

**Cluster Version Operator (CVO)**

Cluster operators perform cluster functions. These operators are installed by default, and the CVO manages them.

Cluster operators use a Kubernetes kind value of clusteroperators, and thus can be queried via oc or kubectl commands. As a user with the cluster-admin role, use the oc get clusteroperators command to list all the cluster operators.

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

NAME VERSION AVAILABLE PROGRESSING DEGRADED SINCE MESSAGE

authentication 4.14.0 True False False 3d1h

baremetal 4.14.0 True False False 38d

cloud-controller-manager 4.14.0 True False False 38d

cloud-credential 4.14.0 True False False 38d

cluster-autoscaler 4.14.0 True False False 38d

config-operator 4.14.0 True False False 38d

console 4.14.0 True False False 38d

*...output omitted...*

For more details about a cluster operator, use the describe clusteroperators *operator-name* command to view the field values that are associated with the operator, including the current status of the operator. The describe command provides a human-readable output format for a resource. As such, the output format might change with an RHOCP version update.

For an output format that is less likely to change with a version update, use one of the -o output options of the get command. For example, use the following oc get clusteroperators command for the YAML-formatted output details for the dns operator.

[user@host ~]$ **oc get clusteroperators dns -o yaml**

apiVersion: config.openshift.io/v1

kind: ClusterOperator

metadata:

annotations:

*...output omitted...*

status:

conditions:

- lastTransitionTime: "2023-03-20T13:55:21Z"

message: DNS "default" is available.

reason: AsExpected

status: "True"

type: Available

*...output omitted...*

relatedObjects:

- group: ""

name: openshift-dns-operator

resource: namespaces

*...output omitted...*

versions:

- name: operator

version: 4.14.0

*...output omitted...*

**Operator Lifecycle Manager (OLM) Operators**

Optional add-on operators that the OLM manages can be made accessible for users to run in their applications.

As a user with the cluster-admin role, use the get operators command to list all the add-on operators.

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

NAME AGE

lvms-operator.openshift-storage 34d

metallb-operator.metallb-system 34d

You can likewise use the describe and get commands to query details about the fields that are associated with the add-on operators.

Operators use one or more pods to provide cluster services. You can find the namespaces for these pods under the relatedObjects section of the detailed output for the operator. As a user with a cluster-admin role, use the -n *namespace* option on the get pod command to view the pods. For example, use the following get pods command to retrieve the list of pods in the openshift-dns-operator namespace.

[user@host~]$ **oc get pods -n openshift-dns-operator**

NAME READY STATUS RESTARTS AGE

dns-operator-64688bfdd4-8zklh 2/2 Running 38 38d

Use the -o yaml or -o json output formats to view or analyze more details about the pods. The resource conditions, which are found in the status for the resource, track the current state of the resource object. The following example uses the jq processor to extract the status values from the JSON output details for the dns pod.

[user@host~]$ **oc get pod -n openshift-dns-operator \**

**dns-operator-64688bfdd4-8zklh -o json | jq .status**

{

"conditions": [

{

"lastProbeTime": null,

"lastTransitionTime": "2023-02-09T21:24:50Z",

"status": "True",

"type": "Initialized"

},

*...output omitted...*

In addition to listing the pods of a namespace, you can also use the --show-labels option of the get command to print the labels used by the pods. The following example retrieves the pods and their labels in the openshift-etcd namespace.

[user@host~]$ **oc get pods -n openshift-etcd --show-labels**

NAME READY STATUS RESTARTS AGE LABELS

etcd-master01 4/4 Running 68 35d app=etcd,etcd=true,k8s-app=etcd,revision=3

installer-3-master01 0/1 Completed 0 35d app=installer

Examining Cluster Metrics

Another way to gauge the health of an RHOCP cluster is to examine the compute resource usage of cluster nodes and pods. The oc adm top command provides this information. For example, to list the total memory and CPU usage of all pods in the cluster, you can use the --sum option with the command to print the sum of the resource usage.

[user@host~]$ **oc adm top pods -A --sum**

NAMESPACE NAME CPU(cores) MEMORY(bytes)

metallb-system controller-...-ddr8v 0m 57Mi

metallb-system metallb-...-n2zsv 0m 48Mi

*...output omitted...*

openshift-storage topolvm-node-9spzf 0m 68Mi

openshift-storage vg-manager-z8g5k 0m 23Mi

------ --------

428m 10933Mi

The -A option shows pods from all namespaces. Use the -n *namespace* option to filter the results to show the pods in a single namespace. Use the --containers option to display the resource usage of containers within a pod. For example, use the following command to list the resource usage of the containers in the etcd-master01 pod in the openshift-etcd namespace.

[user@host~]$ **oc adm top pods etcd-master01 -n openshift-etcd --containers**

POD NAME CPU(cores) MEMORY(bytes)

etcd-master01 POD 0m 0Mi

etcd-master01 etcd 71m 933Mi

etcd-master01 etcd-metrics 6m 32Mi

etcd-master01 etcd-readyz 4m 66Mi

etcd-master01 etcdctl 0m 0Mi

Viewing Cluster Metrics

The OpenShift web console incorporates graphs to visualize cluster and resource analytics. Cluster administrators and users with either the view or the cluster-monitoring-view cluster role can access the **Home** → **Overview** page. The Overview page displays a collection of cluster-wide metrics, and provides a high-level view of the overall health of the cluster.

The Overview page displays the following metrics:

* Current cluster capacity based on CPU, memory, storage, and network usage
* A time-series graph of total CPU, memory, and disk usage
* The ability to display the top consumers of CPU, memory, and storage

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

For any of the listed resources in the **Cluster Utilization** section, administrators can click the link for current resource usage. The link displays a window with a breakdown of top consumers for that resource. Top consumers can be sorted by project, by pod, or by node. The list of top consumers can be useful for identifying problematic pods or nodes. For example, a pod with an unexpected memory leak might appear at the top of the list.

Viewing Project Metrics

The **Project Details** page displays metrics that provide an overview of the resources that are used within the scope of a specific project. The **Utilization** section displays usage information about resources, such as CPU and memory, along with the ability to display the top consumers for each resource.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

All metrics are pulled from Prometheus. Click any graph to navigate to the **Metrics** page. View the executed query, and inspect the data further.

If a resource quota is created for the project, then the current project request and limits appear on the **Project Details** page.

Viewing Resource Metrics

When troubleshooting, it is often useful to view metrics at a smaller granularity than for the entire cluster or project. The **Pod Details** page displays time-series graphs of the CPU, memory, and file system usage for a specific pod. A sudden change in these critical metrics, such as a CPU spike caused by high load, is visible on this page.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 2.9: Time-series graphs showing various metrics for a pod

Performing Prometheus Queries in the Web Console

The Prometheus UI is a feature-rich tool for visualizing metrics and configuring alerts. The OpenShift web console provides an interface for executing Prometheus queries directly from the web console.

To perform a query, navigate to **Observe** → **Metrics**, enter a Prometheus Query Language expression in the text field, and click **Run Queries**. The results of the query are displayed as a time-series graph:

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

Figure 2.10: Using a Prometheus query to display a time-series graph

**Note**

The Prometheus Query Language is not discussed in detail in this course. Refer to the references section for a link to the official documentation.

Query Cluster Events and Alerts

Some developers consider OpenShift logs to be too low-level, thus making troubleshooting difficult. Fortunately, RHOCP provides a high-level logging and auditing facility called *events*. Kubernetes generates event objects in response to state changes in cluster objects, such as nodes, pods, and containers. Events signal significant actions, such as starting a container or destroying a pod.

To read events, use the get events command. The command lists the events for the current RHOCP project (namespace). You can display the events for a different project by adding the -n *namespace* option to the command. To list the events for all the projects, use the -A (or --all-namespaces) option.

**Note**

To sort the events by time, add the --sort-by .metadata.creationTimestamp option to the oc get events command.

The following get events command prints events in the openshift-kube-controller-manager namespace.

[user@host~]$ **oc get events -n openshift-kube-controller-manager**

LAST SEEN TYPE REASON OBJECT MESSAGE

12m Normal CreatedSCCRanges pod/kube-controller-manager-master01 created SCC...

You can use the describe pod *pod-name* command to further narrow the results to a single pod. For example, to retrieve only the events that relate to a mysql pod, you can refer to the Events field from the output of the oc describe pod mysql command:

[user@host~]$ **oc describe pod mysql**

*...output omitted...*

Events:

FirstSeen LastSeen Count From Reason Message

Wed, 10 ... Wed, 10 ... 1 {scheduler } scheduled Successfully as...

*...output omitted...*

Kubernetes Alerts

RHOCP includes a monitoring stack, which is based on the Prometheus open source project. The monitoring stack is configured to monitor the core RHOCP cluster components, by default. You can optionally configure the monitoring stack also to monitor user projects.

The components of the monitoring stack are installed in the openshift-monitoring namespace. The Prometheus Operator in the openshift-monitoring namespace creates, configures, and manages platform Prometheus and Alertmanager instances. An Alertmanager pod in the openshift-monitoring namespace receives alerts from Prometheus. Alertmanager can also send alerts to external notification systems.

Use the following get all command to display a list of all resources, their status, and their types in the openshift-monitoring namespace.

[user@host~]$ **oc get all -n openshift-monitoring --show-kind**

NAME READY STATUS RESTARTS AGE

pod/alertmanager-main-0 6/6 Running 85 34d

pod/cluster-monitoring-operator-56b769b58f-dtmqj 2/2 Running 34 35d

pod/kube-state-metrics-75455b796c-8q28d 3/3 Running 51 35d

*...output omitted...*

The alertmanager-main-0 pod is the Alertmanager for the cluster. The following logs command shows the logs of the alertmanager-main-0 pod, which displays the received messages from Prometheus.

[user@host~]$ **oc logs alertmanager-main-0 -n openshift-monitoring**

ts=2023-03-16T14:21:50.479Z caller=main.go:231 level=info msg="Starting Alertmanager" version="(version=0.24.0, branch=rhaos-4.14-rhel-8, revision=519cbb87494d2830821a0da0a657af69d852c93b)"

ts=2023-03-16T14:21:50.479Z caller=main.go:232 level=info build\_context="(go=go1.19.4, user=root@232132c11c68, date=20230105-00:26:49)"

ts=2023-03-16T14:21:50.527Z caller=coordinator.go:113 level=info component=configuration msg="Loading configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml

*...output omitted...*

Check Node Status

RHOCP clusters can have several components, including at least one control plane and at least one compute node. The two components can occupy a single node. The following oc command, or the matching kubectl command, can display the overall health of all cluster nodes.

[user@host~]$ **oc cluster-info**

The oc cluster-info output is high-level, and can verify that the cluster nodes are running. For a more detailed view into the cluster nodes, use the get nodes command.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master,worker 35d v1.27.6+f67aeb3

The example shows a single master01 node with multiple roles. The STATUS value of Ready means that this node is healthy and can accept new pods. A STATUS value of NotReady means that a condition triggered the NotReady status and the node is not accepting new pods.

As with any other RHOCP resource, you can drill down into further details of the node resource with the describe node *node-name* command. For parsable output of the same information, use the -o json or the -o yaml output options with the get node *node-name* command. For more information about using and parsing these output formats, see [the section called “ Inspect Kubernetes Resources ”](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s03/c2358540-87d5-48de-b49e-6f23bdcd629c).

The output of the get nodes *node-name* command with the -o json or -o yaml option is long. The following examples use the -jsonpath option or the jq processor to parse the get node *node-name* command output.

[user@host~]$ **oc get node master01 -o jsonpath=\**

**\*'{"Allocatable:\n"}{.status.allocatable}{"\n\n"}**

**{"Capacity:\n"}{.status.capacity}{"\n"}'**

Allocatable:

{"cpu":"7500m","ephemeral-storage":"114396791822","hugepages-1Gi":"0",

"hugepages-2Mi":"0","memory":"19380692Ki","pods":"250"}

Capacity:

{"cpu":"8","ephemeral-storage":"125293548Ki","hugepages-1Gi":"0",

"hugepages-2Mi":"0","memory":"20531668Ki","pods":"250"}

The JSONPath expression in the previous command extracts the allocatable and capacity measures for the master01 node. These measures help to understand the available resources on a node.

View the status object of a node to understand the current health of the node.

[user@host~]$ **oc get node master01 -o json | jq '.status.conditions'**

[

{

"lastHeartbeatTime": "2023-03-22T16:34:57Z",

"lastTransitionTime": "2023-02-23T20:35:15Z",

"message": "kubelet has sufficient memory available",

"reason": "KubeletHasSufficientMemory",

"status": "False",

"type": "MemoryPressure"

},

{

"lastHeartbeatTime": "2023-03-22T16:34:57Z",

"lastTransitionTime": "2023-02-23T20:35:15Z",

"message": "kubelet has no disk pressure",

"reason": "KubeletHasNoDiskPressure",

"status": "False",

"type": "DiskPressure"

},

{

"lastHeartbeatTime": "2023-03-22T16:34:57Z",

"lastTransitionTime": "2023-02-23T20:35:15Z",

"message": "kubelet has sufficient PID available",

"reason": "KubeletHasSufficientPID",

"status": "False",

"type": "PIDPressure"

},

{

"lastHeartbeatTime": "2023-03-22T16:34:57Z",

"lastTransitionTime": "2023-02-23T20:35:15Z",

"message": "kubelet is posting ready status",

"reason": "KubeletReady",

"status": "True",

"type": "Ready"

}

]

|  |  |
| --- | --- |
|  | If the status of the MemoryPressure condition is true, then the node is low on memory. |
|  | If the status of the DiskPressure condition is true, then the disk capacity of the node is low. |
|  | If the status of the PIDPressure condition is true, then too many processes are running on the node. |
|  | If the status of the Ready condition is false, then the node is not healthy and is not accepting pods. |

More conditions indicate other potential problems with a node.

**Table 2.5. Possible Node Conditions**

| **Condition** | **Description** |
| --- | --- |
| OutOfDisk | If true, then the node has insufficient free space on the node for adding new pods. |
| NetworkUnavailable | If true, then the network for the node is not correctly configured. |
| NotReady | If true, then one of the underlying components, such as the container runtime or network, is experiencing issues or is not yet configured. |
| SchedulingDisabled | Pods cannot be scheduled for placement on the node. |

To gain deeper insight into a given node, you can view the logs of processes that run on the node. A cluster administrator can use the oc adm node-logs command to view node logs. Node logs might contain sensitive output, and thus are limited to privileged node administrators. Use oc adm node-logs *node-name* to filter the logs to a single node.

The oc adm node-logs command has other options to further filter the results.

**Table 2.6. Filters for oc adm node-logs**

| **Option Example** | **Description** |
| --- | --- |
| --role master | Use the --role option to filter the output to nodes with a specified role. |
| -u kubelet | The -u option filters the output to a specified unit. |
| --path=cron | The --path option filters the output to a specific process under the /var/logs directory. |
| --tail 1 | Use --tail *x* to limit output to the last x log entries. |

Use oc adm node-logs --help for a complete list of command options.

For example, to retrieve the most recent log entry for the crio service on the master01 node, you can use the following command.

[user@host~]$ **oc adm node-logs master01 -u crio --tail 1**

-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Fri 2023-03-17 15:11:43 UTC. --

Mar 17 06:16:09.519642 master01 crio[2987]: time="2023-03-17 06:16:09.519474755Z" level=info msg="Image status:

&ImageStatusResponse{Image:&Image{Id:6ef8...79ce,RepoTags:[],RepoDigests:

*...output omitted...*

When you create a pod with the CLI, the oc or kubectl command is sent to the apiserver service, which then validates the command. The scheduler service reads the YAML or JSON pod definition, and then assigns pods to compute nodes. Each compute node runs a kubelet service that converts the pod manifest to one or more containers in the CRI-O container runtime.

Each compute node must have an active kubelet service and an active crio service. To verify the health of these services, first start a debug session on the node by using the debug command.

[user@host~]$ **oc debug node/*node-name***

Replace the *node-name* value with the name of your node.

**Note**

The debug command is covered in greater detail in a later section.

Within the debug session, change to the /host root directory so that you can run binaries in the host's executable path.

sh-4.4# **chroot /host**

Then, use the systemctl is-active calls to confirm that the services are active.

sh-4.4# **for SERVICES in kubelet crio; do echo ---- $SERVICES ---- ;**

**systemctl is-active $SERVICES ; echo ""; done**

---- kubelet ----

active

---- crio ----

active

For more details about the status of a service, use the systemctl status command.

sh-4.4# **systemctl status kubelet**

● kubelet.service - Kubernetes Kubelet

Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: disabled)

Drop-In: /etc/systemd/system/kubelet.service.d

└─01-kubens.conf, 10-mco-default-madv.conf, 20-logging.conf, 20-nodenet.conf

Active: active (running) since Thu 2023-03-23 14:39:11 UTC; 1h 26min ago

Main PID: 3215 (kubelet)

Tasks: 28 (limit: 127707)

Memory: 391.7M

CPU: 14min 34.568s

*...output omitted...*

Check Pod Status

With RHOCP, you can view logs in running containers and pods to ease troubleshooting. When a container starts, RHOCP redirects the container's standard output and standard error to a disk in the container's ephemeral storage. With this redirect, you can view the container logs by using logs commands, even after the container stops. However, the pod hosting the container must still exist.

In RHOCP, the following command returns the output for a container within a pod:

[user@host~]$ **oc logs *pod-name* -c *container-name***

Replace *pod-name* with the name of the target pod, and replace *container-name* with the name of the target container. The -c *container-name* argument is optional, if the pod has only one container. You must use the -c *container-name* argument to connect to a specific container in a multicontainer pod. Otherwise, the command defaults to the only running container and returns the output.

When debugging images and setup problems, it is useful to get an exact copy of a running pod configuration, and then troubleshoot it with a shell. If a pod is failing or does not include a shell, then the rsh and exec commands might not work. To resolve this issue, the debug command creates a copy of the specified pod and starts a shell in that pod.

By default, the debug command starts a shell inside the first container of the referenced pod. The debug pod is a copy of your source pod, with some additional modifications. For example, the pod labels are removed. The executed command is also changed to the '/bin/sh' command for Linux containers, or the 'cmd.exe' executable for Windows containers. Additionally, readiness and liveness probes are disabled.

A common problem for containers in pods is security policies that prohibit a container from running as a root user. You can use the debug command to test running a pod as a non-root user by using the --as-user option. You can also run a non-root pod as the root user with the --as-root option.

With the debug command, you can invoke other types of objects besides pods. For example, you can use any controller resource that creates a pod, such as a deployment, a build, or a job. The debug command also works with nodes, and with resources that can create pods, such as image stream tags. You can also use the --image=IMAGE option of the debug command to start a shell session by using a specified image.

If you do not include a resource type and name, then the debug command starts a shell session into a pod by using the OpenShift tools image.

[user@host~]$ **oc debug**

The next example tests running a job pod as a non-root user.

[user@host~]$ **oc debug job/test --as-user=1000000**

The following example creates a debug session for a node.

[user@host~]$ **oc debug node/master01**

Starting pod/master01-debug-wtn9r ...

To use host binaries, run **chroot /host**

Pod IP: 192.168.50.10

If you don't see a command prompt, try pressing enter.

sh-4.4# chroot /host

sh-5.1#

The debug pod is deleted when the remote command completes, or when the user interrupts the shell.

Collect Information for Support Requests

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support. It is recommended that you provide the following information:

* Data gathered by using the oc adm must-gather command as a cluster administrator
* The unique cluster ID

The oc adm must-gather command collects resource definitions and service logs from your cluster that are most likely needed for debugging issues. This command creates a pod in a temporary namespace on your cluster, and the pod then gathers and downloads debugging information. By default, the oc adm must-gather command uses the default plug-in image, and writes into the ./must-gather.local. directory on your local system. To write to a specific local directory, you can also use the --dest-dir option, such as in the following example:

[user@host~]$ **oc adm must-gather --dest-dir /home/student/must-gather**

Then, create a compressed archive file from the must-gather directory. For example, on a Linux-based system, you can run the following command:

[user@host~]$ **tar cvaf mustgather.tar must-gather/**

Replace must-gather/ with the actual directory path.

Then, attach the compressed archive file to your support case in the Red Hat Customer Portal.

Similar to the oc adm must-gather command, the oc adm inspect command gathers information on a specified resource. For example, the following command collects debugging data for the openshift-apiserver and kube-apiserver cluster operators.

[user@host~]$ **oc adm inspect clusteroperator/openshift-apiserver \**

**clusteroperator/kube-apiserver**

The oc adm inspect command can also use the --dest-dir option to specify a local directory to write the gathered information. The command shows all logs by default. Use the --since option to filter the results to logs that are later than a relative duration, such as 5s, 2m, or 3h.

[user@host~]$ **oc adm inspect clusteroperator/openshift-apiserver --since 10m**

Guided Exercise: Assess the Health of an OpenShift Cluster

Verify the health of an OpenShift cluster by querying the status of its cluster operators, nodes, pods, and systemd services. Also verify cluster events and alerts.

**Outcomes**

* View the status and get information about cluster operators.
* Retrieve information about cluster pods and nodes.
* Retrieve the status of a node's systemd services.
* View cluster events and alerts.
* Retrieve debugging information for the cluster.

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 cli-health**

**Instructions**

1. Retrieve the status and view information about cluster operators.
   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. List the operators that users installed in the OpenShift cluster.
  2. [student@workstation ~]$ **oc get operators**
  3. NAME AGE
  4. lvms-operator.openshift-storage 27d

metallb-operator.metallb-system 27d

* 1. List the cluster operators that are installed by default in the OpenShift cluster.
  2. [student@workstation ~]$ **oc get clusteroperators**
  3. NAME VERSION AVAILABLE PROGRESSING DEGRADED ...
  4. authentication 4.14.0 True False False ...
  5. baremetal 4.14.0 True False False ...
  6. cloud-controller-manager 4.14.0 True False False ...
  7. cloud-credential 4.14.0 True False False ...
  8. cluster-autoscaler 4.14.0 True False False ...
  9. config-operator 4.14.0 True False False ...
  10. console 4.14.0 True False False ...
  11. control-plane-machine-set 4.14.0 True False False ...
  12. csi-snapshot-controller 4.14.0 True False False ...
  13. dns 4.14.0 True False False ...
  14. etcd 4.14.0 True False False ...

*...output omitted...*

* 1. Use the describe command to view detailed information about the openshift-apiserver cluster operator, such as related objects, events, and version.
  2. [student@workstation ~]$ **oc describe clusteroperators openshift-apiserver**
  3. Name: openshift-apiserver
  4. Namespace:
  5. Labels: <none>
  6. Annotations: exclude.release.openshift.io/internal-openshift-hosted: true
  7. include.release.openshift.io/self-managed-high-availability: true
  8. include.release.openshift.io/single-node-developer: true
  9. API Version: config.openshift.io/v1
  10. Kind: ClusterOperator
  11. Metadata:
  12. *...output omitted...*
  13. Spec:
  14. Status:
  15. Conditions:
  16. Last Transition Time: 2023-02-09T22:41:08Z
  17. Message: All is well
  18. Reason: AsExpected
  19. Status: False
  20. Type: Degraded
  21. *...output omitted...*
  22. Extension: <nil>
  23. Related Objects:
  24. Group: operator.openshift.io
  25. Name: cluster
  26. Resource: openshiftapiservers
  27. Group:
  28. Name: openshift-config
  29. Resource: namespaces
  30. Group:
  31. Name: openshift-config-managed
  32. Resource: namespaces
  33. Group:
  34. Name: openshift-apiserver-operator
  35. Resource: namespaces
  36. Group:
  37. Name: openshift-apiserver
  38. Resource: namespaces
  39. *...output omitted...*
  40. Versions:
  41. Name: operator
  42. Version: 4.14.0
  43. Name: openshift-apiserver
  44. Version: 4.14.0

Events: <none>

The Related Objects attribute includes information about the name, resource type, and groups for objects that are related to the operator.

* 1. List the pods in the openshift-apiserver-operator namespace. Then, view the detailed status of an openshift-apiserver-operator pod by using the JSON format and the jq command. Your pod names might differ.
  2. [student@workstation ~]$ **oc get pods -n openshift-apiserver-operator**
  3. NAME READY STATUS RESTARTS AGE

openshift-apiserver-operator-7ddc8958fb-7m2kr 1/1 Running 11 27d

[student@workstation ~]$ **oc get pod -n openshift-apiserver-operator \**

**openshift-apiserver-operator-*7ddc8958fb-7m2kr* \**

**-o json | jq .status**

{

"conditions": [

*...output omitted...*

{

"lastProbeTime": null,

"lastTransitionTime": "2023-03-08T15:41:34Z",

"status": "True",

"type": "Ready"

},

*...output omitted...*

],

"containerStatuses": [

{

*...output omitted...*

"name": "openshift-apiserver-operator",

"ready": true,

"restartCount": 11,

"started": true,

"state": {

"running": {

"startedAt": "2023-03-08T15:41:34Z"

}

}

}

],

"hostIP": "192.168.50.10",

"phase": "Running",

"podIP": "10.8.0.5",

*...output omitted...*

}

1. Retrieve the status, resource consumption, and events of cluster pods.
   1. List the memory and CPU usage of all pods in the cluster. Use the --sum option to print the sum of the resource usage. The resource usage on your system probably differs.
   2. [student@workstation ~]$ **oc adm top pods -A --sum**
   3. NAMESPACE NAME CPU(cores) MEMORY(bytes)metallb-system controller-5f6dfd8c4f-ddr8v 0m 39Mi
   4. metallb-system metallb-operator-controller-manager-... 1m 38Mi
   5. metallb-system metallb-operator-webhook-server-... 1m 18Mi
   6. metallb-system speaker-2dds4 10m 94Mi
   7. *...output omitted...*

505m 8982Mi

* 1. List the pods and their labels in the openshift-etcd namespace.
  2. [student@workstation ~]$ **oc get pods -n openshift-etcd --show-labels**
  3. NAME READY STATUS RESTARTS AGE LABELS
  4. etcd-master01 4/4 Running 40 27d app=etcd,etcd=true,k8s-app=etcd,revision=3

installer-3-master01 0/1 Completed 0 27d app=installer

* 1. List the resource usage of the containers in the etcd-master01 pod in the openshift-etcd namespace. The resource usage on your system probably differs.
  2. [student@workstation ~]$ **oc adm top pods etcd-master01 \**
  3. **-n openshift-etcd --containers**
  4. POD NAME CPU(cores) MEMORY(bytes)
  5. etcd-master01 POD 0m 0Mi
  6. etcd-master01 etcd 57m 1096Mi
  7. etcd-master01 etcd-metrics 7m 20Mi
  8. etcd-master01 etcd-readyz 4m 40Mi

etcd-master01 etcdctl 0m 0Mi

* 1. Display a list of all resources, their status, and their types in the openshift-monitoring namespace.
  2. [student@workstation ~]$ **oc get all -n openshift-monitoring --show-kind**
  3. NAME READY STATUS ...
  4. pod/alertmanager-main-0 6/6 Running ...
  5. pod/cluster-monitoring-operator-56b769b58f-dtmqj 2/2 Running ...
  6. pod/kube-state-metrics-75455b796c-8q28d 3/3 Running ...
  7. *...output omitted...*
  8. NAME TYPE CLUSTER-IP ...
  9. service/alertmanager-main ClusterIP 172.30.85.183 ...
  10. service/alertmanager-operated ClusterIP None ...
  11. service/cluster-monitoring-operator ClusterIP None ...
  12. service/kube-state-metrics ClusterIP None ...

*...output omitted...*

* 1. View the logs of the alertmanager-main-0 pod in the openshift-monitoring namespace. The logs might differ on your system.
  2. [student@workstation ~]$ **oc logs alertmanager-main-0 -n openshift-monitoring**
  3. *...output omitted...*
  4. ts=2023-03-09T14:57:11.850Z caller=coordinator.go:113 level=info component=configuration msg="Loading configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml

ts=2023-03-09T14:57:11.850Z caller=coordinator.go:126 level=info component=configuration msg="Completed loading of configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml

* 1. Retrieve the events for the openshift-kube-controller-manager namespace.
  2. [student@workstation ~]$ **oc get events -n openshift-kube-controller-manager**
  3. LAST SEEN TYPE REASON OBJECT ...
  4. 175m Normal CreatedSCCRanges pod/kube-controller-manager-master01...

11m Normal CreatedSCCRanges pod/kube-controller-manager-master01...

1. Retrieve information about cluster nodes.
   1. View the status of the nodes in the cluster.
   2. [student@workstation ~]$ **oc get nodes**
   3. NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master,worker 27d v1.27.6+f67aeb3

* 1. Retrieve the resource consumption of the master01 node. The resource usage on your system probably differs.
  2. [student@workstation ~]$ **oc adm top node**
  3. NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

master01 781m 10% 11455Mi 60%

* 1. Use a JSONPath filter to determine the capacity and allocatable CPU for the master01 node. The values might differ on your system.
  2. [student@workstation ~]$ **oc get node master01 -o jsonpath=\**
  3. **'Allocatable: {.status.allocatable.cpu}{"\n"}'\**
  4. **'Capacity: {.status.capacity.cpu}{"\n"}'**
  5. Allocatable: 7500m

Capacity: 8

* 1. Determine the number of allocatable pods for the node.
  2. [student@workstation ~]$ **oc get node master01 -o jsonpath=\**
  3. **'{.status.allocatable.pods}{"\n"}'**

250

* 1. Use the describe command to view the events, resource requests, and resource limits for the node. The output might differ on your system.
  2. [student@workstation ~]$ **oc describe node master01**
  3. *...output omitted...*
  4. Allocated resources:
  5. (Total limits may be over 100 percent, i.e., overcommitted.)
  6. Resource Requests Limits
  7. -------- -------- ------
  8. cpu 3158m (42%) 980m (13%)
  9. memory 12667Mi (66%) 1250Mi (6%)
  10. ephemeral-storage 0 (0%) 0 (0%)
  11. hugepages-1Gi 0 (0%) 0 (0%)
  12. hugepages-2Mi 0 (0%) 0 (0%)
  13. Events:
  14. Type Reason Age From Message
  15. ---- ------ ---- ---- -------
  16. Normal Starting 106m kubelet Starting kubelet.
  17. Normal NodeHasSufficientMemory 106m (x9 over 106m) kubelet Node master01 status is now: NodeHasSufficientMemory
  18. Normal NodeHasNoDiskPressure 106m (x7 over 106m) kubelet Node master01 status is now: NodeHasNoDiskPressure
  19. Normal NodeHasSufficientPID 106m (x7 over 106m) kubelet Node master01 status is now: NodeHasSufficientPID

*...output omitted...*

1. Retrieve the logs and status of the systemd services on the master01 node.
   1. Display the logs of the node. Filter the logs to show the most recent log for the crio service. The logs might differ on your system.
   2. [student@workstation ~]$ **oc adm node-logs master01 -u crio --tail 1**
   3. -- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:57:00 UTC. --
   4. Mar 09 02:39:29.158989 master01 crio[3201]: time="2023-03-09 02:39:29.158737393Z" level=info msg="Image status: &ImageStatusResponse

*...output omitted...*

* 1. Display the two most recent logs of the kubelet service on the node. The logs might differ on your system.
  2. [student@workstation ~]$ **oc adm node-logs master01 -u kubelet --tail 2**
  3. -- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:59:16 UTC. --
  4. Mar 09 02:40:57.466711 master01 systemd[1]: Stopped Kubernetes Kubelet.
  5. Mar 09 02:40:57.466835 master01 systemd[1]: kubelet.service: Consumed 1h 27min 8.069s CPU time
  6. -- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:59:16 UTC. --
  7. Mar 09 16:58:52.133046 master01 kubenswrapper[3195]: I0309 16:58:52.132866 3195 kubelet\_getters.go:182] "Pod status updated" pod="openshift-etcd/etcd-master01" status=Running

Mar 09 16:58:52.133046 master01 kubenswrapper[3195]: I0309 16:58:52.132882 3195 kubelet\_getters.go:182] "Pod status updated" pod="openshift-kube-apiserver/kube-apiserver-master01" status=Running

* 1. Create a debug session for the node. The, use the chroot /host command to access the host binaries.
  2. [student@workstation ~]$ **oc debug node/master01**
  3. Starting pod/master01-debug-khltm ...
  4. To use host binaries, run `chroot /host`
  5. Pod IP: 192.168.50.10
  6. If you don't see a command prompt, try pressing enter.
  7. sh-4.4# **chroot /host**

sh-5.1#

* 1. Verify the status of the kubelet service.
  2. sh-5.1# **systemctl status kubelet**
  3. ● kubelet.service - Kubernetes Kubelet
  4. Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; preset: disabled)
  5. Drop-In: /etc/systemd/system/kubelet.service.d
  6. └─01-kubens.conf, 10-mco-default-madv.conf, 20-logging.conf, 20-nodenet.conf
  7. Active: active (running) since Thu 2023-03-09 14:54:51 UTC; 2h 8min ago
  8. Main PID: 3195 (kubelet)
  9. Tasks: 28 (limit: 127707)
  10. Memory: 540.7M
  11. CPU: 18min 32.117s

*...output omitted...*

Press **Ctrl**+**C** to quit the command.

* 1. Confirm that the crio service is active.
  2. sh-5.1# **systemctl is-active crio**

active

* 1. Exit the debug pod.
  2. sh-5.1# **exit**
  3. exit
  4. sh-4.4# **exit**
  5. exit

Removing debug pod ...

1. Retrieve debugging information for the cluster.
   1. Retrieve debugging information of the cluster by using the oc adm must-gather command. Specify the /home/student/must-gather directory as the destination directory. This command might take several minutes to complete.
   2. [student@workstation ~]$ **oc adm must-gather --dest-dir /home/student/must-gather**
   3. [must-gather ] OUT Using must-gather plug-in image: quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:07d3...e94c
   4. *...output omitted...*
   5. Reprinting Cluster State:
   6. When opening a support case, bugzilla, or issue please include the following summary data along with any other requested information:
   7. ClusterID: 94ff22c1-88a0-44cf-90f6-0b7b8b545434
   8. ClusterVersion: Stable at "4.14.0"
   9. ClusterOperators:

All healthy and stable

* 1. Verify that the debugging information exists in the destination directory. List the last five kubelet service logs, and confirm that an error occurred with proxying the data from the 192.168.50.10 IP address. Replace quay-io…​ with the generated directory name.
  2. [student@workstation ~]$ **ls ~/must-gather**
  3. event-filter.html
  4. quay-io-openshift-release-dev-ocp-v4-0-art-dev-sha256-07d3...e94c
  5. timestamp
  6. [student@workstation ~]$ **tail -5 \**
  7. **~/must-gather/quay-io.../host\_service\_logs/masters/kubelet\_service.log**
  8. *...output omitted...*
  9. Mar 09 01:12:09.680445 master01 kubenswrapper[3275]: I1206 01:12:09.680399 3275 logs.go:323] "Finished parsing log file" path="/var/log/pods/openshift-service-ca\_service-ca-5d96446959-69jq8\_c9800778-c955-4b89-9bce-9f043237c986/service-ca-controller/9.log"

Mar 09 01:12:12.771111 master01 kubenswrapper[3275]: E1206 01:12:12.770971 3275 upgradeaware.go:426] Error proxying data from client to backend: readfrom tcp 192.168.50.10:44410->192.168.50.10:10010: write tcp 192.168.50.10:44410->192.168.50.10:10010: write: broken pipe

* 1. Generate debugging information for the openshift-apiserver cluster operator. Specify the /home/student/inspect directory as the destination directory. Limit the debugging information to the last five minutes.
  2. [student@workstation ~]$ **oc adm inspect clusteroperator/openshift-apiserver \**
  3. **--dest-dir /home/student/inspect --since 5m**
  4. Gathering data for ns/openshift-config...
  5. Gathering data for ns/openshift-config-managed...
  6. Gathering data for ns/openshift-kube-apiserver-operator...
  7. Gathering data for ns/openshift-kube-apiserver...
  8. Gathering data for ns/openshift-etcd-operator...
  9. Wrote inspect data to /home/student/inspect.

*...output omitted...*

* 1. Verify that the debugging information exists in the destination directory, and review the cluster.yaml file from the ~/inspect/cluster-scoped-resources/operator.openshift.io/openshiftapiservers directory.
  2. [student@workstation ~]$ **ls inspect/**
  3. cluster-scoped-resources
  4. event-filter.html
  5. namespaces
  6. timestamp
  7. [student@workstation ~]$ **cat \**
  8. **~/inspect/cluster-scoped-resources/operator.openshift.io/\**
  9. **openshiftapiservers/cluster.yaml**
  10. apiVersion: operator.openshift.io/v1
  11. kind: OpenShiftAPIServer
  12. metadata:
  13. annotations:
  14. include.release.openshift.io/ibm-cloud-managed: "true"
  15. include.release.openshift.io/self-managed-high-availability: "true"
  16. include.release.openshift.io/single-node-developer: "true"
  17. release.openshift.io/create-only: "true"
  18. creationTimestamp: "2023-12-12T16:03:42Z"
  19. generation: 3
  20. managedFields:
  21. - apiVersion: operator.openshift.io/v1
  22. fieldsType: FieldsV1
  23. fieldsV1:
  24. f:metadata:
  25. f:annotations:
  26. .: {}

*...output omitted...*

* 1. Delete the debugging information from your system.

[student@workstation ~]$ **rm -rf must-gather inspect**

**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 cli-health**

Lab: Kubernetes and OpenShift Command-line Interfaces and APIs

Find detailed information about your OpenShift cluster and assess its health by querying its Kubernetes resources.

**Outcomes**

* Use the command line to retrieve information about the cluster resources.
* Identify cluster operators and API resources.
* List the available namespaced resources.
* Identify the resources that belong to the core API group.
* List the resource types that the oauth.openshift.io API group provides.
* List the resource usage of containers in a pod.
* Use the JSONPath filter to get the number of allocatable pods and compute resources for a node.
* List the memory and CPU usage of all pods in the cluster.
* Use jq filters to retrieve the conditions status of a pod.
* View cluster events and alerts.

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

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

**Instructions**

The API URL of your OpenShift cluster is https://api.ocp4.example.com:6443, and the oc command is already installed on your workstation machine.

Log in to the OpenShift cluster as the developer user with the developer password. Use the cli-review project for your work.

1. Log in to the OpenShift cluster and create the cli-review project.
   1. Log in to the OpenShift cluster.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**

*...output omitted...*

* 1. Create the cli-review project.
  2. [student@workstation ~]$ **oc new-project cli-review**
  3. Now using project "cli-review" on server "https://api.ocp4.example.com:6443".

*...output omitted...*

1. Hide Solution
2. Use the oc command to list the following information for the cluster:
   1. Retrieve the cluster version.
   2. Identify the supported API versions.
   3. Identify the fields for the pod.spec.securityContext object.

Show Solution

1. From the terminal, log in to the OpenShift cluster as the admin user with the redhatocp password. Then, use the command line to identify the following cluster resources:
   1. List the cluster operators.
   2. Identify the available namespaced resources.
   3. Identify the resources that belong to the core API group.
   4. List the resource types that the oauth.openshift.io API group provides.
   5. List the events in the openshift-kube-controller-manager namespace.
   6. Log in to the OpenShift cluster.
   7. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   8. **https://api.ocp4.example.com:6443**

*...output omitted...*

* 1. List the cluster operators.
  2. [student@workstation ~]$ **oc get clusteroperators**
  3. NAME VERSION AVAILABLE PROGRESSING DEGRADED SINCE
  4. authentication 4.14.0 True False False 12h
  5. baremetal 4.14.0 True False False 31d
  6. cloud-controller-manager 4.14.0 True False False 31d
  7. cloud-credential 4.14.0 True False False 31d
  8. cluster-autoscaler 4.14.0 True False False 31d
  9. config-operator 4.14.0 True False False 31d
  10. console 4.14.0 True False False 31d

*...output omitted...*

* 1. List the available namespaced resources.
  2. [student@workstation ~]$ **oc api-resources --namespaced**
  3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
  4. bindings v1 true Binding
  5. configmaps cm v1 true ConfigMap
  6. endpoints ep v1 true Endpoints
  7. events ev v1 true Event
  8. limitranges limits v1 true LimitRange
  9. persistentvolumeclaims pvc v1 true PersistentVolumeClaim
  10. pods po v1 true Pod

*...output omitted...*

* 1. Identify the resources that belong to the core API group.
  2. [student@workstation ~]$ **oc api-resources --api-group ''**
  3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
  4. bindings v1 true Binding
  5. componentstatuses cs v1 false ComponentStatus
  6. configmaps cm v1 true ConfigMap
  7. endpoints ep v1 true Endpoints
  8. events ev v1 true Event
  9. limitranges limits v1 true LimitRange
  10. namespaces ns v1 false Namespace
  11. nodes no v1 false Node

*...output omitted...*

* 1. List the resource types that the oauth.openshift.io API group provides.
  2. [student@workstation ~]$ **oc api-resources --api-group oauth.openshift.io**
  3. NAME SHORTNAMES APIVERSION NAMESPACED KIND
  4. oauthaccesstokens oauth.openshift.io/v1 false OAuthAccessToken
  5. oauthauthorizationtokens oauth.openshift.io/v1 false OAutheAuthorizationToken

*...output omitted...*

* 1. Retrieve the events for the openshift-kube-controller-manager namespace.
  2. [student@workstation ~]$ **oc get events -n openshift-kube-controller-manager**
  3. LAST SEEN TYPE REASON OBJECT ...
  4. 48m Normal CreatedSCCRanges pod/kube-controller-manager-master ...
  5. 21m Normal CreatedSCCRanges pod/kube-controller-manager-master ...

14m Normal CreatedSCCRanges pod/kube-controller-manager-master ...

1. Hide Solution
2. Identify the following information about the cluster services and its nodes:
   1. Retrieve the conditions status of the etcd-master01 pod in the openshift-etcd namespace by using jq filters to limit the output.
   2. List the compute resource usage of the containers in the etcd-master01 pod in the openshift-etcd namespace.
   3. Get the number of allocatable pods for the master01 node by using a JSONPath filter.
   4. List the memory and CPU usage of all pods in the cluster.
   5. Retrieve the compute resource consumption of the master01 node.
   6. Retrieve the capacity and allocatable CPU for the master01 node by using a JSONPath filter.
   7. Retrieve the conditions status of the etcd-master01 pod in the openshift-etcd namespace. Use jq filters to limit the output to the .status.conditions attribute of the pod.
   8. [student@workstation ~]$ **oc get pods etcd-master01 -n openshift-etcd \**
   9. **-o json | jq .status.conditions**
   10. [
   11. {
   12. "lastProbeTime": null,
   13. "lastTransitionTime": "2023-03-12T16:40:35Z",
   14. "status": "True",
   15. "type": "Initialized"
   16. },
   17. {
   18. "lastProbeTime": null,
   19. "lastTransitionTime": "2023-03-12T16:40:47Z",
   20. "status": "True",
   21. "type": "Ready"
   22. },
   23. {
   24. "lastProbeTime": null,
   25. "lastTransitionTime": "2023-03-12T16:40:47Z",
   26. "status": "True",
   27. "type": "ContainersReady"
   28. },
   29. {
   30. "lastProbeTime": null,
   31. "lastTransitionTime": "2023-03-12T16:40:23Z",
   32. "status": "True",
   33. "type": "PodScheduled"
   34. }

]

* 1. List the resource usage of the containers in the etcd-master01 pod in the openshift-etcd namespace.
  2. [student@workstation ~]$ **oc adm top pods etcd-master01 \**
  3. **-n openshift-etcd --containers**
  4. POD NAME CPU(cores) MEMORY(bytes)
  5. etcd-master01 POD 0m 0Mi
  6. etcd-master01 etcd 54m 1513Mi
  7. etcd-master01 etcd-metrics 5m 24Mi
  8. etcd-master01 etcd-readyz 4m 39Mi

etcd-master01 etcdctl 0m 0Mi

* 1. Use a JSONPath filter to determine the number of allocatable pods for the master01 node.
  2. [student@workstation ~]$ **oc get node master01 \**
  3. **-o jsonpath='{.status.allocatable.pods}{"\n"}'**

250

* 1. List the memory and CPU usage of all pods in the cluster. Use the --sum option to print the sum of the resource usage. The resource usage on your system probably differs.
  2. [student@workstation ~]$ **oc adm top pods -A --sum**
  3. NAMESPACE NAME CPU(cores) MEMORY(bytes)
  4. metallb-system controller-5f6dfd8c4f-ddr8v 0m 56Mi
  5. metallb-system metallb-operator-controller-manager-... 0m 50Mi
  6. metallb-system metallb-operator-webhook-server-... 0m 26Mi
  7. metallb-system speaker-2dds4 9m 210Mi
  8. *...output omitted...*
  9. -------- --------

505m 8982Mi

* 1. Retrieve the resource consumption of the master01 node.
  2. [student@workstation ~]$ **oc adm top node**
  3. NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

master01 1199m 15% 12555Mi 66%

* 1. Use a JSONPath filter to determine the capacity and allocatable CPU for the master01 node.
  2. [student@workstation ~]$ **oc get node master01 -o jsonpath=\**
  3. **'Allocatable: {.status.allocatable.cpu}{"\n"}'\**
  4. **'Capacity: {.status.capacity.cpu}{"\n"}'**
  5. Allocatable: 7500m

Capacity: 8

1. Hide Solution
2. Retrieve debugging information for the cluster. Specify the /home/student/DO180/labs/cli-review/debugging directory as the destination directory.

Then, generate debugging information for the kube-apiserver cluster operator. Specify the /home/student/DO180/labs/cli-review/inspect directory as the destination directory. Limit the debugging information to the last five minutes.

* 1. Retrieve debugging information for the cluster. Save the output to the /home/student/DO180/labs/cli-review/debugging directory.
  2. [student@workstation ~]$ **oc adm must-gather \**
  3. **--dest-dir /home/student/DO180/labs/cli-review/debugging**
  4. [must-gather ] OUT Using must-gather plug-in image: quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:07d3...e94c
  5. *...output omitted...*
  6. Reprinting Cluster State:
  7. When opening a support case, bugzilla, or issue please include the following summary data along with any other requested information:
  8. ClusterID: 94ff22c1-88a0-44cf-90f6-0b7b8b545434
  9. ClusterVersion: Stable at "4.14.0"
  10. ClusterOperators:

All healthy and stable

* 1. Generate debugging information for the kube-apiserver cluster operator. Save the output to the /home/student/DO180/labs/cli-review/inspect directory, and limit the debugging information to the last five minutes.
  2. [student@workstation ~]$ **oc adm inspect clusteroperator kube-apiserver \**
  3. **--dest-dir /home/student/DO180/labs/cli-review/inspect --since 5m**
  4. Gathering data for ns/metallb-system...
  5. *...output omitted...*

Wrote inspect data to /home/student/DO180/labs/cli-review/inspect.

Hide Solution

**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 cli-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 cli-review**

[Previous](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s06/c2358540-87d5-48de-b49e-6f23bdcd629c)[Next](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s08/c2358540-87d5-48de-b49e-6f23bdcd629c)

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **1.** | Which supported API resource is a member of the oauth.openshift.io api-group? |  |
| A |  | groupoauthaccesstokens |
| B |  | tokenreviews |
| C |  | cloudcredentials |
| D |  | networkpolicies |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **2.** | Which two fields are members of the pod.spec.securityContext object? (Choose two.) |  |
| A |  | readinessGates |
| B |  | runAsUser |
| C |  | sysctls |
| D |  | priorityClassName |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **3.** | Which three commands display the conditions of the master01 node? (Choose three.) |  |
| A |  | oc get node/master01 -o json | jq '.status.conditions' |
| B |  | oc get node/master01 -o wide |
| C |  | oc get node/master01 -o yaml |
| D |  | oc get node/master01 -o json |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **4.** | Select the two valid condition types for a control plane node. (Choose two) |  |
| A |  | PIDPressure |
| B |  | DiskIOPressure |
| C |  | OutOfMemory |
| D |  | Ready |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **5.** | Select three valid options for the oc adm top pods command. (Choose three.) |  |
| A |  | -A |
| B |  | --sum |
| C |  | --pod-selector |
| D |  | --containers |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **6.** | Which command determines the assigned internal IP address on the master01 node? |  |
| A |  | oc get nodes master01 -o=jsonpath='{.status.addresses}{"\n"}' |
| B |  | oc get nodes master01 -o json | jq .status.nodeInfo |
| C |  | oc get nodes master01 -o=jsonpath='{.status.conditions}{"\n"}' |

1. CheckResetShow Solution

Bottom of Form

Top of Form

|  |  |  |
| --- | --- | --- |
|  | | |
| **7.** | Which command displays only the conditions for the authentication cluster operator? |  |
| A |  | oc get clusteroperators.config.openshift.io authentication -o json | jq .status.versions |
| B |  | oc get clusteroperators.config.openshift.io authentication -o json | jq .status |
| C |  | oc get clusteroperators.config.openshift.io authentication -o json | jq .status.conditions |

1. CheckResetShow Solution

Bottom of Form

[Previous](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s07/c2358540-87d5-48de-b49e-6f23bdcd629c)[Next](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s09/c2358540-87d5-48de-b49e-6f23bdcd629c)

Summary

* An RHOCP cluster can be managed from the web console or by using the kubectl or oc command-line interfaces (CLI).
* Use the --help option on any command to view detailed information about the command.
* Projects provide isolation between your application resources.
* Token authentication is the only guaranteed method to work with any RHOCP cluster, because enterprise SSO might replace the login form of the web console.
* All administrative tasks require creating, viewing, and changing the API resources.
* Kubernetes provides YAML- and JSON-formatted output options, which are ideal for parsing or scripting.
* Operators provide the means of monitoring applications, performing health checks, managing over-the-air (OTA) updates, and ensuring that applications remain in your specified state.
* The RHOCP web console incorporates useful graphs to visualize cluster and resource analytics.
* The RHOCP web console provides an interface for executing Prometheus queries, visualizing metrics, and configuring alerts.
* The monitoring stack is based on the Prometheus project, and it is configured to monitor the core RHOCP cluster components, by default.
* RHOCP provides the ability to view logs in running containers and pods to ease troubleshooting.
* You can collect resource definitions and service logs from your cluster by using the oc adm must-gather command.

[Previous](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch02s08/c2358540-87d5-48de-b49e-6f23bdcd629c)[Next](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch03/c2358540-87d5-48de-b49e-6f23bdcd629c)

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

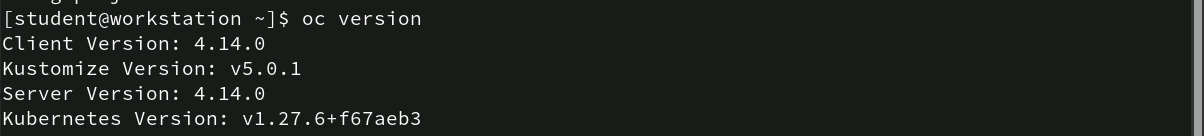
Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated



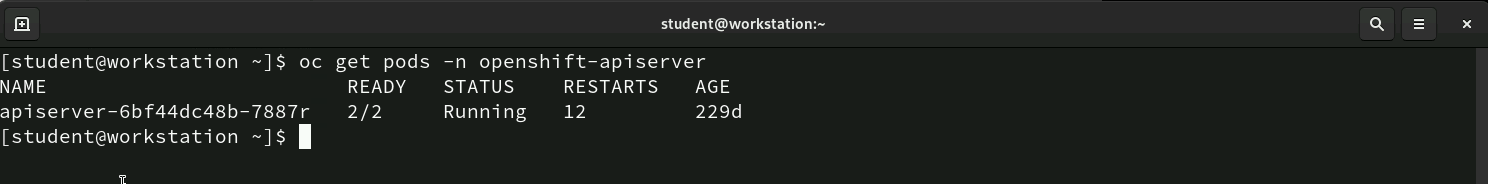


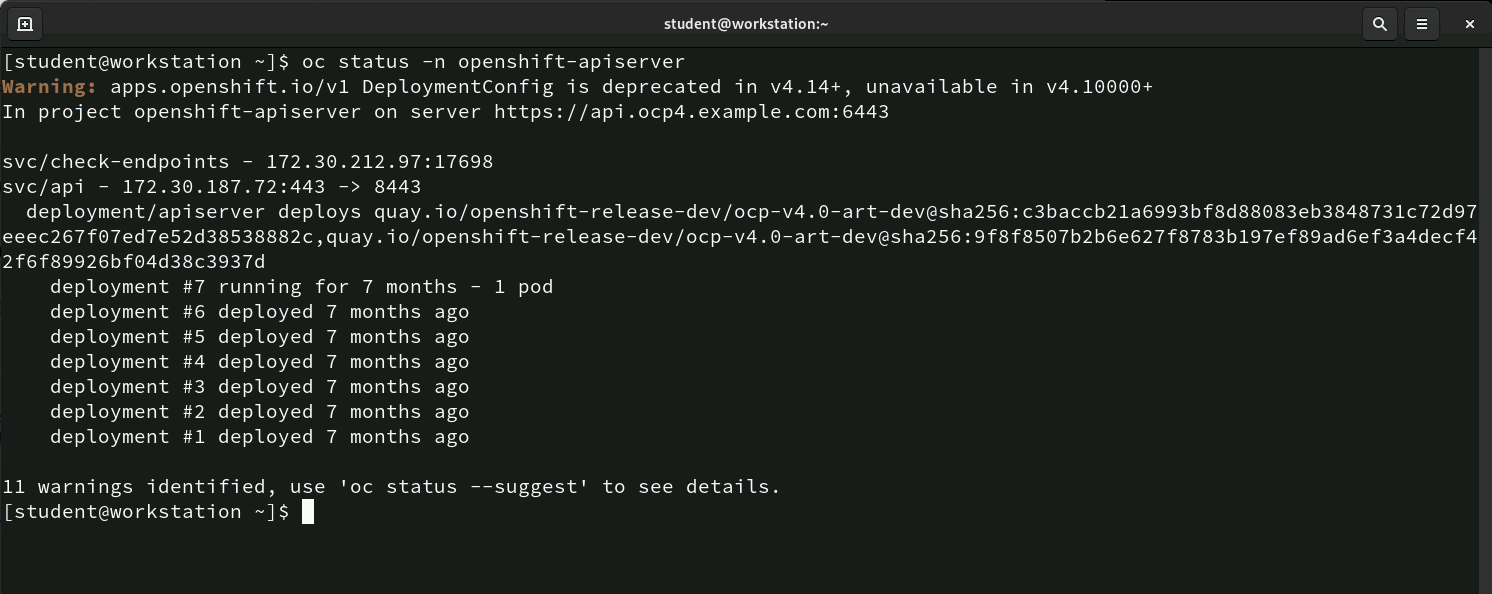
A screenshot of a computer

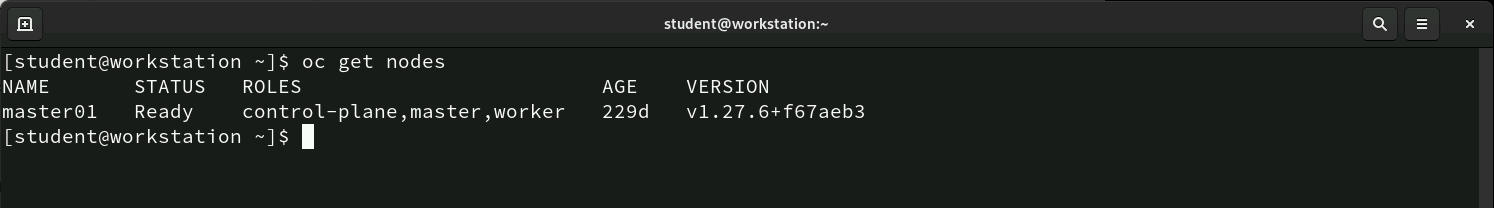
Description automatically generated

A screenshot of a computer

Description automatically generated

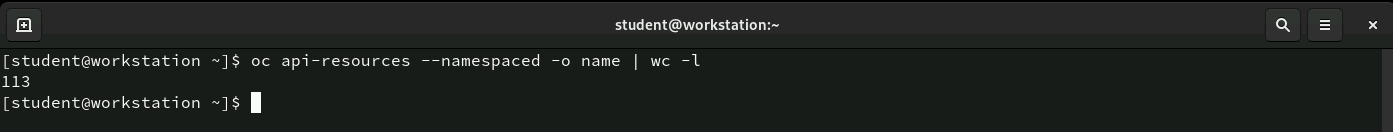


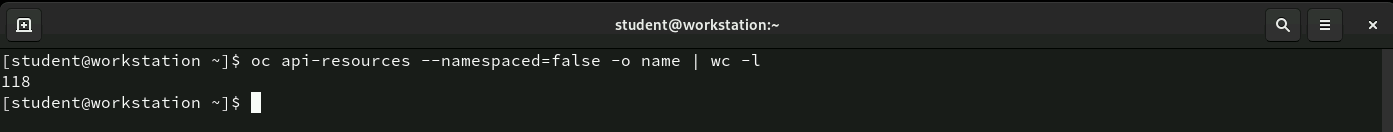




A screenshot of a computer

Description automatically generated



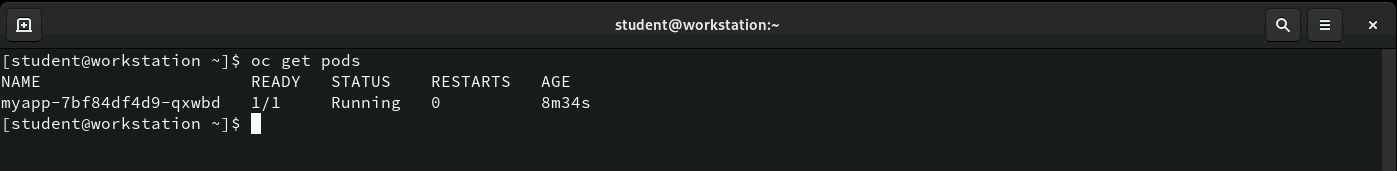


A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

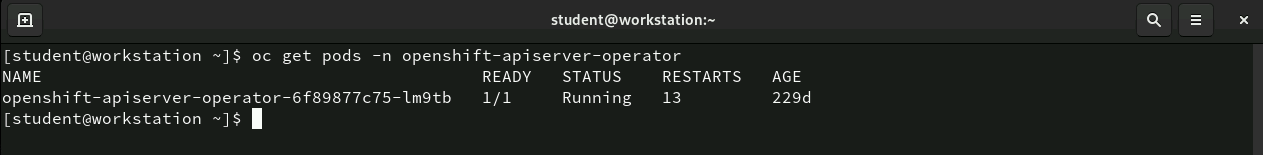
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A black screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

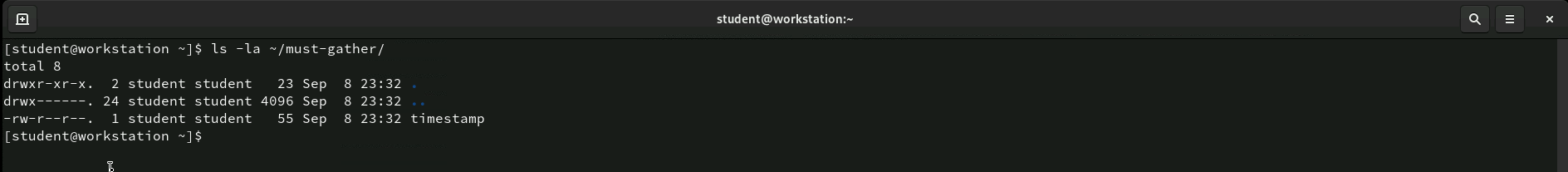
Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated