Red Hat OpenShift Administration III: Scaling Deployments in the Enterprise

Introduction

Red Hat OpenShift Administration III: Scaling Deployments in the Enterprise

This course teaches the required skills to manage OpenShift clusters at scale to productively support a growing number of stakeholders, applications, and users. You will learn how to configure and manage OpenShift clusters at scale to address increasing and special demands from applications and to ensure reliability, performance, and availability.

**Course Objectives**

* Configure pools of cluster nodes with special configurations, and ensure that only the intended workloads for those pools are scheduled on those nodes.
* Configure enterprise authentication and group management with legacy LDAP and cloud-native OIDC identity management systems.
* Deploy, manage, and query OpenShift Logging and configure log forwarding to external log aggregators and security information and event management (SIEM) systems.
* Automate cluster configuration and application deployment by using OpenShift GitOps.
* Troubleshoot application and cluster performance and availability issues by using OpenShift Monitoring.
* Configure and automate application-level backups by using OADP.

**Audience**

* Primary: Platform Engineers, System Administrators, Cloud Administrators, and other infrastructure-related IT roles who are responsible for implementing and managing infrastructure for applications.
* Secondary: Enterprise Architects, Site Reliability Engineers (SRE), DevOps Engineers, and other application-related IT roles who are responsible for designing infrastructure for applications.

**Prerequisites**

* Possess a current Red Hat Certified Specialist in OpenShift Administration certification, or equivalent experience and skills with managing OpenShift clusters and containerized applications, according to the topics of the DO180 and DO280 courses for version 4.12 and any later versions. Earlier 4.x releases of DO180 and DO280 are not sufficient.
* Experience with enterprise authentication, monitoring, and other IT infrastructure systems is helpful but not required.

Chapter 1.  Authentication and Identity Management

[The OpenShift OAuth Server and Identity Providers](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01)

[Quiz: The OpenShift OAuth Server and Identity Providers](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s02)

[LDAP Authentication and Group Synchronization](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s03)

[Guided Exercise: LDAP Authentication and Group Synchronization](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s04)

[Guided Exercise: Automate LDAP Group Synchronization](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s05)

[OIDC Authentication and Group Claims](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s06)

[Guided Exercise: OIDC Authentication and Group Claims](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s07)

[Guided Exercise: Solve User Sync Conflicts](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s08)

[Token and Client Certificate Authentication with kubeconfig Files](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s09)

[Guided Exercise: Token and Client Certificate Authentication with kubeconfig Files](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s10)

[Lab: Authentication and Identity Management](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s11)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch01s12)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Configure OpenShift clusters to authenticate by using LDAP and OIDC enterprise identity systems and to recognize groups that those systems define. |
| **Sections** | * The OpenShift OAuth Server and Identity Providers (and Quiz) * LDAP Authentication and Group Synchronization (and Guided Exercise) * Automate LDAP Group Synchronization (Guided Exercise) * OIDC Authentication and Group Claims (and Guided Exercise) * Solve User Sync Conflicts (Guided Exercise) * Token and Client Certificate Authentication with kubeconfig Files (and Guided Exercise) |
| **Lab** | * Authentication and Identity Management |

The OpenShift OAuth Server and Identity Providers

Objectives

* Define the concepts and custom resources of the OpenShift OAuth server, and explain how these resources augment Kubernetes authentication.

Authenticating API Requests

*Authentication* determines access to an OpenShift cluster, to ensure that only authenticated users can access the OpenShift cluster. To interact with an OpenShift cluster, users must authenticate to the Kubernetes API. After the user successfully authenticates to the cluster, the authorization layer either accepts or rejects the API request depending on the user's access permissions.

Authentication in Kubernetes

Authentication in Kubernetes verifies the signature validity for a token or a client certificate that the user provides. Kubernetes supports any authentication method, provided that the final result of the authentication method is a token or client certificate. Because Kubernetes cannot log in by any means by itself, you need extra tools to operationalize the external authentication methods. The core components for Kubernetes authentication are the API server and the authentication plug-ins. The API server is the central control plane component that is responsible for handling incoming requests. When a request reaches the API server, it forwards the request to the appropriate authentication plug-in. The authentication plug-in validates the provided credentials and determines whether the client is authenticated.

For example, Kubernetes considers to be authenticated any user that presents a valid certificate that is signed by a trusted certificate authority (CA). Kubernetes reads the subject field from the certificate, and assigns the username and the groups with the common name (CN) and the organization (O) fields, respectively. Kubernetes either accepts or rejects the API request for the user depending on the role-based access control (RBAC) rules that are configured for the username and the group.

Kubernetes rejects any request with an invalid access token or certificate by showing a 401 Unauthorized error.

If the request does not present an access token or certificate, then Kubernetes assigns to the request the system:anonymous system user and the system:unauthenticated system group. After assigning the anonymous user to the request, Kubernetes uses RBAC policies to determine whether the anonymous user has appropriate access to perform the requested action. System groups are explained later in this section.

Authentication in OpenShift

OpenShift expands the Kubernetes authentication mechanisms to provide additional features and integrations with external systems. OpenShift includes its own built-in OAuth server, which is not present in Kubernetes. The OpenShift OAuth server handles user authentication, and it issues OAuth tokens that grant access to specific resources.

You can authenticate to OpenShift by using the following methods:

**X.509 client certificates**

Use X.509 client certificates to authenticate to the API server. The API server validates the client certificate against a trusted certificate authority. This authentication method requires an HTTPS connection to the API server. OpenShift authentication through client certificates is the same as in Kubernetes. Client certificates are explained later in this course.

**OAuth access tokens**

In OpenShift, the OpenShift control plane includes a built-in OAuth server that generates OAuth access tokens. OpenShift preconfigures Kubernetes to trust tokens that its own internal OAuth server issues. Thus, after you log in to the OpenShift OAuth server, Kubernetes authenticates the requests by trusting the tokens that the internal OAuth server issued.

The OpenShift OAuth Server

OpenShift provides by default the authentication operator, which runs an internal OAuth server. The OAuth server issues OAuth access tokens to users for authenticating through the API. To configure authentication, the OpenShift OAuth server provides various identity providers (IdPs) that enable integration with identity management systems such as OpenID Connect and LDAP servers. Some provided IdPs support the custom OAuth workflows from internet services, such as GitHub. The OpenShift OAuth server works only with certain supported IdPs, and you cannot add other IdPs.

**NOTE**

A list of the supported IdPs is available at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#supported-identity-providers>

The most common way to authenticate to the OpenShift cluster is to use the oc login command in the command-line interface (CLI). The oc login command uses the OpenShift OAuth server to authenticate the user with an IdP. Then, the OpenShift OAuth server generates an OAuth access token for the user to authenticate their API requests.

The following diagram summarizes the authentication flow when you use the oc login command:

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | A user runs the oc login command, and provides the username and password. |
|  | OpenShift sends the credentials from the oc login command to the internal OAuth server. |
|  | The internal OAuth server validates the credentials with an IdP. |
|  | The internal OAuth server creates an OAuth access token for the user. |
|  | OpenShift stores the OAuth access token in the kubeconfig file, by default located at ~/.kube/config. |

You can verify the two endpoints that the oc login command uses, which are the API server and the OAuth server routes, by increasing the logging level:

[user@host ~]$ **oc login -u user -p password https://api.ocp4.example.com:6443 \**

**--loglevel 7**

I0929 ...] HEAD **https://api.ocp4.example.com:6443/**

I0929 ...] Request Headers:

I0929 ...] Response Status: 403 Forbidden in 20 milliseconds

I0929 ...] GSSAPI Enabled

I0929 ...] GET **https://api.ocp4.example.com:6443/.well-known/oauth-authorization-server**

I0929 ...] Request Headers:

I0929 ...] X-Csrf-Token: 1

I0929 ...] Response Status: 200 OK in 0 milliseconds

I0929 ...] using system roots as no error was encountered

I0929 ...] GET **https://oauth-openshift.apps.ocp4.example.com/oauth/authorize?client\_id=openshift-challenging-client&code\_challenge=B8RP...aCtk&code\_challenge\_method=S256&redirect\_uri=https%3A%2F%2Foauth-openshift.apps.ocp4.example.com%2Foauth%2Ftoken%2Fimplicit&response\_type=code**

*...output omitted...*

This authentication flow is similar when using the OpenShift web console instead of the CLI, except that the OpenShift web console stores the token in the browser memory instead of in a kubeconfig file. The two endpoints for login, which are the API server and the OAuth server, are the reason why you must accept two server certificates the first time that you log in to the OpenShift web console.

After getting the OAuth access token, every time that the user invokes the OpenShift CLI, OpenShift uses the stored credentials in the kubeconfig file to authenticate the user to the Kubernetes API server. The kubeconfig file details are explained later in this course.

The following diagram summarizes the flow when you use the OpenShift CLI after authenticating by using the oc login command:

A diagram of a user command

Description automatically generated

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | A user accesses the Kubernetes API by using the oc command. |
|  | OpenShift reads the credentials from the kubeconfig file. |
|  | OpenShift uses the credentials from the kubeconfig file to authenticate to the Kubernetes API server. |

You can also retrieve OAuth user access tokens from the OpenShift OAuth server by using the *namespace\_route*/oauth/authorize and *namespace\_route*/oauth/token endpoints.

**NOTE**

Requesting an OAuth access token through the OpenShift REST API is out of the scope of this course. For more information about this topic, refer to <https://access.redhat.com/solutions/6610781>

Configure the OpenShift OAuth Server

For an IdP to validate the identity of the requester, you must configure and enable at least one IdP in the OAuth server. If a user requests a new OAuth access token, then the OAuth server uses the configured IdPs to establish the requester's identity. After the user successfully logs in to the cluster, the OAuth server creates the OAuth access token for the user, and OpenShift creates the identity and user resources.

In production clusters, it is typical to configure multiple IdPs in the internal OAuth server. The following example shows the parameters for the custom resource (CR) to configure two IdPs in the OAuth server. The example uses htpasswd and LDAP as the IdPs.

**NOTE**

Configuring the htpasswd IdP for OpenShift authentication is explained in detail in the *DO280: Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster* course.

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- name: ***htpasswd-idp-name***

mappingMethod: ***claim***

type: HTPasswd

htpasswd:

fileData:

name: ***htpasswd-secret***

- name: ***ldap-idp-name***

mappingMethod: ***claim***

type: LDAP

ldap:

*...output omitted...*

|  |  |
| --- | --- |
|  | The identityProviders array contains the information for all the IdPs that are configured in your cluster. |
|  | OpenShift prefixes the value of the identity claim with the provider name to form an identity name, and also uses the identity claim to build the redirect URL. |
|  | The mappingMethod parameter controls how OpenShift establishes mappings between the IdP identities and the User resources. |
|  | The OpenShift secret resource contains the users and passwords. |

Users, Groups, and Identities

Users, groups, and identities play a crucial role in managing access and permissions within the cluster.

Users, Groups, and Identities in Kubernetes

Kubernetes does not have resources that represent user accounts, groups, or identities. Users and groups in Kubernetes are strings that Kubernetes reads from the client certificates or from the tokens.

For example, if you use a client certificate, then Kubernetes reads the common name (CN) and the organization (O) fields, and uses those strings for the username and the groups, respectively. Then, Kubernetes uses the username and groups to allow or deny the API request by using the RBAC policies.

For service accounts, which Kubernetes uses mainly to grant permissions to pods and to other resources to interact with the Kubernetes API server, Kubernetes adds only the serviceaccount string to the service account certificate or token to match RBAC policies. Service accounts are explained later in this course.

System Groups

System groups or virtual groups are predefined groups that are built into the system to grant specific permissions and access rights to certain categories of users. Kubernetes creates these groups during the setup to control access to critical components and resources within the cluster. These groups are a part of cluster roles and cluster role bindings. Thus, system groups are part of the RBAC system in Kubernetes.

**system:authenticated**

All authenticated users in the cluster, regardless of their role or namespace

**system:authenticated:oauth**

All authenticated users with an OAuth access token in the cluster

**system:unauthenticated**

Unauthenticated users, meaning those users who did not yet pass any authentication process

**system:masters**

Users with administrative access to the cluster Members of this group have full control over all resources and actions within the cluster, including cluster-level operations and management.

The Kubernetes API server adds the strings from system groups to the user and group strings from the client certificate or from the token, whenever they apply. Then, Kubernetes uses RBAC rules to approve or deny the request depending on the user and groups.

For example, if a user does not provide an access token or certificate, then Kubernetes assigns the system:unauthenticated group string to the request, and RBAC policies allow or deny the request depending on the RBAC rule configuration.

Users, Groups, and Identities in OpenShift

OpenShift includes the User resource, which defines regular users in the cluster. Regular users typically represent human users who interact with the OpenShift cluster. The OpenShift internal OAuth server automatically creates user resources when users first log in to the cluster through an IdP. You can also manually add users to the cluster by using the Kubernetes API.

OpenShift includes the Group resource, which defines groups in the cluster. Groups are logical collections of users with common access requirements. OpenShift supports users as members of one or more groups, for flexible definition of access control policies. Use groups to simplify the permissions in your cluster, to grant permissions to multiple users at the same time instead of individually. The OpenShift internal OAuth server automatically creates group resources when users first log in to the cluster through an OpenID Connect (OIDC) IdP. However, you must manually manage group resources in OpenShift for most of the supported IdPs, such as from LDAP.

OpenShift includes the Identity resource, which defines identities in the cluster. The identity resource keeps a record of successful authentication attempts from a specific user and IdP. Identities in OpenShift refer to the external identities of users. OpenShift maps user identities from IdPs to user resources. With identities, users can log in to OpenShift with their existing credentials from IdPs.

OAuth Server Mapping Methods

When a user first logs in to the cluster through an IdP, OpenShift creates a user resource and an identity resource, and maps them. OpenShift creates the user resource by using the preferred username from the IdP.

When different IdPs are configured in the OpenShift OAuth server, more than one IdP can provide the same username for a user, which causes collisions. With the mappingMethod parameter in the OAuth server, you can control how OpenShift establishes mappings between the provider's identities and the user resource, and avoid those collisions.

You can use one of the following values:

**claim**

OpenShift provisions a user with the identity's preferred username. However, it fails if a user with that username is already mapped to another identity. This value is the default for the mappingMethod parameter.

**add**

OpenShift provisions a user with the identity's preferred username. If a user with that username exists in OpenShift, then it maps the identity to the existing user, and adds the new identity to any existing identity mappings for the user. Use this method when you configure multiple IdPs that identify the same set of users and map to the same usernames.

**lookup**

OpenShift looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. Thus, with this method you must manually provision users. With this method, cluster administrators can set up identities and users manually, or by using an external process.

Authentication on a New OpenShift Cluster

OpenShift generates two authentication methods with administrative rights during the initial installation of the cluster: with the kubeadmin user, and with a client certificate for the system:admin user in a kubeconfig file.

**NOTE**

The kubeadmin user account is present only in self-managed OpenShift clusters. Cloud services do not use the kubeadmin user. Cloud services provide initial user credentials by using their cloud provider consoles and APIs in different ways.

The kubeadmin and system:admin users have the cluster-admin role, so OpenShift grants administrative access to these accounts by default. Use the kubeadmin user account during the cluster setup and initialization phase only. Red Hat recommends deleting the existing kubeadmin user after you create an administrator user with the cluster-admin role, to increase cluster security. Thus, configure an IdP in the OAuth server and create an administrator user with the cluster-admin role before you remove the kubeadmin user. Store the system:admin client certificate in a safe place so you can use it for recovery tasks.

The password for the kubeadmin user is available in the output that is generated when the OpenShift cluster finishes the installation, and in the corresponding log files. This log file also provides the path for the generated kubeconfig file that contains the client certificate for the system:admin user. The following example shows the OpenShift cluster output with the kubeadmin user details.

*...output omitted...*

INFO Install complete!

INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/home/lab/ocp4/auth/kubeconfig'

INFO Access the OpenShift web-console here: https://console-openshift-console.apps.ocp4.example.com

INFO Login to the console with user: "kubeadmin", and password: "***iMcm5-eCy2Q-PH66E-NkyPN***"

INFO Time elapsed: 10m52s

*...output omitted...*

|  |  |
| --- | --- |
|  | Path for the generated kubeconfig file that contains the client certificate |
|  | The kubeadmin user unique password |

**WARNING**

Removing the kubeadmin user cannot be undone. If you delete the kubeadmin user before creating another administrator user, then you lose administrator access to the OpenShift cluster and you must reinstall the cluster.

To remove the kubeadmin user from your cluster, log in as an administrator user and run the following command:

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

**REFERENCES**

For more information about authentication and authorization on OpenShift, refer to the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index>

For more information about removing the kubeadmin user, refer to the *Removing the kubeadmin User* in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#removing-kubeadmin>

[OAuth Working Group Specifications](https://oauth.net/specs/)

LDAP Authentication and Group Synchronization

Objectives

* Configure an LDAP identity provider and automate group synchronization between OpenShift OAuth and an LDAP server.

The LDAP Service

After the installation of a new Red Hat OpenShift Container Platform (RHOCP) cluster, only a kubeadmin user exists. A cluster administrator must next create users and groups, or more commonly, integrate a business system that both provides these entities and manages authentication. In doing so, an OpenShift cluster adopts a consistent corporate network authentication approach with other business systems, and inherits the existing security implementations in place, such as the company access restrictions that are available through a VPN-based connection.

Integrating an existing authentication method, and available users and groups in your organization, by using a Lightweight Directory Access Protocol (LDAP) server, is a common approach for many businesses. An LDAP directory server defines a standard application protocol for querying the system to authenticate users and groups. Additionally, LDAP integrations enable the use of Red Hat Identity Management (IdM) and Microsoft Active Directory (AD) services that many organizations use for managing authentication.

An LDAP server can be configured as an Identity Provider (IdP) for authentication bindings between the LDAP identity entries and cluster users through the OpenShift OAuth server. When users enter credentials for the cluster, the OAuth server initiates a connection and a corresponding query to the LDAP server, and creates bindings when matching unique entries are found.

Although this implementation is sufficient for authenticating the user, it is still necessary to define the access for the user, based on group memberships and roles. Furthermore, configuration within the OpenShift cluster is required to synchronize groups from the LDAP database, as well as to define the appropriate access and permissions for each group.

**NOTE**

This section does not cover LDAP administration in depth, but only the necessary information for configuring LDAP integrations in an RHOCP cluster.

Authentication by Using LDAP

The LDAP protocol defines a query language to retrieve information from database entries of a remote LDAP server, which can contain entries for users and groups within the organization. When implemented as an OpenShift OAuth IdP, the cluster can rely on the LDAP server to validate user-provided login credentials for the cluster.

An LDAP URI exists and uses a standard syntax for connection parameters to an LDAP server, and is necessary when configuring client connections to an LDAP server. To perform queries, the URI includes the search criteria to match with entries in the LDAP database. Many LDAP servers also require the use of an administrative bind DN, which is a set of privileged credentials with access to perform queries. This set of privileged credentials is included in the URI, when implemented.

A correctly formed search string describes the LDAP endpoint and the specific set of credentials, or a set of search criteria, such as user or group names, along with the bind Distinguished Name (bindDN) credentials when required for the interaction.

An LDAP URI is formed with the following standardized syntax:

ldap://host:port/basedn?attribute?scope?filter

**ldap**

The LDAP protocol designation. For LDAP over SSL, use the ldaps protocol. Red Hat recommends using StartTLS over regular ldap.

**host:port**

The LDAP server name and listening port. The default for ldap is localhost:389, and for ldaps is localhost:636.

**basedn**

The DN of the directory where a query begins. This directory is used as the root location for the search.

**attributes**

The target for the search, with the default as uid, which the LDAP lookup is intended to return. Multiple attributes are provided by a comma-separated list.

**scope**

The scope of the LDAP lookup. The scope is either one or sub. The default is sub if unspecified.

**filter**

An LDAP filter refines the results for the query. If omitted, the filter defaults to (objectClass=\*).

The following table provides an example set of information for an LDAP lookup for a specific username:

| **URL object** | **Value** |
| --- | --- |
| Connection protocol | ldap |
| Server name and port | ldap.example.com:389 |
| Base DN | dc=example,dc=com |
| Attributes | givenName,sn,cn |
| Query filter | uid=payden.tomcheck |

The preceding information would result in the following URL for an LDAP lookup:

ldap://ldap.example.com:389/dc=example,dc=com?givenName,sn,cn?(uid=payden.tomcheck)

The result of this lookup returns one or more usernames from directory entries that match the given uid value in the LDAP server.

In an OpenShift cluster, when a new user provides login credentials for the cluster, the OAuth LDAP IdP searches for the identity by using the configured LDAP search account. This LDAP search account is configured during the OAuth IdP configuration, and contains a Bind Distinguished Name (Bind DN) and a Bind Password. On a unique match from the search, the provided credentials are submitted to the LDAP server in a second interaction, and an authentication binding is created between an OpenShift user resource and the returned LDAP id and preferredUsername values.

If an authentication request has no matching result, then the query fails and no bind occurs.

Whereas the LDAP server provides the validation to create an authentication binding for the identity, a cluster administrator must also grant appropriate cluster roles for the identity through the OpenShift Role Based Access Controls (RBAC) settings. A new user configuration is complete when the binding and role are in place.

An administrator must synchronize groups from an LDAP server as an additional cluster-side configuration before a full LDAP IdP configuration is complete. It is necessary to create a cluster cron job to routinely query the LDAP server and to update the OpenShift groups to ensure consistency within the cluster to any organizational updates to the LDAP group entries.

LDAP CLI Queries

Although this course does not provide guidelines for LDAP administration, a tool is needed to inspect and validate available LDAP connectivity during cluster configuration.

The ldapsearch command is available from the openldap-clients RPM package, and provides a command-line query utility for LDAP interactions. The following example illustrates the format for an LDAP query that uses the ldapsearch command:

[user@host ~]$ **ldapsearch -x -D *bind\_dn* -H *ldap\_host\_URI* -b *search\_base* \**

**--filter *filter* --requestedAttribute *attributes* -W**

In this example, the following parameters are used:

* The -x option specifies using simple authentication.
* The -W option prompts you to provide the bindPassword secret for the specific -D *bind\_dn* option.
* The -b *search\_base* option denotes the root location for the lookup.
* The --filter and --requestedAttribute options provide further criteria to refine the search.

This approach provides a method for testing credentials for the LDAP server from the CLI, as well as retrieving listings for various LDAP entries, such as any defined groups. Although the previous example is too broad to provide meaningful results or detailed information, this approach establishes the validity of the credentials. Specifying additional attributes and filters returns more specific information, such as refining the query to search for an account with uid=1001, as shown in the following example:

[user@host ~]$ **ldapsearch -x -D "cn=administrator,dc=example,dc=com" \**

**-H ldap://ldap.example.com -b "uid=1001" -W "objectclass=account" uid**

Enter Password:

# extended LDIF

#

# LDAPv3

# base <uid=1001> (default) with scope subtree

# filter: (objectclass=account)

# requesting: ALL

# example.com

dn: dc=example,dc=com

*...output omitted...*

The options for ldapsearch queries, and their resulting output, are consistent with the needed information for the configuration of an LDAP server as an OAuth IdP.

The organizational units that describe the path to the correct location for an information lookup within the LDAP system are similar to the structure of directories and files within a file system.

The following example DN shows the uid, ou, and dc values for information within the LDAP server:

dn: uid=ptomchek,ou=users,dc=ocp4,dc=redhat,dc=com

The following commands illustrate a corresponding organizational structure for the same information within a file system:

[user@host ~]$ mkdir -p ocp4.redhat.com/users

[user@host ~]$ touch ocp4.redhat.com/users/ptomchek

[user@host ~]$ tree ocp4.redhat.com/

ocp4.redhat.com/

└── users

└── ptomchek

**NOTE**

For more information about LDAP administration and the ldapsearch command, refer to the Red Hat Directory Server documentation in the references section.

Configuring an LDAP Server as an OpenShift IdP

Integrating an LDAP server as an IdP for an OpenShift cluster requires the following process:

* Create an OpenShift secret resource to store the password for LDAP queries. This OpenShift secret resource contains the base64-encoded secret in the bindPassword field.
* Create an OpenShift configuration map resource that contains the certificate authority bundle (if the LDAP server uses encryption).

**NOTE**

Create this configuration map resource only if your LDAP IdP has a CA certificate that is not present in the default system keystore.

This OpenShift ConfigMap resource belongs in the openshift-config namespace.

* Update the CR to add the LDAP configuration to the existing OpenShift OAuth IdP entries.
* Redeploy the IdP CR to the cluster with the added LDAP IdP.

Create the bindPassword Secret Resource

The bindPassword for the IdP configuration is stored in an OpenShift secret resource. The following example is a YAML file for creating the bindPassword secret for an LDAP IdP:

apiVersion: v1

kind: Secret

metadata:

name: ldap-secret

namespace: openshift-config

type: Opaque

data:

bindPassword: ***base64-encoded-bind-password***

**NOTE**

Bear in mind that cluster administrators can read the password that is stored in the secret. Red Hat recommends using a service account with limited permissions as the bindDN within LDAP to avoid any security breaches.

Create the Certificate Configuration Map

LDAP configurations for OAuth IdPs use an OpenShift configuration map resource in the openshift-config namespace to define the certificate authority bundles.

Create an OpenShift configuration map resource that contains the certificate authority. You must store the certificate authority in the ca.crt key of the configuration map resource.

The following example is a YAML file that defines how to create a configuration map resource for the certificate bundle:

apiVersion: v1

kind: ConfigMap

metadata:

name: ca-config-map

namespace: openshift-config

data:

ca.crt: |

***CA\_certificate\_PEM***

Update the OAuth CR

The following example shows the configuration for defining an LDAP IdP in the OAuth CR YAML file:

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- name: **ldapidp**

mappingMethod: **claim**

type: LDAP

ldap:

attributes:

id:

- **dn**

email:

- **mail**

name:

- **cn**

preferredUsername:

- **uid**

bindDN: **'cn=administrator,dc=example,dc=com'**

bindPassword:

name: **ldap-secret**

**ca**:

name: ca-config-map

insecure: **false**

url: "**ldaps://ldaps.example.com/ou=users,dc=acme,dc=com?uid**"

|  |  |
| --- | --- |
|  | Provider names are shown on the web console login screen and in the list of configured IdPs. |
|  | Controls how mappings are established between this provider's identities and user objects. |
|  | List of attributes where the first non-empty attribute is used. At least one attribute is required. If none of the listed attributes has a value, then authentication fails. |
|  | List of attributes to use as the email address. The first non-empty attribute is used. |
|  | List of attributes to use as the display name. The first non-empty attribute is used. |
|  | List of attributes to use as the preferred username when provisioning a user for this identity. The first non-empty attribute is used. |
|  | Required DN that is used during the search phase, unless anonymous searches are allowed. Must be set if the bindPassword parameter is defined. |
|  | Required OpenShift secret resource that contains the bind password, unless anonymous searches are allowed. Must be set if the bindDN parameter is defined. |
|  | Optional: Reference to an OpenShift configuration map resource that contains the privacy enhanced mail (PEM) encoded certificate authority bundle to validate server certificates for the configured URL. Used only when the insecure option is false. |
|  | When true, no TLS connection is made to the server. When false, ldaps:// URLs connect by using TLS, and ldap:// URLs are upgraded to TLS. The insecure option must be set to false when ldaps:// URLs are in use, because these URLs always attempt to connect by using TLS. |
|  | An RFC 2255 URL, which specifies the LDAP host and maps the identity parameters in the LDAP database schema. This url value associates the user-supplied login credential with the specific entries in the LDAP server for identity lookups. |

**NOTE**

The OAuth CR contains all configured IdPs. Any additional IdP that is configured is appended to the CR. Replacing or removing any entries can result in the loss of access to your cluster.

Log in by Using the Added LDAP IdP

After you update the OAuth CR, the new LDAP IdP is available to the cluster. The IdP Name is shown among any other configured IdPs on the web console login page, as well as in the list of configured OAuth IdPs.

Log in with this added LDAP IdP by using the oc login command or the added selection from the OpenShift web console, and by providing a username and password that are available through the LDAP server. During authentication, OpenShift generates a search filter by combining the attribute and filter in the configured OAuth CR url parameter, with the provided username. Then, OpenShift applies this filter to the LDAP directory to find a unique entry.

From the preceding configuration example, if the user enters payden.tomcheck as a username, then a query for that value is sent to the LDAP server. This query attempts to find a unique match in the uid LDAP field that the OAuth CR url value "ldaps://ldaps.example.com/ou=users,dc=acme,dc=com?uid" specifies.

If you configure the mappingMethod parameter for the LDAP IdP entry in the OAuth CR as claim or add, then OpenShift uses the resulting LDAP identity to create a cluster User resource on the first attempt to access the cluster.

Authentication fails if the LDAP lookup does not return a unique match. In this case, no binding is created, no user is added to the cluster, and access is denied.

OpenShift Group Synchronization with LDAP

Configuring an OAuth IdP for an LDAP server provides only a mechanism to validate and create users in the cluster, based on entries that are provided through the LDAP instance. Businesses also rely on LDAP to provide the groups that are used throughout the organization and to define appropriate RBAC configurations. By configuring LDAP group synchronization, a business can manage group memberships in a single location, and OpenShift can use these groups within the cluster.

OpenShift can synchronize groups from an LDAP server, which enables you to mirror the available LDAP groups within the cluster. LDAP group synchronization requires a sync configuration file with a client configuration and a query definition. Additionally, user-defined mappings can optionally provide granular details for establishing the custom relationships between group entries in the LDAP server and the corresponding OpenShift groups that you require.

Group synchronization requires a project where the pods run, a service account to perform the work, a cluster role where the right permissions are defined, and cluster role binding to associate the cluster role with the service account. When the synchronization configuration is in place, the synchronization of groups is automated by using a cron job to schedule the recurring execution of the synchronization task.

LDAP Sync Configuration File

To synchronize LDAP groups, create an LDAPSyncConfig resource that contains the client LDAP configuration details and query definition for the LDAP group entities that are needed within OpenShift.

The LDAP client configuration part defines the connection parameters for the LDAP server, as well as the bindDN account and bindPassword secret for performing queries. This configuration uses the same connection and credentials that are used during the OAuth LDAP IdP implementation.

The following YAML code is an example of LDAP client configuration details:

url: ldap://1.2.3.4:389

bindDN: cn=administrator,dc=example,dc=com

bindPassword: ***password***

insecure: false

ca: ca-bundle.crt

The LDAP query definition part describes the format of entries within the LDAP server that provide the groups to mirror to the OpenShift cluster.

The following YAML code is an example of an LDAP query definition:

baseDN: ou=groups,dc=example,dc=com

scope: sub

derefAliases: never

timeout: 0

filter: (objectClass=group)

pageSize: 0

By assembling the client configuration details together with the query definition, you can create an LDAPSyncConfig resource by using the parameters for your LDAP server:

kind: LDAPSyncConfig

apiVersion: v1

url: ldap://example.com:389

insecure: false

rfc2307:

groupsQuery:

baseDN: "ou=groups,dc=example,dc=com"

scope: sub

derefAliases: never

pageSize: 0

groupUIDAttribute: dn

groupNameAttributes: [ cn ]

groupMembershipAttributes: [ member ]

usersQuery:

baseDN: "ou=users,dc=example,dc=com"

scope: sub

derefAliases: never

pageSize: 0

userUIDAttribute: dn

userNameAttributes: [ mail ]

tolerateMemberNotFoundErrors: false

tolerateMemberOutOfScopeErrors: false

|  |  |
| --- | --- |
|  | When true, no TLS connection is made to the server. When false, ldaps:// URLs connect by using TLS, and ldap:// URLs are upgraded to TLS. The insecure option must be set to false when ldaps:// URLs are in use, because these URLs always attempt to connect by using TLS. |
|  | The field that corresponds to the unique identifier field on the LDAP server for a group. |
|  | The attribute for the name of the group. |
|  | The field that corresponds the unique identifier field of the LDAP server for a user. |
|  | The attribute for the name of the user. |

Example Active Directory Configuration

Active Directory (AD) is another source for providing group definitions for an OpenShift cluster. The AD configuration requires an LDAP query definition and attributes to represent users within the OpenShift group definitions.

The following YAML code is an example LDAP synchronization configuration for an AD schema:

kind: LDAPSyncConfig

apiVersion: v1

url: ldap://ad.example.com:389

activeDirectory:

usersQuery:

baseDN: "ou=users,dc=example,dc=com"

scope: sub

derefAliases: never

filter: (objectclass=person)

pageSize: 0

userNameAttributes: [ mail ]

groupMembershipAttributes: [ memberOf ]

|  |  |
| --- | --- |
|  | The attribute to use as the username in the OpenShift group record. |
|  | The attribute on the user that stores the group membership information. |

Note the schema that AD uses to map users to the corresponding group membership through the memberOf attribute.

Running an LDAP Synchronization

After the LDAPSyncConfig file is created, an administrator can use the file to synchronize groups from the LDAP server to the OpenShift cluster. The following example command uses the oc adm groups sync command to synchronize all groups from the LDAP server to the cluster by using the config.yaml file that contains the LDAPSyncConfig details:

[user@host ~]$ **oc adm groups sync --sync-config=config.yaml --confirm**

If the --confirm parameter is omitted, then the command performs a dry run of the operation. An initial dry run is advised when testing a new configuration, and to review the resulting groups that the cluster will inherit.

Automating LDAP Group Synchronizations

LDAP group synchronizations are automated through the use of a cron job. Configure this cron job from within a designated OpenShift project for this purpose. Execute the job by using a service account with administrative access to the cluster to manage groups.

Start by creating the project for the cron job to run. The following command creates a project named ldap-group-sync for this purpose:

[user@host ~]$ **oc new-project ldap-group-sync**

The client bindPassword secret resource and the configuration map resource that are used during the configuration of the LDAP IdP are also needed for this task. Additionally, create a service account with an appropriate cluster role and cluster role binding for the synchronization. The following YAML code shows an example definition for creating a service account named ldap-group-sync-acct:

kind: ServiceAccount

apiVersion: v1

metadata:

name: ldap-group-sync-acct

namespace: ldap-group-sync

Next, define a cluster role, as shown in the following example:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: ldap-group-sync-acct

rules:

- apiGroups:

- ''

- user.openshift.io

resources:

- groups

verbs:

- get

- list

- create

- update

|  |  |
| --- | --- |
|  | The core API group is identified by an empty string. |

Then, define a cluster role binding to bind the previous cluster role to the existing service account.

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: ldap-group-sync-acct

subjects:

- kind: ServiceAccount

name: ldap-group-sync-acct

namespace: ldap-group-sync

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: ldap-group-sync-acct

The cluster is configured to communicate with the LDAP IdP, and a service account is available to configure OpenShift groups from the LDAP group entities. The following YAML code is an example configuration map file to specify the sync configuration:

kind: ConfigMap

apiVersion: v1

metadata:

name: ldap-group-sync-acct

namespace: ldap-group-sync

data:

sync.yaml: |

kind: LDAPSyncConfig

apiVersion: v1

url: ldaps://1.2.3.4:389

insecure: false

bindDN: cn=administrator,dc=example,dc=com

bindPassword:

file: "/etc/secrets/bindPassword"

ca: /etc/ldap-ca/ca.crt

rfc2307:

groupsQuery:

baseDN: "ou=groups,dc=example,dc=com"

scope: sub

filter: "(objectClass=groupOfMembers)"

derefAliases: never

pageSize: 0

groupUIDAttribute: dn

groupNameAttributes: [ cn ]

groupMembershipAttributes: [ member ]

usersQuery:

baseDN: "ou=users,dc=example,dc=com"

scope: sub

derefAliases: never

pageSize: 0

userUIDAttribute: dn

userNameAttributes: [ uid ]

tolerateMemberNotFoundErrors: false

tolerateMemberOutOfScopeErrors: false

Finally, define and deploy a cron job for the LDAP group synchronization. The following YAML code is an example cron job definition that could be customized, such as by adapting the cron-specified schedule to meet business needs:

kind: CronJob

apiVersion: batch/v1

metadata:

name: ldap-group-sync-acct

namespace: ldap-group-sync

spec:

schedule: "​\*/30 \* \* \* \*​"

concurrencyPolicy: Forbid

jobTemplate:

spec:

backoffLimit: 0

ttlSecondsAfterFinished: 1800

template:

spec:

containers:

- name: ldap-group-sync

image: "registry.redhat.io/openshift4/ose-cli:latest"

command:

- "/bin/bash"

- "-c"

- "oc adm groups sync --sync-config=/etc/config/sync.yaml --confirm"

volumeMounts:

- mountPath: "/etc/config"

name: "ldap-sync-volume"

- mountPath: "/etc/secrets"

name: "ldap-bind-password"

- mountPath: "/etc/ldap-ca"

name: "ldap-ca"

volumes:

- name: "ldap-sync-volume"

configMap:

name: "ldap-group-sync-acct"

- name: "ldap-bind-password"

secret:

secretName: "ldap-secret"

- name: "ldap-ca"

configMap:

name: "ca-config-map"

restartPolicy: "Never"

terminationGracePeriodSeconds: 30

activeDeadlineSeconds: 500

dnsPolicy: "ClusterFirst"

serviceAccountName: "ldap-group-sync-acct"

|  |  |
| --- | --- |
|  | The defined schedule for the job in cron format, to show every 30 minutes. |
|  | The time, in seconds, to keep completed sync information, to correspond to the preceding cron schedule. |
|  | The LDAP group sync command to execute by using the sync configuration file that is defined in the configuration map. **The oc adm groups sync command is defined on a single line** |
|  | The LDAP IdP bindDN and bindPassword. |

After the file is created, the configured cron job executes the group synchronization command during the specified schedule to ensure that consistent group configurations that are available from the LDAP IdP are added to the OpenShift cluster. This job runs within the project namespace, by using the service account, at an interval that the cron time spec defines in the CronJob definition.

**REFERENCES**

For more information about integrating LDAP IdP for a RHOCP cluster, refer to the *Configuring LDAP Identity Providers* section in the *Authentication and Authorization* chapter in the Red Hat OpenShift Container Platform 4.14 *OpenShift Container Platform* documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#configuring-ldap-identity-provider>

For more information about LDAP administration and the ldapsearch command, refer to the Red Hat Directory Server documentation at <https://access.redhat.com/documentation/en-us/red_hat_directory_server/11/html-single/administration_guide/index#Examples-of-common-ldapsearches>

Guided Exercise: LDAP Authentication and Group Synchronization

Configure an LDAP identity provider and automate group synchronization between OpenShift OAuth and an LDAP server.

**Outcomes**

* Configure an LDAP identity provider (IdP) for RHOCP.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Your company requires you to configure a Red Hat Directory Services (RHDS) Identity Provider (IdP), which is available on the rhds machine as an LDAP IdP for RHOCP.

**IMPORTANT**

LDAP administration is outside the scope of this course. The RHDS server is already correctly configured for this exercise.

1. As the admin user, locate and then navigate to the Red Hat OpenShift web console.
   1. Use the terminal to 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**

*...output omitted...*

* 1. Identify the URL for the OpenShift web console.
  2. [student@workstation ~]$ **oc whoami --show-console**

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

* 1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com. Either type the URL in a web browser, or right-click and select **Open Link** from the terminal.

1. Log in to the OpenShift web console as the admin user.
   1. Click **Red Hat Identity Management** and log in as the admin user with the redhatocp password.
2. Navigate to the Configuration page for OAuth in the RHOCP cluster web console.
   1. Click **Administration** → **Cluster Settings** from the sidebar.
   2. Click the **Configuration** tab to browse the list of configurable resources.

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

* 1. Scroll through the alphabetical list of configuration resources and select **OAuth**.

|  |
| --- |
| A close-up of a text  Description automatically generated |

1. View the existing YAML configuration for the Red Hat Identity Management IdP.
   1. Click the **YAML** tab and scroll down to the bottom of the configuration file.

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

* 1. **NOTE**
  2. You can view the cluster OAuth configuration from the CLI in YAML format, by using the oc get oauth cluster --output yaml command.
  3. You can also view the cluster OAuth configuration details from the CLI, by using the oc describe oauth command.
  4. The settings for the existing Red Hat Identity Management IdP are shown in the spec keys on lines 57-78.

spec:

identityProviders:

- ldap:

attributes:

email:

- mail

id:

- dn

name:

- cn

preferredUsername:

- uid

bindDN: 'uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com'

bindPassword:

name: ldap-secret

ca:

name: ca-config-map

insecure: false

url: >-

ldap://idm.ocp4.example.com/cn=users,cn=accounts,dc=ocp4,dc=example,dc=com?uid

mappingMethod: claim

name: Red Hat Identity Management

type: LDAP

1. From the CLI, test the RHDS credentials to ensure a valid LDAP connection, before configuring the additional IdP.
   1. Open a terminal on the workstation, and create an ldapsearch query with the information in the following table:

| **Query option** | **Value** |
| --- | --- |
| Bind DN (-D) | cn=Directory Manager |
| URI (-H) | ldaps://rhds.ocp4.example.com |
| Bind password (-w) | redhatocp |

* 1. From the CLI, test the ldapsearch connection by using the authored query. With the -w option that is shown, you can supply the password, in plain text.

[student@workstation ~] **ldapsearch -D "cn=Directory Manager" \**

**-w redhatocp -H ldaps://rhds.ocp4.example.com**

# extended LDIF

#

# LDAPv3

# base <> (default) with scope subtree

# filter: (objectclass=\*)

# requesting: ALL

#

# example.com

dn: dc=example,dc=com

objectClass: top

objectClass: domain

dc: example

description: dc=example,dc=com

# people, example.com

dn: ou=people,dc=example,dc=com

objectClass: top

objectClass: organizationalUnit

ou: people

# kristendelgado, people, example.com

dn: uid=kristendelgado,ou=people,dc=example,dc=com

objectClass: top

objectClass: account

objectClass: posixAccount

objectClass: shadowAccount

objectClass: nsMemberOf

cn: Kristen Delgado

uid: kristendelgado

uidNumber: 10001

gidNumber: 101

homeDirectory: /home/kristendelgado

loginShell: /bin/bash

gecos: kristendelgado

shadowLastChange: 0

shadowMax: 0

shadowWarning: 0

userPassword:: e1NTSEF9Wm1sMWd1WjJLajlRM1dKZGlFVnV6aTNEaCs5NzFPeFg=

memberOf: cn=administrators,ou=people,dc=example,dc=com

# administrators, people, example.com

dn: cn=administrators,ou=people,dc=example,dc=com

objectClass: top

objectClass: groupOfUniqueNames

cn: administrators

uniqueMember: uid=kristendelgado,ou=people,dc=example,dc=com

# search result

search: 2

result: 0 Success

# numResponses: 5

# numEntries: 4

1. In the web console, add a new IdP for RHDS by using the validated information.
   1. Return to the previous page by clicking the **Details** tab.

**NOTE**

The existing Red Hat Identity Management IdP is shown in the list of available IdPs.

* 1. In the **Identity providers** section, select LDAP from the **Add** dropdown.

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

* 1. Complete the corresponding form fields by using the information in the following table:

| **Field** | **Value** |
| --- | --- |
| Name | Red Hat Directory Server |
| URL and BaseDN | ldaps://rhds.ocp4.example.com/dc=example,dc=com?uid |
| Bind DN | cn=Directory Manager |
| Bind password | redhatocp |
| Email | mail |

* 1. **NOTE**
  2. Use the default values for all other fields.
  3. In the CA File field, enter the certificate for the RHDS server from the rhds classroom machine, which is provided on the workstation machine in the file ~/DO380/labs/auth-ldap/rhds\_ca.crt.

[student@workstation ~] **cat ~/DO380/labs/auth-ldap/rhds\_ca.crt**

-----BEGIN CERTIFICATE-----

*...output omitted...*

* 1. Click **Add**.

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

1. Verify the addition of the new IdP.
   1. View the new entry in the IdP list on the OAuth details page.

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

* 1. Review the configuration additions on the **YAML** tab on lines 78-98.

spec:

identityProviders:

- ldap:

attributes:

email:

- mail

id:

- dn

name:

- cn

preferredUsername:

- uid

bindDN: 'cn=Directory Manager'

bindPassword:

name: ldap-bind-password-cjckt

ca:

name: ca-config-map

insecure: false

url: >-

ldaps://rhds.ocp4.example.com/dc=example,dc=com?uid

mappingMethod: claim

name: Red Hat Directory Server

type: LDAP

* 1. From the upper-right dropdown menu, click **Log out** to log out of the web console and return to the authentication page.
  2. The new IdP appears in the list as Red Hat Directory Server.

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

* 1. **NOTE**
  2. It might take several minutes for the cluster to redeploy the pods and for the new IdP to appear.

1. Select the Red Hat Directory Server IdP, and verify the authentication function by using the following credentials to log in to the cluster.

| **Username** | **Password** |
| --- | --- |
| kristendelgado | redhat123 |

**Finish**

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

[student@workstation ~]$ **lab finish auth-ldap**

Guided Exercise: Automate LDAP Group Synchronization

Automate group synchronization between OpenShift OAuth and an LDAP server.

**Outcomes**

* Configure an automated group synchronization for the Red Hat Directory Services (RHDS) LDAP identity provider (IdP).

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

**NOTE**

This exercise requires the completion of the previous Guided Exercise to configure the IdP. Ensure that you completed the previous section before proceeding.

Configure an automated group synchronization for the secondary RHDS IdP to maintain updated user and group information.

The implementation configures the cluster administrator privilege for the administrators group, which includes the kristendelgado user.

The RHDS LDAP IdP information is in the following table:

| **Parameter** | **Value** |
| --- | --- |
| Bind DN (-D) | cn=Directory Manager,dc=example,dc=com |
| URI (-H) | ldaps://rhds.ocp4.example.com |
| Password (-w) | redhatocp |

1. From the CLI, test the RHDS connection by using the ldapsearch command with the information in the table.

[student@workstation ~]$ **ldapsearch -D "cn=Directory Manager" \**

**-w redhatocp -H ldaps://rhds.ocp4.example.com**

# extended LDIF

#

# LDAPv3

# base <> (default) with scope subtree

# filter: (objectclass=\*)

# requesting: ALL

#

# example.com

dn: dc=example,dc=com

objectClass: top

objectClass: domain

dc: example

description: dc=example,dc=com

# people, example.com

dn: ou=people,dc=example,dc=com

objectClass: top

objectClass: organizationalUnit

ou: people

# kristendelgado, people, example.com

dn: uid=kristendelgado,ou=people,dc=example,dc=com

objectClass: top

objectClass: account

objectClass: posixAccount

objectClass: shadowAccount

objectClass: nsMemberOf

cn: Kristen Delgado

uid: kristendelgado

uidNumber: 10001

gidNumber: 101

homeDirectory: /home/kristendelgado

loginShell: /bin/bash

gecos: kristendelgado

shadowLastChange: 0

shadowMax: 0

shadowWarning: 0

userPassword::

e1NTSEF9Wm1sMWd1WjJLajlRM1dKZGlFVnV6aTNEaCs5NzFPeFg=

memberOf: cn=administrators,ou=people,dc=example,dc=com

# administrators, people, example.com

dn: cn=administrators,ou=people,dc=example,dc=com

objectClass: top

objectClass: groupOfUniqueNames

cn: administrators

uniqueMember: uid=kristendelgado,ou=people,dc=example,dc=com

# search result

search: 2

result: 0 Success

# numResponses: 5

# numEntries: 4

1. Verify that you can log in to the cluster as the kristendelgado user with redhat123 as the password, to ensure that the user from the RHDS IdP is available.

[student@workstation ~]$ **oc login -u kristendelgado -p redhat123 \**

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

Login successful.

*...output omitted...*

1. Change to the ~/DO380/labs/auth-sync/sync directory and switch back to using the admin user and the redhatocp password.
   1. Change to the ~/DO380/labs/auth-sync/sync directory.
   2. [student@workstation ~]$ **cd ~/DO380/labs/auth-sync/sync**

[student@workstation sync]$

* 1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

*...output omitted...*

1. Create the auth-rhds-sync project.

[student@workstation sync]$ **oc new-project auth-rhds-sync**

Now using project "auth-rhds-sync" on server "https://api.ocp4.example.com:6443".

*...output omitted...*

1. Create a service account with permissions to synchronize groups from the RHDS server.
   1. Create a service account called rhds-group-syncer.
   2. [student@workstation sync]$ **oc create sa rhds-group-syncer**

serviceaccount/rhds-group-syncer created

* 1. Create the rhds-group-syncer cluster role with the get,list,create,update verbs.

[student@workstation sync]$ **oc create clusterrole rhds-group-syncer \**

**--verb get,list,create,update --resource groups**

clusterrole.rbac.authorization.k8s.io/rhds-group-syncer created

* 1. Create the cluster role binding for the rhds-group-syncer service account and cluster role.

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

**rhds-group-syncer -z rhds-group-syncer**

clusterrole.rbac.authorization.k8s.io/rhds-group-syncer added: "rhds-group-syncer"

1. Create a secret that contains the provided RHDS bind password.

[student@workstation sync]$ **oc create secret generic rhds-secret \**

**--from-literal bindPassword='redhatocp'**

secret/rhds-secret created

1. Create the configuration files for the RHDS automated group synchronization.
   1. Create the RHDS synchronization configuration from the example that is provided in the rhds-sync.yaml file in the working directory to supply the bind DN and password.

kind: LDAPSyncConfig

apiVersion: v1

url: ldaps://rhds.ocp4.example.com:636

bindDN: **'cn=Directory Manager'**

bindPassword:

file: **/etc/secrets/bindPassword**

ca: **/etc/config/ca.crt**

augmentedActiveDirectory:

groupsQuery:

baseDN: "ou=people,dc=example,dc=com"

scope: sub

derefAliases: never

pageSize: 0

groupUIDAttribute: dn

groupNameAttributes: [ cn ]

usersQuery:

baseDN: "ou=people,dc=example,dc=com"

scope: sub

derefAliases: never

filter: (objectclass=account)

pageSize: 0

userNameAttributes: [ uid ]

groupMembershipAttributes: [ memberOf ]

* 1. Create a configuration map that contains the LDAPSyncConfig file and the trusted certificate.

[student@workstation sync]$ **oc create configmap rhds-config \**

**--from-file rhds-sync.yaml=rhds-sync.yaml,ca.crt=rhds\_ca.crt**

configmap/rhds-config created

1. Create the cron job for the automated schedule to synchronize groups every minute.
   1. Update the cron job configuration for the group synchronization example that is provided in the rhds-groups-cronjob.yaml file in the working directory to supply the secret and service account information.

apiVersion: batch/v1

kind: CronJob

metadata:

name: rhds-group-sync

namespace: auth-rhds-sync

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

template:

spec:

restartPolicy: Never

containers:

- name: ldap-group-sync

image: "registry.ocp4.example.com:8443/openshift4/ose-cli:v4.12"

command:

- "/bin/sh"

- "-c"

- "oc adm groups sync --sync-config=/etc/config/rhds-sync.yaml --confirm"

volumeMounts:

- mountPath: "/etc/config"

name: "ldap-sync-volume"

- mountPath: "/etc/secrets"

name: "ldap-bind-password"

volumes:

- name: "ldap-sync-volume"

configMap:

name: "rhds-config"

- name: "ldap-bind-password"

secret:

secretName: **"rhds-secret"**

serviceAccountName: **rhds-group-syncer**

serviceAccount: **rhds-group-syncer**

|  |  |
| --- | --- |
|  | Write this value in a single line. |

* 1. Create the cron job from the rhds-groups-cronjob.yaml file.

[student@workstation sync]$ **oc create -f rhds-groups-cronjob.yaml**

*...output omitted...*

cronjob.batch/rhds-group-sync created

* 1. Wait for one minute for the cron job to trigger, and verify the synchronization of the administrators group from RHDS.

[student@workstation sync]$ **oc get groups**

NAME USERS

Default SMB Group

**administrators kristendelgado**

*...output omitted...*

1. Apply the cluster-admin role to the administrators group.

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

**cluster-admin administrators**

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "administrators"

1. Log in to the cluster as the kristendelgado user and verify that the user has cluster administrator privileges from the administrators group membership.
   1. Log in to the cluster.

[student@workstation sync]$ **oc login -u kristendelgado -p redhat123**

Login successful.

*...output omitted...*

* 1. Verify you can perform an administrative task.

[student@workstation sync]$ **oc auth can-i create users -A**

yes

1. Change to the student HOME directory.

[student@workstation sync]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish auth-sync**

## **OIDC Authentication and Group Claims**

### **Objectives**

* Configure an OIDC identity provider and automate group synchronization between OpenShift OAuth and an OIDC server.

### **OpenID Connect**

You can use OpenShift to configure OpenID Connect (OIDC) identity providers (IdPs) for synchronizing users and groups.

OIDC is a set of standards for delegating the authentication of a user who accesses a protected resource. OIDC provides a way for applications to verify the identity of users and to obtain user profile information.

OIDC is the standard for cloud authentication, social authentication, single sign-on (SSO), and two-factor authentication (2FA). OIDC removes the responsibility of setting, storing, and managing passwords locally, which is often associated with credential-based data breaches, and which needs administration resources.

Examples of OIDC providers that are tested and supported on OpenShift include Google, Microsoft identity platform, and Keycloak. Red Hat provides Red Hat Single Sign-On (SSO) to extend the capabilities of the OpenShift internal OAuth, and to serve as the solution for an OIDC identity infrastructure. Red Hat SSO can run on bare metal or virtualized environments, or as pods on OpenShift.

### NOTE

For more information about Red Hat SSO, refer to the Product Documentation for Red Hat Single Sign-On 7.6 at <https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.6>

To see the full list of supported OIDC providers on OpenShift, you can refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#identity-provider-oidc-supported_configuring-oidc-identity-provider>

#### **OIDC Tokens**

OIDC relies on the JavaScript Object Signing and Encryption (JOSE) set of standards. The primary standard within JOSE is JSON Web Token (JWT), which serves as a standardized format for representing information in a readable piece of text data. The following list shows some JWT advantages:

* Can parse in many programming languages
* Can propagate across networks
* Validates message integrity without relying on external validators or resource-intensive checks

Three parts, which are separated by a period, form the JWT token: the header, the payload, and the hash signature.

The header contains the type of the token and the signing algorithm.

The payload contains the user claims. User claims are attributes for the user with details about their identity, profile, privileges, and group membership. Examples of user claims are the sub, iat, exp, or iss parameters, which are the identity for the user, the issue date, the expiration date, and the token issuer, respectively.

Finally, the signature is composed of the concatenation of the encoded header, the encoded payload, and the result of applying a signing algorithm.

#### **OIDC in OpenShift**

An identity broker is a service that connects clients with IdPs. The identity broker delegates the authentication of the user to the external IdP. OpenShift includes a built-in OAuth server that you can configure to determine the user's identity from the configured IdPs. Thus, the OpenShift built-in OAuth server acts as an identity broker.

When a user tries to log in to OpenShift, the OpenShift built-in OAuth server redirects the user to a login screen to choose from the configured IdPs. For OIDC IdPs, you must previously configure the OIDC client in the IdP that authenticates the user to OpenShift. Then, use the connection parameters and credentials from the OIDC client to configure the OIDC IdP in the OpenShift built-in OAuth server. After authenticating the user in the configured IdP, the internal OAuth server creates an access token for the request and returns the token. The built-in OAuth server, as for any other IdP, also creates or updates the user resources. If you define OIDC group claims, then the OAuth server also creates or updates the groups.

#### **OIDC Claims**

OIDC claims are key/value pairs that contain information about the user. OpenShift reads user claims from the JWT token that the IdP issues, and uses these user claims to populate the user, identity, and group resources. You must configure one claim to use as the user's identity. The default identity claim is sub, which stands for subject identifier.

You can configure additional user parameters in other standard claims, such as the preferred username, display name, or email address. For any of those user parameters, you can specify multiple claims. OpenShift uses the first claim with a non-empty value.

The following list includes some standard claims that are defined in OIDC:

**sub**

The remote identity for the user at the IdP.

**preferred\_username**

The preferred username when provisioning a user, which typically corresponds to the username in the authentication system for the user.

**email**

Email address.

**name**

Display name.

### NOTE

To see the full list of standard claims that are defined in OIDC, refer to <https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims>

#### **OIDC Group Claims**

You can also define additional claims for parameters that OIDC does not define as standard claims. One important additional claim for users is the group claim. OpenShift uses the group claim to map the group membership of a user in the IdP to a group object. Then, you can use role-based access control (RBAC) objects in OpenShift to assign permissions to that group, instead of assigning permissions to individual users.

### NOTE

The Red Hat Communities of Practice Group Sync Operator GitHub repository provides an unsupported operator for synchronizing OIDC groups with external providers that cannot provide group claims as part of their tokens. You can find more information in <https://github.com/redhat-cop/group-sync-operator>

### **Configuring OIDC IdP**

Follow the next steps to integrate an OIDC IdP into the OpenShift cluster:

1. Obtain the client ID and the client secret from the OIDC IdP client for the OpenShift integration.
2. Create an OpenShift secret object, which contains the client secret that is obtained from the OIDC client configuration, in the openshift-config namespace.
3. Create an OpenShift configuration map object, which contains the certificate authority bundle in the ca.crt file parameter, in the openshift-config namespace (required only if the CA certificate is not configured as a system-wide CA).
4. Create the OAuth CR YAML file to include the OIDC IdP.
5. Apply the configuration file to the OAuth CR.

#### **Create the OAuth CR YAML File**

After creating the OpenShift secret object that contains the client secret, and the configuration object that contains the certificate authority bundle (if necessary), you can create the OAuth CR YAML file that contains the information to configure the OIDC IdP. If you add an OIDC IdP to the OAuth CR, then you must include the OIDC IdP information in the identityProviders array. You can add multiple OIDC IdPs to the OAuth CR YAML file.

### IMPORTANT

The identityProviders array in the OAuth CR might not be empty. If you remove other IdPs when adding your OIDC IdP, then you cannot log in to the cluster through those IdPs.

The following example shows a minimal configuration file for Red Hat SSO integration to OpenShift. The settings might differ for other OIDC providers, and you must work with your vendor or identity administrator to get all the necessary attributes for your specific setup.

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- name: ***oidc\_provider\_name***

mappingMethod: claim

type: OpenID

openID:

clientID: ***oidc\_clientid***

clientSecret:

name: ***secret\_name***

ca:

name: ***config\_map\_name***

claims:

preferredUsername:

- preferred\_username

- email

name:

- nickname

- given\_name

- name

email:

- custom\_email\_claim

- email

groups:

- groups

issuer: ***https://external\_idp\_url.com***

|  |  |
| --- | --- |
|  | OpenShift prefixes the value of the identity claim with the provider name to form an identity name, and uses the identity name to build the redirect URL. |
|  | The client ID for the existing client in the IdP. You must enable the client to redirect to https://oauth-openshift.apps.*cluster\_name*.*cluster\_domain*/oauth2callback/*idp\_provider\_name*. |
|  | The name for the OpenShift secret object that contains the client secret. |
|  | (Optional) The name for the OpenShift configuration map object that contains the certificate authority bundle. |
|  | The list of claims to use as the identity, such as the preferred username, email address, or groups. |
|  | The URL for the IdP. OpenShift accepts only HTTPS URLs. |

#### **Log in to the Cluster Through the OIDC IdP**

After adding the OIDC IdP to the cluster, you can log in through the IdP by using the OpenShift web console, and providing the username and the password.

If your OIDC IdP supports the grant flow for resource owner password credentials (ROPC), then you can also log in through the IdP by using the oc login command with the username and the password.

If your OIDC IdP does not support the ROPC grant flow, then you receive a You must obtain an API token login error when you use the oc login command with the username and the password. Then, you must first get an OAuth access token and use it to log in by using the oc login --token=*acces\_token* command. You can get the OAuth access token by logging in through the OpenShift web console and then clicking **Help** → **Command line tools** → **Copy login command**. You can also request the OAuth access token through the OpenShift REST API.

### NOTE

Requesting an OAuth access token through the OpenShift REST API is out of the scope of this course. For more information about this topic, refer to <https://access.redhat.com/solutions/6610781>

#### **IdP Mapping Methods**

IdP mapping methods apply to any IdP. If you configure the mappingMethod parameter for the OIDC IdP as claim or add, then OpenShift establishes mappings between the provider's identity and the User object the first time that a user logs in to the cluster.

#### **OAuth Access Tokens**

The first time that a user logs in to the cluster, the OpenShift built-in OAuth server creates an OAuth token to authenticate to the API. OAuth access tokens are common to any IdP. OpenShift renews the token to authenticate to the API every time that the user logs in to the cluster. As a user, you can list all your user-owned OAuth access tokens by using the following command. This command lists all the user-owned OAuth access tokens from any IdP that is configured in the OpenShift built-in OAuth server.

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

NAME CLIENT NAME CREATED EXPIRES ...

sha256~9BZ3... openshift-... 69m 2023-06-14 13:24:42 +0000 UTC ...

sha256~lm1O... console 75m 2023-06-14 13:19:20 +0000 UTC ...

sha256~xmpC... openshift-... 83m 2023-06-14 13:11:24 +0000 UTC ...

If you modify a user parameter in the OIDC IdP, then the OIDC IdP does not automatically synchronize to OpenShift. For example, if you remove a user account from the OIDC IdP and the user is logged in to OpenShift, then the user can still perform tasks in OpenShift until they log out, because they still have a valid token that the OpenShift built-in OAuth server emits.

For this reason, after you modify a parameter in the OIDC IdP, you must manually remove all the user access tokens from OpenShift to force a user logout. Then, after the user logs in again, OpenShift synchronizes the new user parameters.

You can use the following command to remove all the user access tokens from OpenShift:

[user@host ~]$ **oc delete oauthaccesstoken $(oc get oauthaccesstoken -o \**

**jsonpath='{.items[?(@.userName=="*username*")].metadata.name}')**

### REFERENCES

For more information about OIDC, refer to the OpenID specifications web page at <https://openid.net/developers/specs/>

For more information about how to configure OIDC IdPs on OpenShift, refer to the Configuring an OpenID Connect Identity Provider section in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#identity-provider-oidc-about_configuring-oidc-identity-provider>

For more information about how to configure a GitHub or GitHub Enterprise IdP on OpenShift, refer to the Configuring a GitHub or GitHub Enterprise Identity Provider section in the Red Hat OpenShift Container Platform 4.14 Authentication and Authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#identity-provider-overview_configuring-github-identity-provider>

For more information about how to configure Azure Active Directory authentication on OpenShift, refer to the Configure Azure Active Directory Authentication for an Azure Red Hat OpenShift 4 Cluster (CLI) section in the Azure Red Hat OpenShift documentation at <https://learn.microsoft.com/en-us/azure/openshift/configure-azure-ad-cli>

## **Guided Exercise: OIDC Authentication and Group Claims**

Configure an OIDC identity provider and automate group synchronization between OpenShift OAuth and an OIDC server.

**Outcomes**

* Configure Red Hat Single Sign-On (SSO) as an OIDC identity provider (IdP) for OpenShift.
* Synchronize users and groups from Red Hat SSO to OpenShift.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Your company requires you to configure Red Hat SSO, which is running in the sso machine as an OIDC IdP for OpenShift, to automate user and group synchronization between OpenShift OAuth and the Red Hat SSO OIDC server.

As a use case, the lab script configures Red Hat SSO to include the external\_providers realm. The following table provides the details for three users and two groups that are available in this realm. The password for the three users is redhat\_sso.

| **First name** | **Last name** | **Username** | **Group membership** |
| --- | --- | --- | --- |
| Abby | Quincy | abbyquincy | contractors |
| Fricis | Ritcher | fricisritcher | contractors |
| Jaya | Lamont | jayalamont | partners |

The company requests that you give read access to the auth-oidc OpenShift project for users in the partners group. Additionally, users in the contractors group need to be able to edit objects in the auth-oidc OpenShift project.

Finally, inspect the behavior in OpenShift after the deletion of a synchronized user or a group membership from Red Hat SSO.

### IMPORTANT

Red Hat SSO administration is outside the scope of this course. For more information about Red Hat SSO, refer to the Red Hat Single Sign-On Administration (DO313) course at <https://www.redhat.com/en/services/training/do313-red-hat-single-sign-on-administration>

1. Assign the edit cluster role in OpenShift to the contractors group, so users in that group can modify most of the objects in the auth-oidc project. Assign the view cluster role in OpenShift to the partners group, so users in that group can view most of the objects in the auth-oidc project but cannot make modifications.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

You have access to 70 projects, the list has been suppressed.

You can list all projects with 'oc projects'

Using project "default".

* 1. Change to the auth-oidc project.

[student@workstation ~]$ **oc project auth-oidc**

Now using project "auth-oidc" on server "https://api.ocp4.example.com:6443".

* 1. Assign the edit cluster role in the auth-oidc project to the contractors group.

### NOTE

Ignore the warning message, because OpenShift creates the contractors group after you synchronize the users from Red Hat SSO.

[student@workstation ~]$ **oc adm policy add-role-to-group edit contractors**

Warning: Group 'contractors' not found

clusterrole.rbac.authorization.k8s.io/edit added: "contractors"

* 1. Assign the view cluster role in the auth-oidc project to the partners group.

### NOTE

Ignore the warning message, because OpenShift creates the partners group after you synchronize the users from Red Hat SSO.

[student@workstation ~]$ **oc adm policy add-role-to-group view partners**

Warning: Group 'partners' not found

clusterrole.rbac.authorization.k8s.io/view added: "partners"

1. List the users and groups in the external\_providers realm in Red Hat SSO. List the group membership for the abbyquincy user.
   1. Connect to the Red Hat SSO machine as the rhsso user.
   2. [student@workstation ~]$ **ssh rhsso@sso.ocp4.example.com**

[rhsso@sso ~]$

* 1. Use the kcadm tool to connect to Red Hat SSO.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh config credentials \**

**--server https://sso.ocp4.example.com:8080/auth \**

**--user admin --password redhatocp --realm master**

Logging into https://sso.ocp4.example.com:8080/auth as user admin of realm master

* 1. List the users in the external\_providers realm. The user IDs would differ on your system.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get users \**

**-r external\_providers --fields 'id,username'**

[ {

"id" : "***a175e1b7-6210-40f8-aeda-732021142e84***",

"username" : "abbyquincy"

}, {

"id" : "***958fde0c-360c-48f8-b5e5-942708fbb36e***",

"username" : "fricisritcher"

}, {

"id" : "***64898122-5875-4418-88ac-c9eeeaa0f409***",

"username" : "jayalamont"

} ]

* 1. List the groups in the external\_providers realm. The group IDs would differ on your system.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get groups \**

**-r external\_providers --fields 'id,name'**

[ {

"id" : "***e92319be-d5df-4a0c-833a-687fd25ca34c***",

"name" : "contractors",

}, {

"id" : "***3dcc3053-4ebf-4894-969a-f26d8e2bc22f***",

"name" : "partners",

} ]

* 1. List the groups for the abbyquincy user. Use the ID for the abbyquincy user from an earlier step.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get \**

**users/*a175e1b7-6210-40f8-aeda-732021142e84*/groups -r external\_providers**

[ {

"id" : "***e92319be-d5df-4a0c-833a-687fd25ca34c***",

"name" : "contractors",

"path" : "/contractors"

} ]

1. Retrieve the Red Hat SSO client information and note the client secret.
   1. List the information for the ocp\_rhsso client from Red Hat SSO.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get clients \**

**-r external\_providers -q clientId=ocp\_rhsso**

[ {

"id" : "***f57e9ddc-8c60-4b40-8048-ec0120595be2***",

"clientId" : "ocp\_rhsso",

"surrogateAuthRequired" : false,

"enabled" : true,

"alwaysDisplayInConsole" : false,

"clientAuthenticatorType" : "client-secret",

"redirectUris" : [ "https://oauth-openshift.apps.ocp4.example.com/\*" ],

"webOrigins" : [ "https://oauth-openshift.apps.ocp4.example.com" ],

...output omitted...

* 1. Generate a JSON file with the Red Hat SSO client information, which contains the client ID, the authentication server URL, and the client secret. Use the ocp\_rhsso ID from the previous step. The client secret is necessary for configuring the OAuth Custom Resource (CR) in OpenShift, in a later step.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get \**

**clients/*f57e9ddc-8c60-4b40-8048-ec0120595be2*/installation\**

**/providers/keycloak-oidc-keycloak-json \**

**-r external\_providers > rhsso.json**

|  |  |
| --- | --- |
|  | The /providers/…​ text must come after the …​/installation text in a single line without spaces. |

* 1. View the content of the JSON file, which contains the Red Hat SSO client information. You would need some Red Hat SSO client information when configuring the OIDC IdP on OpenShift.

Note the secret, which you use in a later step. The client secret would differ on your system.

The JSON file also provides the Red Hat SSO client ID, ocp\_rhsso, in the resource parameter. Use this value for the clientID parameter in the IdP configuration file on OpenShift.

The issuer parameter in the IdP configuration on OpenShift concatenates the value from the auth-server-url parameter, the /realms/ string, and the Red Hat SSO realm name, which in this case is external\_providers.

[rhsso@sso ~]$ **cat rhsso.json**

{

"realm" : "external\_providers",

"auth-server-url" : "https://sso.ocp4.example.com:8080/auth/",

"ssl-required" : "external",

"resource" : "ocp\_rhsso",

"credentials" : {

"secret" : "***X4ZTPfDr0b8loqOFArfidhaHq85bHyiy***"

},

"confidential-port" : 0

* 1. Return to the workstation machine.

[rhsso@sso ~]$ **exit**

logout

Connection to sso.ocp4.example.com closed.

[student@workstation ~]$

1. Configure the OpenShift OAuth CR to synchronize users from the Red Hat SSO OIDC client that was configured in the previous step.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. Create the rhsso-oidc-client-secret OpenShift secret for the Red Hat SSO client secret by using the client secret from a previous step.

[student@workstation ~]$ **oc create secret generic rhsso-oidc-client-secret \**

**--from-literal clientSecret=*X4ZTPfDr0b8loqOFArfidhaHq85bHyiy* \**

**-n openshift-config**

secret/rhsso-oidc-client-secret created

* 1. Create the OAuth CR YAML file. You can find an example for the CR in the ~/DO380/labs/auth-oidc/sso\_config.yml file. The YAML file includes an LDAP IdP that you must preserve, because it provides the admin and developer users. Do not remove the LDAP IdP, and add the OIDC IdP for Red Hat SSO.

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- ldap:

...output omitted...

- openID:

claims:

email:

- email

name:

- name

preferredUsername:

- preferred\_username

groups:

- groups

clientID: **ocp\_rhsso**

clientSecret:

name: **rhsso-oidc-client-secret**

extraScopes: []

issuer: >-

https://sso.ocp4.example.com:8080/auth/realms/external\_providers

mappingMethod: claim

name: **RHSSO\_OIDC**

type: OpenID

* 1. Apply the configuration to the OAuth CR.
  2. [student@workstation ~]$ **oc apply -f ~/DO380/labs/auth-oidc/sso\_config.yml**

oauth.config.openshift.io/cluster configured

* 1. Verify the status for the OAuth pods and wait for the OAuth pods to be redeployed. Press **Ctrl**+**C** when done.

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

Every 2.0s: oc get pods -n openshift-authentication workstation: ...

NAME READY STATUS RESTARTS AGE

oauth-openshift-79c7865785-6zxvp 1/1 Running 0 2m10s

oauth-openshift-79c7865785-bbp5w 1/1 Running 0 2m39s

oauth-openshift-79c7865785-jl6th 1/1 Running 0 102s

**^C**

1. Verify that you can log in to the cluster as the abbyquincy user with redhat\_sso as the password, and create resources in the auth-oidc OpenShift project. The ability to create resources derives from the edit cluster role for the contractors group.
   1. Log in to the cluster as the abbyquincy user.

[student@workstation ~]$ **oc login -u abbyquincy -p redhat\_sso**

Login successful.

...output omitted...

Using project "auth-oidc".

* 1. Verify that the user can create a pod.

[student@workstation ~]$ **oc run ubi9-date --restart 'Never' \**

**--image registry.ocp4.example.com:8443/ubi9/ubi -- date**

pod/ubi9-date created

* 1. Verify that the user can view pod information in the project.

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

NAME READY STATUS RESTARTS AGE

ubi9-date 0/1 Completed 0 85s

* 1. Verify that the user cannot review the user information, because the user is not a cluster administrator.

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

Error from server (Forbidden): users.user.openshift.io is forbidden:

User "abbyquincy" cannot list resource "users" in API group "user.openshift.io" at the cluster scope

1. Verify that OpenShift creates the user and group the first time that they log in.
   1. Verify that OpenShift synchronizes the user from Red Hat SSO the first time that they log in, and that the abbyquincy user is a member of the contractors group in OpenShift.

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

...output omitted...

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

NAME UID FULL NAME IDENTITIES

**abbyquincy 11bc3d49-... Abby Quincy RHSSO\_OIDC:a175e1b7-...-732021142e84**

admin 00fed1ea-... Administrator Red Hat Identity Management:dWlk...

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

NAME USERS

Default SMB Group

admins admin

**contractors abbyquincy**

developer

editors

ocpadmins admin

ocpdevs developer

1. Synchronize the jayalamont user to OpenShift, and verify that the user can view objects in the OpenShift project, but cannot edit them.
   1. Log in as the jayalamont user with redhat\_sso as the password.

[student@workstation ~]$ **oc login -u jayalamont -p redhat\_sso**

Login successful.

...output omitted...

Using project "auth-oidc".

* 1. Verify that the user can view pod information in the project.

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

NAME READY STATUS RESTARTS AGE

ubi9-date 0/1 Completed 0 7m47s

* 1. Verify that the user cannot remove the pod in the project.

[student@workstation ~]$ **oc delete pod ubi9-date**

Error from server (Forbidden): pods "ubi9-date" is forbidden: User "jayalamont" cannot delete resource "pods" in API group "" in the namespace "auth-oidc"

1. Log in to OpenShift as the fricisritcher user, which is another user in the contractors group, and which can edit resources in the auth-oidc OpenShift project.
   1. Log in as the fricisritcher user in the OpenShift web console. To do so, open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com. Click **RHSSO\_OIDC**.
   2. Log in as the fricisritcher user with redhat\_sso as the password.
2. Remove the user's membership of the contractors group in Red Hat SSO.
   1. Change to the terminal window. Verify that OpenShift correctly synchronizes the user from Red Hat SSO, and that the user is a member of the contractors group.

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

...output omitted...

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

NAME UID FULL NAME IDENTITIES

abbyquincy 11bc3d49-... Abby Quincy RHSSO\_OIDC:a175e1b7-...-732021142e84

admin 00fed1ea-... Administrator Red Hat Identity Management:dWlk...

**fricisritcher af752e5b-... Fricis Ritcher RHSSO\_OIDC:958fde0c-...-942708fbb36e**

jayalamont e5f74993-... Jaya Lamont RHSSO\_OIDC:64898122-...-c9eeeaa0f409

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

NAME USERS

Default SMB Group

admins admin

**contractors abbyquincy, fricisritcher**

developer

editors

ocpadmins admin

ocpdevs developer

partners jayalamont

* 1. Connect to the Red Hat SSO machine as the rhsso user. Remove the fricisritcher user's membership of the contractors group in Red Hat SSO, by using the user and group ID from a previous step.

[student@workstation ~]$ **ssh rhsso@sso.ocp4.example.com**

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh config credentials \**

**--server https://sso.ocp4.example.com:8080/auth \**

**--user admin --password redhatocp --realm master**

Logging into https://sso.ocp4.example.com:8080/auth as user admin of realm master

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh delete \**

**users/*958fde0c-360c-48f8-b5e5-942708fbb36e*\**

**/groups/*e92319be-d5df-4a0c-833a-687fd25ca34c* \**

**-r external\_providers**

|  |  |
| --- | --- |
|  | The /groups/…​ text must come after the user ID, without spaces. |

* 1. Verify that, even though you remove the fricisritcher user's membership of the contractors group in Red Hat SSO, the user is still a member of the contractors group in OpenShift.

[rhsso@sso ~]$ **exit**

logout

Connection to sso.ocp4.example.com closed.

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

NAME USERS

Default SMB Group

admins admin

**contractors abbyquincy, fricisritcher**

developer

editors

ocpadmins admin

ocpdevs developer

partners jayalamont

1. Verify that the fricisritcher user can still edit resources in the auth-oidc OpenShift project. Verify that the changes in the OpenShift user apply only when the user logs out and then logs back in.
   1. Try to remove the ubi9-date pod as the fricisritcher user. To do so, change to the web browser window, and from the perspective switcher select **Administrator**. Click **Workloads** → **Pods**. Select **auth-oidc** from the project drop-down menu.

In the list with the ubi9-date pod, click the icon with three dots and then click **Delete Pod**.

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

Click **Delete**. The user can still edit objects in OpenShift, because it did not synchronize the group membership for the user.

* 1. Log out and log back in as the fricisritcher user in the OpenShift web console. Click the username in the upper right corner and click **Log out**.

Click **RHSSO\_OIDC** and log in again as the fricisritcher user with redhat\_sso as the password.

* 1. Change to the terminal window. Verify that OpenShift synchronizes the group membership for the user, because the user is not a member of the contractors group.

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

NAME USERS

Default SMB Group

admins admin

contractors abbyquincy

developer

editors

ocpadmins admin

ocpdevs developer

partners jayalamont

1. Remove the fricisritcher user from Red Hat SSO, and verify that the user session is still open in the OpenShift web console until the user logs out. Force the user to log out so they cannot reconnect. Verify that the user account is present as a leftover in OpenShift and requires manual removal.
   1. Change to the web browser window and verify that the session for the fricisritcher user is not expired. If the session expires, then log in again as the fricisritcher user.
   2. Change to the terminal window and remove the fricisritcher user from Red Hat SSO, by using the user ID from an earlier step.

[student@workstation ~]$ **ssh rhsso@sso.ocp4.example.com**

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh config credentials \**

**--server https://sso.ocp4.example.com:8080/auth \**

**--user admin --password redhatocp --realm master**

Logging into https://sso.ocp4.example.com:8080/auth as user admin of realm master

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh delete \**

**users/*958fde0c-360c-48f8-b5e5-942708fbb36e* -r external\_providers**

* 1. Change to the web browser window and try to create a project called fricis-project as the fricisritcher user. To do so, click **Create a new project**.

In the name field, enter fricis-project, and click **Create**. The user can create a project even when you remove them from Red Hat SSO.

* 1. Create a pod in the fricis-project project as the fricisritcher user. To do so, click **Workloads** → **Pods** and then **Create Pod**.
  2. Change the default YAML definition file as follows. Then, click **Create**.

apiVersion: v1

kind: Pod

metadata:

name: example

labels:

name: httpd

namespace: **fricis-project**

spec:

...output omitted...

The user can create a pod even if you remove them from Red Hat SSO.

* 1. Change to the terminal window, and verify that the user has active access tokens in OpenShift.

[rhsso@sso ~]$ **exit**

logout

Connection to sso.ocp4.example.com closed.

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

NAME USER NAME CLIENT NAME CREATED EXPIRES ...

sha256~4v47... admin openshift-... 159m 2023-05-31 ...

...output omitted...

**sha256~WbOZ... fricisritcher** console 8m42s 2023-05-31 ...

sha256~Y52j... admin openshift-... 159m 2023-05-31 ...

...output omitted...

Remove all the access tokens for the fricisritcher user.

[student@workstation ~]$ **oc delete oauthaccesstoken \**

**$(oc get oauthaccesstoken \**

**-o jsonpath='{.items[?(@.userName=="fricisritcher")].metadata.name}')**

oauthaccesstoken.oauth.openshift.io "sha256~WbOZ..." deleted

* 1. Change to the web browser window and wait until OpenShift automatically logs out the user. Verify that you can no longer log in as the fricisritcher user. Close the web browser window.
  2. Change to the terminal window, and verify that the user and the user identity are still present in OpenShift, and require manual removal.

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

NAME UID FULL NAME IDENTITIES

abbyquincy 11bc3d49-... Abby Quincy RHSSO\_OIDC:a175e1b7-...-732021142e84

admin 00fed1ea-... Administrator Red Hat Identity Management:dWlk...

developer 514df291-... Developer User Red Hat Identity Management:dWlk...

**fricisritcher af752e5b-... Fricis Ritcher RHSSO\_OIDC:958fde0c-...-942708fbb36e**

jayalamont e5f74993-... Jaya Lamont RHSSO\_OIDC:64898122-...-c9eeeaa0f409

* 1. Remove the fricisritcher user and identity from OpenShift.

[student@workstation ~]$ **oc delete user fricisritcher**

user.user.openshift.io "fricisritcher" deleted

[student@workstation ~]$ **oc delete identity \**

***RHSSO\_OIDC:958fde0c-360c-48f8-b5e5-942708fbb36e***

identity.user.openshift.io "RHSSO\_OIDC:958fde0c-...-942708fbb36e" deleted

**Finish**

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

[student@workstation ~]$ **lab finish auth-oidc**

## **Guided Exercise: Solve User Sync Conflicts**

Solve conflicts when synchronizing users from more than one identity provider.

**Outcomes**

* Synchronize users and groups from Red Hat SSO to OpenShift.
* Solve conflicts when synchronizing a user from more than one IdP.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

In this exercise, you solve the conflicts that appear when two IdPs provide the same user.

As a use case, the lab script configures Red Hat SSO and htpasswd as OpenShift IdPs.

The htpasswd IdP provides the abbyquincy user with redhat\_htpasswd as the password.

Red Hat SSO provides the abbyquincy user with redhat\_sso as the password in the external\_providers realm.

1. Log in as the abbyquincy user through the htpasswd IdP. The user also exists in Red Hat SSO. Try to synchronize the user from both IdPs.
   1. Log in as the abbyquincy user through the htpasswd IdP.

### NOTE

If you get a 401 login error, then wait for a few moments for the openshift-authentication pods to re-create, or check them by logging in as an administrator and using the oc get pods -n openshift-authentication command.

[student@workstation ~]$ **oc login -u abbyquincy -p redhat\_htpasswd \**

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

Login successful.

...output omitted...

* 1. Verify that OpenShift correctly synchronizes the user from the htpasswd IdP the first time that they log in.

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

...output omitted...

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

NAME UID FULL NAME IDENTITIES

**abbyquincy d43c5583-... htpasswd\_provider:abbyquincy**

admin 00fed1ea-... Administrator Red Hat Identity Management:dWlk...

* 1. Try to log in as the abbyquincy user through Red Hat SSO.
  2. [student@workstation ~]$ **oc login -u abbyquincy -p redhat\_sso**

Error from server (InternalError): Internal error occurred: unexpected response: 500

* 1. Review the logs for the authentication pods to find the login error. OpenShift cannot synchronize the user from Red Hat SSO, because the identity was synchronized from the htpasswd IdP, and the mappingMethod parameter for both IdPs is set to claim. The pod names might differ on your system.

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

NAME READY STATUS RESTARTS AGE

oauth-openshift-69df8585dc-rhh7j 1/1 Running 0 4m46s

oauth-openshift-69df8585dc-rwvzb 1/1 Running 0 5m42s

oauth-openshift-69df8585dc-zlkzd 1/1 Running 0 5m14s

[student@workstation ~]$ **oc logs *oauth-openshift-69df8585dc-zlkzd* \**

**-n openshift-authentication**

...output omitted...

… Error authenticating login "abbyquincy" with provider "RHSSO\_OIDC": **user "abbyquincy" cannot be claimed by identity "RHSSO\_OIDC:a175e1b7-6210-40f8-aeda-732021142e84" because it is already mapped to [htpasswd\_provider:abbyquincy]**

… AuthenticationError: user "abbyquincy" cannot be claimed by identity "RHSSO\_OIDC:a175e1b7-6210-40f8-aeda-732021142e84" because it is already mapped to [htpasswd\_provider:abbyquincy]

* 1. To log in by using the OIDC Red Hat SSO, delete the abbyquincy user and its identity.

[student@workstation ~]$ **oc get user abbyquincy**

NAME UID FULL NAME IDENTITIES

**abbyquincy** e9538f53-... **htpasswd\_provider:abbyquincy**

[student@workstation ~]$ **oc get identity htpasswd\_provider:abbyquincy**

NAME IDP NAME IDP USER NAME USER NAME

**htpasswd\_provider:abbyquincy** htpasswd\_provider abbyquincy abbyquincy

[student@workstation ~]$ **oc delete user abbyquincy**

user.user.openshift.io "abbyquincy" deleted

[student@workstation ~]$ **oc delete identity htpasswd\_provider:abbyquincy**

identity.user.openshift.io "htpasswd\_provider:abbyquincy" deleted

* 1. Log in as the abbyquincy user through Red Hat SSO.

[student@workstation ~]$ **oc login -u abbyquincy -p redhat\_sso**

Login successful.

...output omitted...

* 1. Verify that OpenShift correctly synchronizes the user from Red Hat SSO.

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

...output omitted...

[student@workstation ~]$ **oc get user abbyquincy**

NAME UID FULL NAME IDENTITIES

abbyquincy 54f7... Abby Quincy **RHSSO\_OIDC:a175...**

1. To avoid conflicts when two IdPs provide the same user, change the OIDC Red Hat SSO and htpasswd mappingMethod parameter from claim to add, so OpenShift adds the identity from Red Hat SSO and htpasswd to the user no matter which one is used first.
   1. Change the OIDC Red Hat SSO mappingMethod parameter from claim to add.

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

...output omitted...

**- htpasswd:**

fileData:

name: htpasswd-secret

**mappingMethod**: **add**

name: htpasswd\_provider

type: HTPasswd

**- openID:**

...output omitted...

**mappingMethod**: **add**

name: RHSSO\_OIDC

type: OpenID

### NOTE

You can find the YAML file for the OAuth CR in the ~/DO380/labs/auth-conflict/oauth\_config.yml file.

* 1. Apply the OAuth CR YAML file to the cluster.

[student@workstation ~]$ **oc apply -f ~/DO380/labs/auth-conflict/oauth\_config.yml**

oauth.config.openshift.io/cluster configured

* 1. Wait for the OAuth pods to redeploy.

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

Every 2.0s: oc get pods -n openshift-authentication workstation: ...

NAME READY STATUS RESTARTS AGE

oauth-openshift-f94c8d5fd-8w9nb 1/1 Running 0 92s

oauth-openshift-f94c8d5fd-jnt8r 1/1 Running 0 63s

oauth-openshift-f94c8d5fd-vk79l 1/1 Running 0 35s

**^C**

* 1. Try to log in again as the abbyquincy user through the htpasswd IdP.

[student@workstation ~]$ **oc login -u abbyquincy -p redhat\_htpasswd**

Login successful.

...output omitted...

* 1. Verify that OpenShift correctly synchronizes the user from the htpasswd IdP. Verify that the abbyquincy user has two identities, one from each of the two IdPs.

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

...output omitted...

[student@workstation ~]$ **oc get user abbyquincy**

NAME UID FULL NAME IDENTITIES

**abbyquincy** 54f7... Abby Quincy **RHSSO\_OIDC:a175...,**

**htpasswd\_provider:abbyquincy**

**Finish**

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

[student@workstation ~]$ **lab finish auth-conflict**

## **Token and Client Certificate Authentication with kubeconfig Files**

### **Objectives**

* Generate a token and a client certificate and add them to a kubeconfig file.

### **External Client Authentication**

In certain scenarios, you might authenticate external clients, such as CI/CD pipelines or monitoring tools, to the Kubernetes cluster. Kubernetes enables external clients to authenticate to the Kubernetes API by embedding either a client certificate or an authentication token into a kubeconfig configuration file. This feature ensures that only authorized external clients can access the Kubernetes cluster.

### **Service Accounts**

A service account (SA) is a Kubernetes resource that provides an identity for processes that run in a pod. SA tokens authenticate the interactions between components within the OpenShift cluster, as well as with external resources. OpenShift uses SA tokens to grant permissions to pods and other resources to interact with the Kubernetes API server. External services use SA tokens to access other OpenShift resources, or to access the Kubernetes API. With SAs, you can control API access without the need to borrow a regular user's credentials. SAs are specific to a particular project and cannot be directly shared across projects.

For example, you can use SAs in the following scenarios:

* A replication controller makes API calls to create or delete pods.
* A pod that collects logs might need access to certain log storage resources.
* A backup and restore tool might require access to the cluster's configuration and data to back up and restore data.
* An external application makes monitoring or integration API calls.

#### **Service Account Tokens**

When you create an SA in OpenShift, the SA automatically contains two secrets: an API token and credentials for the OpenShift Container Registry. The automatically generated API token and credentials never expire. However, you can revoke the token and credentials by deleting the secrets, because OpenShift automatically generates a new secret.

Starting with Kubernetes 1.23, SAs do not automatically create the secrets that contain long-term credentials for accessing the Kubernetes API. Thus, you must obtain the credentials by using the TokenRequest API. The tokens that the TokenRequest API provides are more secure than the tokens that are stored in secrets, because the TokenRequest tokens have a bounded lifetime; other API clients cannot read the TokenRequest tokens; and OpenShift automatically invalidates the TokenRequest tokens when the pod that they are mounted into is deleted.

Although Red Hat OpenShift Container Platform (RHOCP) 4.14 is based on Kubernetes 1.27, it still creates the SA token secrets to communicate with the Kubernetes API server, because some features and workloads need the SA secrets. However, this behavior will change in a future release. Thus, although you can use the automatically generated token secret to authenticate the SA to the Kubernetes API server, Red Hat recommends using the TokenRequest API to generate tokens that are bound to the SA, and not relying on the automatically generated token secret.

### NOTE

You can still manually create long-lived API tokens for SAs. However, Red Hat recommends using long-lived API tokens only if you cannot use the TokenRequest API and if the security exposure of a non-expiring token is acceptable to you. To manually create long-lived API tokens for SAs, refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-secrets-creating-sa_nodes-pods-secrets>

The following diagram summarizes the authentication flow by using the TokenRequest API:

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | An application that is running in a pod needs to authenticate to the Kubernetes API. The kubelet agent requests a bound SA account token from the TokenRequest API. |
|  | The kubelet agent receives the SA token from the TokenRequest API and mounts it into the pod. |
|  | The application reads the SA token and uses it to authenticate to the Kubernetes API. |

For external applications, you can manually generate a bound SA token through the TokenRequest API, and use it as the bearer token for authentication to the Kubernetes API. Generating a bound SA token in OpenShift by using the TokenRequest API does not remove or override the automatically generated token secret.

After creating the SA, use the following command to generate a bound SA token by using the TokenRequest API:

[user@host ~]# **oc create token *SA\_name* -n *project***

You can set the lifetime for the SA token by using the --duration option. The default lifetime for the SA token is one hour. You must manually refresh the manually generated SA token before it expires, so that your application can continue authenticating to the Kubernetes API.

The following example shows a decoded JWT SA token:

HEADER

...output omitted...

PAYLOAD:DATA

{

"aud": [

"https://kubernetes.default.svc"

],

"exp": 1698059299,

"iat": 1698055699,

"iss": "https://kubernetes.default.svc",

"kubernetes.io": {

"namespace": "my-project",

"serviceaccount": {

"name": "my-sa",

"uid": "3acb4630-babe-4e85-996d-cbc80f62146f"

}

},

"nbf": 1698055699,

"sub": "system:serviceaccount:my-project:my-sa"

}

SIGNATURE

...output omitted...

|  |  |
| --- | --- |
|  | SA token expiration date in Unix epoch format |
|  | SA token issued date in Unix epoch format |
|  | Project for the SA token |
|  | Name for the SA |

OpenShift assigns an SA to every pod that you deploy. By default, OpenShift creates the following SAs for every project:

**builder**

OpenShift uses this SA to build pods. By default, this SA has the system:image-builder role, so the resource can push images to any image stream in the project by using the internal Docker registry.

**deployer**

OpenShift uses this SA in deployment pods. By default, this SA has the system:deployer role, so the resource can view and modify replication controllers and pods in the project. This SA exists only for applications that use OpenShift deployment configuration resources.

**default**

OpenShift assigns this default SA to pods if you do not specify a different SA when you create the pods.

#### **Manage an SA**

You can manage SAs by using the typical oc commands, such as create, get, or describe. You can also grant roles to SAs the same as for regular users.

If you grant roles to an SA, then you must use the name of the SA together with the name of the project and the system:serviceaccount string, according to the following syntax:

system:serviceaccount:***project***:***name***

You can also use the -z option to avoid using the long system:serviceaccount:*project*:*sa\_name* name, and use instead the short *sa\_name* name.

Whenever you create an SA, it is automatically a member of the following groups:

**system:serviceaccounts**

This group includes all the SAs in the cluster.

**system:serviceaccounts:*project***

This group includes all the SAs in the specified project.

### **Client Certificate Authentication**

Client certificate authentication in Kubernetes clusters refers to the process of authenticating clients, such as users or services, which access the Kubernetes cluster by using TLS client certificates.

By default, OpenShift provides an internal certificate authority (CA). The OpenShift internal CA is a built-in component of the OpenShift cluster that manages and issues digital certificates in the cluster. The internal CA provides a trusted source for generating X.509 certificates for secure communication, authentication, and encryption in the OpenShift cluster.

The Kubernetes API server requires client authentication by using client certificates. OpenShift is preconfigured to trust the client certificates that the OpenShift internal CA signs.

### NOTE

You can also configure additional client CAs for the Kubernetes API server. For more information about configuring additional client CAs for the Kubernetes API server, refer to <https://access.redhat.com/solutions/6054271>

This feature is mainly used to generate a client certificate for an administrator user, such as the predefined system:admin user, to use this client certificate as a backdoor for cluster administrators in the event of failure of the IdP that provides the administrator credentials.

Although Red Hat does not recommend doing so, you can also use client certificates in certain scenarios when running outside the cluster, such as CI/CD pipelines, automation playbooks, or monitoring tools. These clients need to present valid client certificates during the TLS handshake to establish a secure connection with the API server. This mechanism ensures that only authorized clients can access the API server from outside the cluster.

### WARNING

Red Hat recommends using SAs to run automation outside the cluster whenever you can, instead of using client certificates, because a certificate cannot be revoked in OpenShift. Revoking a client certificate would require invalidating all client certificates that the current CA ever signed, and creating replacement certificates for all client users and applications. For more information about this topic, refer to <https://github.com/kubernetes/kubernetes/issues/18982>

#### **Create a Client Certificate**

OpenShift assigns the username and the groups for the user by using the common name (CN) and the organization (O) fields from the certificate, respectively. You can use role-based access control (RBAC) rules to provide the minimal rights to the client account to perform the job. For example, you could provide view permissions for the entire cluster to a monitoring tool, or edit permissions on selected projects to a CI/CD pipeline.

To create client certificates by using the internal OpenShift CA, you must follow these steps:

1. Create a certificate signing request (CSR): The first step is to create a CSR for the client. You can use the OpenSSL tool to create the CSR. This request includes the client's information and the public key. You can use more than one group for the user if you require it.
2. [user@host ~]$ **openssl req -nodes -newkey rsa:4096 -keyout *key\_filename* \**

**-subj "/O=*group1*/O=*group2*/CN=*username*" -out *csr\_filename***

1. Create the CSR YAML resource manifest. The following example shows the parameters for a CSR:
2. apiVersion: certificates.k8s.io/v1
3. kind: CertificateSigningRequest
4. metadata:
5. name: ***csr\_name***
6. spec:
7. signerName: kubernetes.io/kube-apiserver-client
8. expirationSeconds: 604800 # one week
9. request: $(base64 -w0 ***csr\_filename***)
10. usages:

- client auth

|  |  |
| --- | --- |
|  | The CSR name. |
|  | OpenShift provides the kube-apiserver-client built-in signer of certificates that the API server accepts. OpenShift provides different built-in signers that you can use to sign certificates. For more information, refer to <https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#kubernetes-signers> |
|  | Expiration time for the certificate in seconds. If you do not specify a value, then the default value is 30 days. |
|  | Contains the OpenSSL X.509 CSR, which is encoded in Base64. |
|  | Specifies the use cases for the client certificate. It must include the client auth usage, which indicates that the certificate is intended for client authentication. |

1. Submit the CSR to the Kubernetes API server. The API server interacts with the internal CA to process the CSR.

[user@host ~]$ **oc apply -f csr.yaml**

1. Review, approve, and sign the CSR: You can review the CSR details by using the oc describe csr *csr\_name* command. After reviewing the CSR details, use the following command so the internal CA signs the CSR to generate the client certificate:

[user@host ~]$ **oc adm certificate approve *csr\_name***

1. Retrieve the client certificate: After you approve the CSR and the OpenShift internal CA generates the certificate, you can retrieve the signed certificate from the API server.
2. [user@host ~]$ **oc get csr *csr\_name* -o jsonpath='{.status.certificate}' \**

**| base64 -d > *certificate\_filename***

1. Distribute the certificate: The signed certificate, together with the private key that generates the CSR, enable the client to authenticate to the cluster.

### **OpenShift CLI Configuration Files**

You can use a kubeconfig file as a command-line interface (CLI) configuration file to set up profiles for use with the Kubernetes kubectl and OpenShift oc CLI tools. Moreover, most Kubernetes client libraries use kubeconfig files in the same way as the kubectl and oc CLI tools. Use kubeconfig files to authenticate external applications to the cluster by storing tokens and client certificates inside the kubeconfig files.

The kubeconfig file is defined as a YAML file that contains clusters, users, and contexts.

**clusters**

The clusters parameter in the kubeconfig file contains information about the OpenShift clusters, such as the IP or fully qualified domain name (FQDN), or the CA.

**users**

The users parameter contains the user credentials to interact with the Kubernetes API. This parameter contains information such as the username, the user password, or the user token.

**contexts**

The contexts parameter contains information about the combination of a cluster and a user to interact with the Kubernetes API. Whenever you run an oc command, you reference a context inside the kubeconfig file.

#### **kubeconfig File Details**

When you run any oc command, OpenShift reads a kubeconfig file in the following ways in turn:

1. The specified file in the --kubeconfig option, if you use it
2. The specified file in the KUBECONFIG environment variable, if it is set
3. The default kubeconfig file in ~/.kube/config

When you log in to the OpenShift cluster through the oc login command for the first time, OpenShift creates a kubeconfig file in the default location at ~/.kube/config, if the file does not exist. You can add more authentication and connection details to the kubeconfig file automatically by using the oc login and oc project commands, or by manually editing the kubeconfig file.

You can read the details from your kubeconfig file by using the oc config view command or by opening the file with a text editor. The following example shows the parameters for a kubeconfig file from an OpenShift cluster:

apiVersion: v1

clusters:

- cluster:

server: https://api.ocp4.prod.com:6443

name: production

- cluster:

server: https://api.ocp4.stage.com:6443

certificate-authority: ocp-apiserver-cert.crt

name: stage

users:

- name: admin-production

user:

token: REDACTED

- name: admin-stage

user:

client-certificate: admin-stage.crt

client-key: tls.key

contexts:

- context:

cluster: production

namespace: prod-app

user: admin-production

name: prod-app/api-ocp4-prod-com:6443/admin-production

- context:

cluster: stage

namespace: demo-app

user: admin-stage

name: demo-app/api-ocp4-stage-com:6443/admin-stage

current-context: demo-app/api-ocp4-stage-com:6443/admin-stage

kind: Config

preferences: {}

|  |  |
| --- | --- |
|  | The list of all the clusters that you already connected to. The example defines two clusters, production and stage, with their FQDNs that are defined in the server parameter. The stage cluster definition also contains the public certificate from the API server in the certificate-authority parameter. |
|  | The list of all the users that you already connected to the cluster. The example defines two users, admin-production and admin-stage. The admin-production user definition uses the token that is defined in the token parameter to authenticate to the API server. OpenShift truncates the token information, to prevent the configuration file from becoming too long. You can use the --raw option in the oc config view command to show the token information. The admin-stage user authenticates to the cluster by using a client certificate that the internal OpenShift CA previously signed, as defined in the client-certificate parameter, and the key that signed the client certificate, as defined in the client-key parameter. |
|  | The list of contexts that you can reference when using the oc command. Every context specifies a combination of a user and a cluster that are already defined in the kubeconfig file, through the cluster and user parameters, and a project through the namespace parameter. |

#### **Configure CLI Profiles**

Although you can set most kubeconfig file parameters by using the oc login and oc project OpenShift commands, you can use the oc config command to manually configure these parameters if needed, instead of directly modifying the file. You can use the oc config command with the --kubeconfig=*file* option to create and store kubeconfig files with different parameters that you can use when required. The oc config command comes from the Kubernetes kubectl CLI tool, with no modifications from OpenShift.

Use the following oc config subcommands to manually configure your kubeconfig files:

**set-cluster**

Creates a cluster entry in the kubeconfig file. This subcommand accepts different cluster parameters as the server IP or the CA file.

**set-credentials**

Creates a user entry in the kubeconfig file. This subcommand accepts different user parameters as the username, the user password, or the token.

**set-context**

Creates a context entry in the kubeconfig file. Use this subcommand to define a context to specify a combination of a user and a cluster. You can also set the OpenShift project.

**use-context**

Sets the current context.

### NOTE

For more information about manually configuring the kubeconfig files by using the oc config command, refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#manual-configuration-of-cli-profiles_managing-cli-profiles>

### **User Impersonation**

User impersonation in OpenShift enables certain users or SAs to act on behalf of other users or SAs with different permissions and roles. This feature is useful when administrators or privileged users need to perform actions on behalf of regular users, or when SAs need to execute tasks on behalf of other SAs, or for regular users to perform actions with administrative permissions. For example, a system administrator can debug an RBAC rule by impersonating another user and verifying whether OpenShift accepts or denies a request. As another example, you want system administrators to escalate privileges when it is necessary instead of them logging in as cluster administrators.

#### **System Administrator That Impersonates a Regular User**

As a system administrator, use the --as and --as-group options when using the oc command to impersonate a user or group. Use the oc auth can-i command to test the user access to a particular resource in the cluster. Use the -n option to specify a project for the request, or use the -A option to verify the action in all the projects. Thus, use the following command to test the RBAC rules for a particular user or group by impersonating them:

[user@host ~]$ **oc auth can-i *command* --as *user\_to\_impersonate* \**

**--as-group *group\_to\_impersonate***

You can also list all the permissions for a specific user or group by using the oc auth can-i --list command.

#### **Regular User That Impersonates Another User**

To grant a regular user permissions to impersonate another user, you must create a custom role with the appropriate permissions, and then create a role binding to assign the role to the user.

For example, to allow a regular user to run commands as an administrator, you can create the following cluster role, which enables anyone to impersonate the administrator user:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: sudo-admin

rules:

- apiGroups: [""]

resources: ["users"]

verbs: ["impersonate"]

resourceNames: ["admin"]

|  |  |
| --- | --- |
|  | The core API group is identified with an empty string. |

Then, bind the role to the user:

[user@host ~]$ **oc create clusterrolebinding *binding\_name* \**

**--clusterrole sudo-admin --user regularuser\_\_**

The user can impersonate the admin user by using the --as=admin option. The following example shows how the user cannot retrieve the node information when using their account, but can retrieve that information when impersonating the admin user:

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

Error from server (Forbidden): nodes is forbidden:

User "regularuser" cannot list resource "nodes" in API group "" at the cluster scope

[user@host ~]$ **oc get nodes --as admin**

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 62d v1.25.7+eab9cc9

worker01 Ready worker 57d v1.25.7+eab9cc9

### REFERENCES

For more information about using SAs for applications on OpenShift, refer to the Using Service Accounts in Applications section in the Red Hat OpenShift Container Platform 4.14 Authentication and authorization documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#service-accounts-overview_using-service-accounts>

For more information about how to configure CLI profiles on OpenShift, refer to the Managing CLI Profiles section in the Red Hat OpenShift Container Platform 4.14 CLI Tools documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#about-switches-between-cli-profiles_managing-cli-profiles>

[Kubernetes: Certificates and Certificate Signing Requests](https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/)

[Kubernetes: Authenticating](https://kubernetes.io/docs/reference/access-authn-authz/authentication/#user-impersonation)

### **Guided Exercise: Token and Client Certificate Authentication with kubeconfig Files**

Generate a token and a client certificate and add them to a kubeconfig file.

**Outcomes**

* Generate a service account (SA) token for authenticating a script that runs outside the OpenShift cluster.
* Generate a client certificate for a system administrator as a backdoor account for the OpenShift cluster.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ lab start auth-tls

**Instructions**

Your company provides you a Bash script to verify the health of an OpenShift cluster. Because this script requires authentication to the cluster, you must generate an SA token to authenticate it. Use the TokenRequest API to generate the SA token. Do not use the automatically generated token secret. The username for the SA must be health-robot. The SA token must be valid for one week. The script verifies the health status for all the pods in the OpenShift cluster, and creates the /tmp/cluster.log file if a pod is in a pending, failed, or unknown state. Thus, because you need to read the pod information for the whole cluster, you must assign the cluster-reader cluster role to the SA. You can find the Bash script in the ~/DO380/labs/auth-tls/ocp\_health.sh file. Create a RHEL cron job to run the script every minute.

Your company also needs a backdoor administrator account to use if the IdP that provides the administrator account fails. As a solution, you decide to create a client certificate. The username for the administrator certificate must be admin-backdoor. The admin-backdoor account must be part of the backdoor-administrators group, and you must assign the cluster-admin cluster role to that group to have administrator permissions for the cluster. The expiration for the administrator certificate must be one week.

1. Create the health-robot SA in the auth-tls OpenShift project and assign it the cluster-reader role.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

* 1. Login successful.
  2. You have access to 70 projects, the list has been suppressed. You can list all
  3. projects with 'oc projects'
  4. Using project "default".

Welcome! See 'oc help' to get started.

* 1. Change to the auth-tls project.

[student@workstation ~]$ oc project auth-tls

Now using project "auth-tls" on server "https://api.ocp4.example.com:6443".

* 1. Create the health-robot SA in the auth-tls project.

[student@workstation ~]$ oc create sa health-robot

serviceaccount/health-robot created

* 1. Assign the cluster-reader cluster role to the health-robot SA so the SA can retrieve information from most of the objects in the cluster.

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

cluster-reader system:serviceaccount:auth-tls:health-robot

clusterrole.rbac.authorization.k8s.io/cluster-reader added: "system:serviceaccount:auth-tls:health-robot"

1. Create the health-robot SA token and store it in the HEALTHROBOT\_TOKEN variable. Create a kubeconfig file with the health-robot user credentials. Store the kubeconfig file in the ~/DO380/labs/auth-tls/robot-cert/health-robot.config file.
   1. Generate the health-robot SA account token and store it in the HEALTHROBOT\_TOKEN variable. The SA token must be valid for one week, which is 604800 seconds.

[student@workstation ~]$ HEALTHROBOT\_TOKEN=$(oc create token -n auth-tls \

health-robot --duration 604800s)

* 1. Change to the ~/DO380/labs/auth-tls directory.

[student@workstation ~]$ cd DO380/labs/auth-tls

* 1. Add the health-robot user credentials to the kubeconfig file.

[student@workstation auth-tls]$ oc config set-credentials health-robot \

--token $HEALTHROBOT\_TOKEN --kubeconfig robot-cert/health-robot.config

User "health-robot" set.

* 1. Set the cluster options in the kubeconfig file.

[student@workstation auth-tls]$ oc config set-cluster \

$(oc config view -o jsonpath='{.clusters[0].name}') \

--server https://api.ocp4.example.com:6443 \

--kubeconfig robot-cert/health-robot.config

Cluster "api-ocp4-example-com:6443" set.

* 1. Set the context for the health-robot user.

[student@workstation auth-tls]$ oc config set-context health-robot \

--cluster $(oc config view -o jsonpath='{.clusters[0].name}') \

--namespace auth-tls --user health-robot \

--kubeconfig robot-cert/health-robot.config

Context "health-robot" created.

* 1. Use the context for the health-robot user.

[student@workstation auth-tls]$ oc config use-context health-robot \

--kubeconfig robot-cert/health-robot.config

Switched to context "health-robot".

1. Verify the permissions for the health-robot SA. Use the health-robot SA token through the kubeconfig file and user impersonation.
   1. Verify the identity for the health-robot user.

[student@workstation auth-tls]$ oc whoami \

--kubeconfig robot-cert/health-robot.config

system:serviceaccount:auth-tls:health-robot

* 1. Verify that the health-robot SA can retrieve information from the users in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i get users -A \

--as system:serviceaccount:auth-tls:health-robot

yes

* 1. Verify that the health-robot SA cannot create projects in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i create project -A \

--as system:serviceaccount:auth-tls:health-robot

no

* 1. Verify that the health-robot SA can retrieve information from the pods in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i get pods -A \

--as system:serviceaccount:auth-tls:health-robot

yes

* 1. Run the cluster-health.sh script to verify the status of the pods in the OpenShift cluster. The script uses the credentials that are stored in the ~/DO380/labs/auth-tls/robot-cert/health-robot.config file.

[student@workstation auth-tls]$ sh cluster-health.sh

Connected to the cluster as the 'system:serviceaccount:auth-tls:health-robot' user

✔ OpenShift is reacheable and up, at version: '4.14.12'

✔ All pods are either running or succeeded.

1. Create a local RHEL CRON job to execute the cluster-health.sh script every minute. Create a failing pod, and verify that the script creates a log file with the projects that contain pods in a pending, failed, or unknown state.
   1. Open the crontab file for the student with the default text editor.

[student@workstation auth-tls]$ crontab -e

* 1. Insert the following line to define the CRON job.

\* \* \* \* \* ~/DO380/labs/auth-tls/cluster-health.sh

* 1. Press **Esc** and type **:wq** to save the changes and exit the editor. When the editor exits, you should see the following output:
  2. crontab: installing new crontab

[student@workstation auth-tls]$

* 1. Create a failing pod in the auth-tls project.

[student@workstation auth-tls]$ oc run failing-pod \

--image registry.ocp4.example.com:8443/failing-pod

...output omitted...

pod/failing-pod created

* 1. Wait for a few minutes and verify the log file. The log file contains the date and the projects with pods in a pending, failed, or unknown state.

[student@workstation auth-tls]$ cat /tmp/cluster.log

...output omitted...

Tue Jul 11 10:29:04 EDT 2023

✘ Namespaces with pending pods:

✘ auth-tls

1. Create a client certificate for a backdoor administrator account to use if the IdP that provides the administrator account fails. The username for the administrator certificate must be admin-backdoor, and the user must be part of the backdoor-administrators group with the cluster-admin cluster role. Set the expiration time for the certificate to one week.
   1. Create the backdoor-administrators group in the OpenShift cluster.

[student@workstation auth-tls]$ oc adm groups new backdoor-administrators

group.user.openshift.io/backdoor-administrators created

* 1. Assign the cluster-admin cluster role to the backdoor-administrators group so users in that group can act as cluster administrators.

[student@workstation auth-tls]$ oc adm policy add-cluster-role-to-group \

cluster-admin backdoor-administrators

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "backdoor-administrators"

* 1. Create the admin-cert directory.

[student@workstation auth-tls]$ mkdir admin-cert

* 1. Create an OpenSSL certificate signing request (CSR) for the admin-backdoor username. The user must be part of the backdoor-administrators group.

[student@workstation auth-tls]$ openssl req -newkey rsa:4096 -nodes \

-keyout tls.key -subj "/O=backdoor-administrators/CN=admin-backdoor" \

-out admin-cert/admin-backdoor-auth.csr

Generating a RSA private key

.................++++

...............................................++++

writing new private key to 'tls.key'

-----

* 1. Define a signing request resource in a YAML file called admin-backdoor-csr.yaml inside the admin-cert directory. Set the expiration time for the certificate to one week. The name for the CSR must be admin-backdoor-access.

[student@workstation auth-tls]$ cat << EOF >> admin-cert/admin-backdoor-csr.yaml

apiVersion: certificates.k8s.io/v1

kind: CertificateSigningRequest

metadata:

name: admin-backdoor-access

spec:

signerName: kubernetes.io/kube-apiserver-client

expirationSeconds: 604800 # one week

request: $(base64 -w0 admin-cert/admin-backdoor-auth.csr)

usages:

- client auth

EOF

* 1. Create the CSR for the admin-backdoor user in the OpenShift cluster.

[student@workstation auth-tls]$ oc create -f admin-cert/admin-backdoor-csr.yaml

certificatesigningrequest.certificates.k8s.io/admin-backdoor-access created

* 1. Verify the CSR status for the admin-backdoor user in the OpenShift cluster.
  2. [student@workstation auth-tls]$ **oc get csr admin-backdoor-access**
  3. NAME AGE SIGNERNAME REQUESTOR REQUESTEDDURATION CONDITION

**admin-backdoor-access** 3m28s kuberne... admin 7d **Pending**

* 1. Approve the CSR in the OpenShift cluster.

[student@workstation auth-tls]$ oc adm certificate approve admin-backdoor-access

certificatesigningrequest.certificates.k8s.io/admin-backdoor-access approved

Verify again the status for the CSR for the admin-backdoor user in the OpenShift cluster.

[student@workstation auth-tls]$ oc get csr admin-backdoor-access

NAME AGE SIGNERNAME REQUESTOR REQUESTEDDURATION CONDITION

admin-backdoor-access 4m23s kuberne... admin 7d Approved,Issued

1. Create a kubeconfig file with the admin-backdoor user credentials. Store the kubeconfig file in the ~/DO380/labs/auth-tls/admin-cert/admin-backdoor.config file.
   1. Extract the signed client certificate for the admin-backdoor user. Store the signed client certificate in the ~/DO380/labs/auth-tls/admin-cert/admin-backdoor-access.crt file.

[student@workstation auth-tls]$ oc get csr admin-backdoor-access \

-o jsonpath='{.status.certificate}' \

| base64 -d > admin-cert/admin-backdoor-access.crt

* 1. Add the admin-backdoor user credentials to the kubeconfig file.

[student@workstation auth-tls]$ oc config set-credentials admin-backdoor \

--client-certificate admin-cert/admin-backdoor-access.crt --client-key tls.key \

--embed-certs --kubeconfig admin-cert/admin-backdoor.config

User "admin-backdoor" set.

* 1. Get the public certificate from the API server.

[student@workstation auth-tls]$ openssl s\_client -showcerts \

-connect api.ocp4.example.com:6443 </dev/null 2>/dev/null|openssl x509 \

**-outform PEM > ocp-apiserver-cert.crt**

* 1. Set the cluster options in the kubeconfig file.

[student@workstation auth-tls]$ oc config set-cluster \

$(oc config view -o jsonpath='{.clusters[0].name}') \

--certificate-authority ocp-apiserver-cert.crt --embed-certs=true \

--server https://api.ocp4.example.com:6443 \

--kubeconfig admin-cert/admin-backdoor.config

Cluster "api-ocp4-example-com:6443" set.

* 1. Set the context for the admin-backdoor user.

[student@workstation auth-tls]$ oc config set-context admin-backdoor \

--cluster $(oc config view -o jsonpath='{.clusters[0].name}') \

--namespace default --user admin-backdoor \

--kubeconfig admin-cert/admin-backdoor.config

Context "admin-backdoor" created.

* 1. Use the context for the admin-backdoor user.

[student@workstation auth-tls]$ oc config use-context admin-backdoor \

--kubeconfig admin-cert/admin-backdoor.config

Switched to context "admin-backdoor".

1. Verify the permissions for the admin-backdoor user. Use the admin-backdoor user client certificate through the kubeconfig file and user impersonation.
   1. Verify the identity for the admin-backdoor user.

[student@workstation auth-tls]$ oc whoami \

--kubeconfig admin-cert/admin-backdoor.config

admin-backdoor

* 1. Verify that the admin-backdoor user can retrieve information from the users in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i get users -A \

--as admin-backdoor --as-group backdoor-administrators

yes

* 1. Verify that the admin-backdoor user can create users in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i create users -A \

--as admin-backdoor --as-group backdoor-administrators

yes

* 1. Verify that the admin-backdoor user can create projects in the OpenShift cluster.

[student@workstation auth-tls]$ oc auth can-i create project -A \

--as admin-backdoor --as-group backdoor-administrators

yes

* 1. Change to the student HOME directory.

[student@workstation auth-tls]$ cd

Finish

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

[student@workstation ~]$ lab finish auth-tls

## **Lab: Authentication and Identity Management**

Define the concepts and custom resources of the OpenShift OAuth server, and explain how these resources augment Kubernetes authentication.

Configure an LDAP identity provider and automate group synchronization between OpenShift OAuth and an LDAP server.

Configure an OIDC identity provider and automate group synchronization between OpenShift OAuth and an OIDC server.

Generate a token and a client certificate and add them to a kubeconfig file.

**Outcomes**

* Use a client certificate for a system administrator account to recover access to the OpenShift cluster.
* Configure RHDS as an LDAP identity provider (IdP) for OpenShift.
* Configure Red Hat SSO as an OIDC IdP for OpenShift.
* Synchronize users and groups from the LDAP and Red Hat SSO IdPs to OpenShift.
* Generate a service account (SA) token for authenticating a script that runs from outside the OpenShift cluster.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

A cluster administrator in your company erroneously removed all the configured IdPs from the OAuth server. You must use a backdoor administrator certificate to re-create the IdPs. For this purpose, create a kubeconfig file that contains the client certificate, the certificate key, and the OpenShift CA server certificate. The certificate is already approved in the OpenShift CA server with administrator privileges.

After you create the administrator kubeconfig file, restore the LDAP IdP that contains one of the company managers, Payden Tomcheck, with the paydentomcheck username and the redhat123 password. You must configure the LDAP automated group synchronization, because the user is part of the administrators group. This group must have administrative privileges to the cluster.

Then, configure an OIDC IdP. This IdP provides the consultant user, Lauren Chan, with the laurenchan username and the redhat\_sso password. You must configure the OIDC IdP to automatically synchronize the groups, because the user is part of the consultants group. This group must have read privileges to the auth-review project.

Finally, configure an external monitoring app, which runs outside the cluster and uses an SA account token that must have read access to the cluster. You must create the SA account in the auth-review project with the audit-bot name.

1. Create a kubeconfig file that contains the client certificate, the certificate key, and the OpenShift CA server certificate.

You must create the kubeconfig file in the ~/DO380/labs/auth-review/certificate directory with the admin.config name.

You can find the admin-access.crt client certificate, the tls.key certificate key, and the ocp-apiserver-cert.crt CA server certificate in the same directory. Use api-ocp4-example-com:6443 as the cluster name and https://api.ocp4.example.com:6443 as the cluster IP in the kubeconfig file.

* 1. Change to the ~/DO380/labs/auth-review/certificate directory.
  2. [student@workstation ~]$ **cd ~/DO380/labs/auth-review/certificate**

[student@workstation certificate]$

* 1. Add the admin user credentials to the kubeconfig file.

[student@workstation certificate]$ **oc config set-credentials admin \**

**--client-certificate admin-access.crt --client-key tls.key \**

**--embed-certs --kubeconfig admin.config**

User "admin" set.

* 1. Set the cluster options in the kubeconfig file.

[student@workstation certificate]$ **oc config set-cluster \**

**api-ocp4-example-com:6443 --certificate-authority ocp-apiserver-cert.crt \**

**--embed-certs --server https://api.ocp4.example.com:6443 \**

**--kubeconfig admin.config**

Cluster "api-ocp4-example-com:6443" set.

* 1. Set the context for the admin user.

[student@workstation certificate]$ **oc config set-context admin \**

**--cluster api-ocp4-example-com:6443 --namespace auth-review --user admin \**

**--kubeconfig admin.config**

Context "admin" created.

* 1. Use the context for the admin user.

[student@workstation certificate]$ **oc config use-context admin \**

**--kubeconfig admin.config**

Switched to context "admin".

* 1. Test the administrator certificate.

[student@workstation certificate]$ **oc whoami --kubeconfig admin.config**

system:admin

* 1. Change to the student HOME directory.

[student@workstation certificate]$ **cd**

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

1. Use the administrator kubeconfig file to configure the LDAP IdP. Use the LDAP IdP information from the following table:

| **Parameter** | **Value** |
| --- | --- |
| DN (-D) | cn=Directory Manager |
| URI (-H) | ldaps://rhds.ocp4.example.com |
| Password (-w) | redhatocp |

1. Use the ~/DO380/labs/auth-review/rhds directory to create any necessary files.
2. You can find the RHDS certificate in that directory with the rhds\_ca.crt name. Use the rhds-ladp-secret and rhds-ca-config-map names for the RHDS secret and configuration map respectively. Use Red Hat Directory Server as the name for the RHDS LDAP IdP.
3. You can find an incomplete example for the OAuth CR in the rhds-ldap-idp.yaml file.
   1. Change to the ~/DO380/labs/auth-review/rhds directory.

[student@workstation ~]$ **cd ~/DO380/labs/auth-review/rhds**

[student@workstation rhds]$

* 1. From the CLI, test the connection with the ldapsearch command by using the information in the table.

[student@workstation rhds]$ **ldapsearch -D "cn=Directory Manager" \**

**-w redhatocp -H ldaps://rhds.ocp4.example.com**

...output omitted...

# paydentomcheck, people, example.com

dn: uid=paydentomcheck,ou=people,dc=example,dc=com

objectClass: top

objectClass: account

objectClass: posixAccount

objectClass: shadowAccount

cn: Payden Tomcheck

uid: paydentomcheck

uidNumber: 10001

gidNumber: 101

homeDirectory: /home/paydentomcheck

loginShell: /bin/bash

gecos: paydentomcheck

shadowLastChange: 0

shadowMax: 0

shadowWarning: 0

userPassword::

e1NTSEF9Wm1sMWd1WjJLajlRM1dKZGlFVnV6aTNEaCs5NzFPeFg=

# administrators, people, example.com

dn: cn=administrators,ou=people,dc=example,dc=com

objectClass: top

objectClass: groupOfUniqueNames

cn: administrators

uniqueMember: uid=paydentomcheck,ou=people,dc=example,dc=com

...output omitted...

* 1. Create the RHDS secret.

[student@workstation rhds]$ **oc create secret generic rhds-ldap-secret \**

**--from-literal bindPassword=redhatocp -n openshift-config \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

secret/rhds-ldap-secret created

* 1. Create the configuration map that contains the certificate.

[student@workstation rhds]$ **oc create configmap rhds-ca-config-map \**

**--from-file ca.crt=rhds\_ca.crt -n openshift-config \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

configmap/rhds-ca-config-map created

* 1. Edit the RHDS LDAP IdP in the OAuth server. This configuration replaces the OAuth configuration by overriding any IdP configuration that was previously in the cluster. You can find an incomplete example for the OAuth CR in the rhds-ldap-idp.yaml file.

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- ldap:

attributes:

id:

- dn

email:

- mail

name:

- cn

preferredUsername:

- uid

bindDN: **'cn=Directory Manager'**

bindPassword:

name: **rhds-ldap-secret**

ca:

name: **rhds-ca-config-map**

insecure: false

url: >-

ldaps://rhds.ocp4.example.com/dc=example,dc=com?uid

mappingMethod: claim

name: **Red Hat Directory Server**

type: LDAP

* 1. Use the oc apply command to configure the RHDS LDAP IdP in the OAuth server.

[student@workstation rhds]$ **oc apply -f rhds-ldap-idp.yaml \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

oauth.config.openshift.io/cluster configured

* 1. Verify the status for the OAuth pods and wait for the OAuth pods to be redeployed. Press **Ctrl**+**C** when all the new pods are running.

[student@workstation rhds]$ **watch oc get pods -n openshift-authentication \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

Every 2.0s: oc get pods -n openshift-authentication workstation: Tue Aug 22 ...

NAME READY STATUS RESTARTS AGE

oauth-openshift-78cbdc45f7-2z887 1/1 Running 0 35s

oauth-openshift-78cbdc45f7-4jsdc 1/1 Running 0 62s

oauth-openshift-78cbdc45f7-rlv2q 1/1 Running 0 89s

**^C**

* 1. Verify that you can log in to the cluster as the paydentomcheck user with redhat123 as the password.

[student@workstation group-sync]$ **oc login -u paydentomcheck -p redhat123 \**

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

Login successful.

...output omitted...

* 1. Change to the student HOME directory.

[student@workstation rhds]$ **cd**

1. Configure the LDAP automated group synchronization in a project called auth-ldapsync.

Use the ldap-group-syncer SA for this purpose. Create the ldap-group-syncer cluster role with get, list, create, and update permissions to the groups resource, and assign the cluster role to the SA.

Use the ldap-secret and ldap-config names for the RHDS secret and configuration map, respectively. Mount the secret and the configuration map in the /etc/secrets/bindPassword and /etc/config/ca.crt files, respectively.

Use the ~/DO380/labs/auth-review/group-sync directory to create any necessary files. You can find the RHDS certificate in that directory with the rhds\_ca.crt name. You can find an incomplete example for the LDAP synchronization configuration in the ldap-sync.yaml file. For the cron job for the group synchronization, you can apply the ready-to-use rhds-groups-cronjob.yaml file.

Assign the cluster-admin role to the administrators group, and verify its permissions.

* 1. Change to the ~/DO380/labs/auth-review/group-sync directory.

[student@workstation ~]$ **cd ~/DO380/labs/auth-review/group-sync**

[student@workstation group-sync]$

* 1. Create a project called auth-ldapsync.

[student@workstation group-sync]$ **oc new-project auth-ldapsync \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

Now using project "auth-ldapsync" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Create a service account called ldap-group-syncer.

[student@workstation group-sync]$ **oc create sa ldap-group-syncer \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

serviceaccount/ldap-group-syncer created

* 1. Create the ldap-group-syncer cluster role.

[student@workstation group-sync]$ **oc create clusterrole ldap-group-syncer \**

**--verb get,list,create,update --resource groups \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

clusterrole.rbac.authorization.k8s.io/ldap-group-syncer created

* 1. Create the cluster role binding for the ldap-group-syncer SA and cluster role.

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

**ldap-group-syncer -z ldap-group-syncer \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

clusterrole.rbac.authorization.k8s.io/ldap-group-syncer added: "ldap-group-syncer"

* 1. Create a secret that contains the LDAP bind password.

[student@workstation group-sync]$ **oc create secret generic ldap-secret \**

**--from-literal bindPassword='redhatocp' \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

secret/ldap-secret created

* 1. Create the LDAP synchronization configuration file. You can find an incomplete example for the LDAP synchronization configuration in the ldap-sync.yaml file.

kind: LDAPSyncConfig

apiVersion: v1

url: ldaps://rhds.ocp4.example.com:636

bindDN: **'cn=Directory Manager'**

bindPassword:

file: **/etc/secrets/bindPassword**

ca: **/etc/config/ca.crt**

augmentedActiveDirectory:

groupsQuery:

baseDN: "ou=people,dc=example,dc=com"

scope: sub

derefAliases: never

pageSize: 0

groupUIDAttribute: dn

groupNameAttributes: [ cn ]

usersQuery:

baseDN: "ou=people,dc=example,dc=com"

scope: sub

derefAliases: never

filter: (objectclass=account)

pageSize: 0

userNameAttributes: [ uid ]

groupMembershipAttributes: [ memberOf ]

* 1. Create a configuration map that contains the LDAPSyncConfig file and the trusted certificate.

[student@workstation group-sync]$ **oc create configmap ldap-config \**

**--from-file ldap-sync.yaml=ldap-sync.yaml,ca.crt=rhds\_ca.crt \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

configmap/ldap-config created

* 1. Review the cron job resource manifest for the group synchronization in the rhds-groups-cronjob.yaml file.

apiVersion: batch/v1

kind: CronJob

metadata:

name: group-sync

namespace: auth-ldapsync

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

template:

spec:

restartPolicy: Never

containers:

- name: ldap-group-sync

image: "registry.ocp4.example.com:8443/openshift4/ose-cli:v4.12"

command:

- "/bin/sh"

- "-c"

- "oc adm groups sync --sync-config=/etc/config/ldap-sync.yaml --confirm"

volumeMounts:

- mountPath: "/etc/config"

name: "ldap-sync-volume"

- mountPath: "/etc/secrets"

name: "ldap-bind-password"

volumes:

- name: "ldap-sync-volume"

configMap:

name: "ldap-config"

- name: "ldap-bind-password"

secret:

secretName: "ldap-secret"

serviceAccountName: ldap-group-syncer

serviceAccount: ldap-group-syncer

|  |  |
| --- | --- |
|  | This command should be written in a single line. |

* 1. Create the cron job for the group synchronization.

[student@workstation group-sync]$ **oc create -f rhds-groups-cronjob.yaml \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

...output omitted...

cronjob.batch/group-sync created

* 1. Wait for one minute for the cron job to start, and verify that OpenShift synchronizes the administrators group from RHDS.

[student@workstation group-sync]$ **oc get groups \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

NAME USERS

Default SMB Group

**administrators paydentomcheck**

admins admin

...output omitted...

* 1. Apply the cluster-admin role to the administrators group.

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

**cluster-admin administrators \**

**--kubeconfig ~/DO380/labs/auth-review/certificate/admin.config**

clusterrole.rbac.authorization.k8s.io/cluster-admin added: "administrators"

* 1. Log in to the cluster as the paydentomcheck user.

[student@workstation group-sync]$ **oc login -u paydentomcheck -p redhat123**

Login successful.

...output omitted...

* 1. Verify that the user has cluster administrator privileges.

[student@workstation group-sync]$ **oc auth can-i create users -A**

yes

* 1. Change to the student HOME directory.

[student@workstation group-sync]$ **cd**

1. Configure the Red Hat SSO OIDC IdP.

The URL for the Red Hat SSO server is https://sso.ocp4.example.com:8080. Use the admin user and the redhatocp password to connect to the Red Hat SSO server. The Red Hat SSO client name is ocp\_rhsso and the realm is external\_providers.

Use the rhsso-oidc-client-secret name for the Red Hat SSO secret. Use RHSSO\_OIDC as the name for the Red Hat SSO IdP.

Use the ~/DO380/labs/auth-review/sso directory to create any necessary files.

You can find an incomplete example for the Red Hat SSO IdP configuration in the sso\_config.yaml file. You must include the IdP configuration for the LDAP server that you set up in a previous step.

* 1. Change to the ~/DO380/labs/auth-review/sso directory.

[student@workstation ~]$ **cd ~/DO380/labs/auth-review/sso**

[student@workstation sso]$

* 1. Connect to the Red Hat SSO machine as the rhsso user.

[student@workstation sso]$ **ssh rhsso@sso.ocp4.example.com**

[rhsso@sso ~]$

* 1. Use the kcadm tool to connect to Red Hat SSO.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh config credentials \**

**--server https://sso.ocp4.example.com:8080/auth \**

**--user admin --password redhatocp --realm master**

Logging into https://sso.ocp4.example.com:8080/auth as user admin of realm master

* 1. List the information for the ocp\_rhsso client from Red Hat SSO.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get clients \**

**-r external\_providers -q clientId=ocp\_rhsso**

[ {

"id" : "***f57e9ddc-8c60-4b40-8048-ec0120595be2***",

"clientId" : "ocp\_rhsso",

"surrogateAuthRequired" : false,

"enabled" : true,

"alwaysDisplayInConsole" : false,

"clientAuthenticatorType" : "client-secret",

"redirectUris" : [ "https://oauth-openshift.apps.ocp4.example.com/\*" ],

"webOrigins" : [ "https://oauth-openshift.apps.ocp4.example.com" ],

...output omitted...

* 1. Generate a JSON file, which contains the client ID, the authentication server URL, and the client secret. Use the ocp\_rhsso ID from the previous step.

[rhsso@sso ~]$ **/opt/rh-sso-7.6/bin/kcadm.sh get \**

**clients/*f57e9ddc-8c60-4b40-8048-ec0120595be2*/installation\**

**/providers/keycloak-oidc-keycloak-json \**

**-r external\_providers > rhsso.json**

|  |  |
| --- | --- |
|  | The /providers/…​ text must come after the …​/installation text in a single line without spaces. |

* 1. View the content of the JSON file, which contains the Red Hat SSO client information. Note the secret, which you use in a later step. The client secret would differ on your system. Use the ocp\_rhsso Red Hat SSO client ID for the clientID parameter in the IdP configuration file on OpenShift. The issuer parameter in the IdP configuration on OpenShift concatenates the value from the auth-server-url parameter, the /​realms/ string, and the external\_providers Red Hat SSO realm name.

[rhsso@sso ~]$ **cat rhsso.json**

{

"realm" : "external\_providers",

"auth-server-url" : "https://sso.ocp4.example.com:8080/auth/",

"ssl-required" : "external",

"resource" : "ocp\_rhsso",

"credentials" : {

"secret" : "***X4ZTPfDr0b8loqOFArfidhaHq85bHyiy***"

},

"confidential-port" : 0

* 1. Return to the workstation machine.

[rhsso@sso ~]$ **exit**

logout

Connection to sso.ocp4.example.com closed.

[student@workstation sso]$

* 1. Connect to the OpenShift cluster as the paydentomcheck user with redhat123 as the password.

[student@workstation sso]$ **oc login -u paydentomcheck -p redhat123 \**

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

Login successful.

...output omitted...

* 1. Create the rhsso-oidc-client-secret OpenShift secret for the Red Hat SSO client secret by using the client secret from a previous step.

[student@workstation sso]$ **oc create secret generic rhsso-oidc-client-secret \**

**--from-literal clientSecret=*X4ZTPfDr0b8loqOFArfidhaHq85bHyiy* \**

**-n openshift-config**

secret/rhsso-oidc-client-secret created

* 1. Edit the Red Hat SSO IdP in the OAuth server. You can find an example for the CR in the sso\_config.yaml file. You must include the IdP configuration for the LDAP server that you set up in a previous step.

[student@workstation sso]$ **oc edit oauth cluster**

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- ldap:

...output omitted...

**- openID:**

claims:

email:

- email

name:

- name

preferredUsername:

- preferred\_username

groups:

- groups

clientID: **ocp\_rhsso**

clientSecret:

name: **rhsso-oidc-client-secret**

extraScopes: []

issuer: >-

**https://sso.ocp4.example.com:8080/auth/realms/external\_providers**

mappingMethod: claim

name: **RHSSO\_OIDC**

type: OpenID

### Note

You can also use the solution in the ~/DO380/solutions/auth-review/sso/sso\_config.yaml file to configure the Red Hat SSO IdP in the OAuth server.

[student@workstation]$ **oc apply -f ~/DO380/solutions/auth-review/sso/sso\_config.yaml**

oauth.config.openshift.io/cluster configured

* 1. Verify the status for the OAuth pods and wait for the OAuth pods to be redeployed. Press **Ctrl**+**C** when all the new pods are running.

[student@workstation sso]$ **watch oc get pods -n openshift-authentication**

Every 2.0s: oc get pods -n openshift-authentication workstation: Thu Aug 22 ...

NAME READY STATUS RESTARTS AGE

oauth-openshift-7d5f479864-6jktw 1/1 Running 0 3m25s

oauth-openshift-7d5f479864-86cgf 1/1 Running 0 3m42s

oauth-openshift-7d5f479864-dc6jw 1/1 Running 0 3m48s

**^C**

* 1. Verify that you can log in to the cluster as the laurenchan user with redhat\_sso as the password.

[student@workstation sso]$ **oc login -u laurenchan -p redhat\_sso**

Login successful.

...output omitted...

* 1. Log in to the cluster as the paydentomcheck user with redhat123 as the password.

[student@workstation sso]$ **oc login -u paydentomcheck -p redhat123**

Login successful.

...output omitted...

* 1. Verify that OpenShift synchronizes the user from Red Hat SSO the first time that they log in.

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

NAME UID FULL NAME IDENTITIES

**laurenchan dbe329a2-... Lauren Chan RHSSO\_OIDC:a175e1b7-...**

paydentomcheck 2554ba58-... Payden Tomcheck Red Hat Directory Server:dWlk...

* 1. Verify that the laurenchan user is a member of the consultants group.

[student@workstation sso]$ **oc get groups**

NAME USERS

Default SMB Group

administrators paydentomcheck

admins admin

**consultants laurenchan**

...output omitted...

* 1. Assign the view cluster role in the auth-review project to the consultants group.

[student@workstation sso]$ **oc adm policy add-role-to-group view consultants \**

**-n auth-review**

clusterrole.rbac.authorization.k8s.io/view added: "consultants"

* 1. Change to the student HOME directory.

[student@workstation sso]$ **cd**

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

1. Configure an external monitoring app to run outside the cluster.

Create the audit-bot SA, generate an SA token, and store the SA token in the AUDITBOT\_TOKEN variable. Use the auth-review project. Create a kubeconfig file with the audit-bot user credentials. Store the kubeconfig file in the ~/DO380/labs/auth-review/log\_script/audit-bot.config file.

Run the cluster-health.sh script to verify the status of the pods in the OpenShift cluster. You can find the script in the ~/DO380/labs/auth-review/log\_script directory. The script uses the credentials that are stored in the ~/DO380/labs/auth-review/log\_script/audit-bot.config file.

* 1. Change to the ~/DO380/labs/auth-review/log\_script directory.
  2. [student@workstation ~]$ **cd ~/DO380/labs/auth-review/log\_script**

[student@workstation log\_script]$

* 1. Change to the auth-review project.
  2. [student@workstation log\_script]$ **oc project auth-review**

Now using project "auth-review" on server "https://api.ocp4.example.com:6443".

* 1. Create the audit-bot SA in the auth-review project.
  2. [student@workstation log\_script]$ **oc create sa audit-bot**

serviceaccount/audit-bot created

* 1. Assign the cluster-reader cluster role to the audit-bot SA so that the SA can retrieve information from most of the objects in the cluster.
  2. [student@workstation log\_script]$ **oc adm policy add-cluster-role-to-user \**
  3. **cluster-reader system:serviceaccount:auth-review:audit-bot**

clusterrole.rbac.authorization.k8s.io/cluster-reader added: "system:serviceaccount:auth-review:audit-bot"

* 1. Generate the audit-bot SA account token and store it in the AUDITBOT\_TOKEN variable.

[student@workstation log\_script]$ **AUDITBOT\_TOKEN=$(oc create token \**

**-n auth-review audit-bot)**

* 1. Add the audit-bot user credentials to the kubeconfig file.

[student@workstation log\_script]$ **oc config set-credentials audit-bot \**

**--token $AUDITBOT\_TOKEN --kubeconfig audit-bot.config**

User "audit-bot" set.

* 1. Set the cluster options in the kubeconfig file.

[student@workstation log\_script]$ **oc config set-cluster \**

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

**--kubeconfig audit-bot.config**

Cluster "api-ocp4-example-com:6443" set.

* 1. Set the context for the audit-bot user.

[student@workstation log\_script]$ **oc config set-context audit-bot \**

**--cluster api-ocp4-example-com:6443 --namespace auth-review --user audit-bot \**

**--kubeconfig audit-bot.config**

Context "audit-bot" created.

* 1. Use the context for the audit-bot user.

[student@workstation log\_script]$ **oc config use-context audit-bot \**

**--kubeconfig audit-bot.config**

Switched to context "audit-bot".

* 1. Verify the identity of the audit-bot user by using the kubeconfig file.

[student@workstation log\_script]$ **oc whoami --kubeconfig audit-bot.config**

system:serviceaccount:auth-review:audit-bot

* 1. Run the cluster-health.sh script to verify the status of the pods in the OpenShift cluster. The script uses the credentials that are stored in the ~/DO380/labs/auth-review/log\_script/audit-bot.config file.

[student@workstation log\_script]$ **sh cluster-health.sh**

Connected to the cluster as the 'system:serviceaccount:auth-review:audit-bot' user

✔ OpenShift is reacheable and up, at version: '4.14.12'

✔ All pods are either running or succeeded.

* 1. Change to the student HOME directory.

[student@workstation log\_script]$ **cd**

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

**Evaluation**

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

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

**Finish**

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

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

# **Chapter 2.  Backup, Restore, and Migration of Applications with OADP**

[Export and Import Application Data and Settings](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02)

[Guided Exercise: Export and Import Application Data and Settings](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s02)

[OADP Operator Deployment and Features](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s03)

[Guided Exercise: OADP Operator Deployment and Features](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s04)

[Backup and Restore with OADP](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s05)

[Guided Exercise: Backup and Restore with OADP](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s06)

[Lab: Backup, Restore, and Migration of Applications with OADP](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s07)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch02s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Back up and restore application settings and data with OpenShift API for Data Protection (OADP). |
| **Sections** | * Export and Import Application Data and Settings (and Guided Exercise) * OADP Operator Deployment and Features (and Guided Exercise) * Backup and Restore with OADP (and Guided Exercise) |
| **Lab** | * Backup, Restore, and Migration of Applications with OADP |

## **Export and Import Application Data and Settings**

### **Objectives**

* Export and import application data and settings between projects.

### **Export and Import a Kubernetes Application**

The ability to export and import a Kubernetes application is useful in many scenarios. The following list describes some of these scenarios:

* Partial or full restoration to a previous working state
* Disaster recovery and business continuity
* Migration from one cluster to another
* Duplication on multiple environments

Backup and restore procedures are critical for an organization to recover from data loss or corruption. The most common data loss scenarios are hardware failures, cyberattacks, software bugs, or human errors.

A Kubernetes application backup includes all the needed resources to restore that application to a previous working state. This backup must include the following artifacts:

* Kubernetes resources that define the application and its settings.
* Container images in the internal registry that containers of this application use.
* Data that is stored in persistent volumes or object storage for a stateful application.

### Note

Application data or configuration that is hosted on external services such as Database-as-a-Service or object storage might be required but is outside the scope of this chapter.

Red Hat OpenShift and Red Hat partners provide data protection solutions for a faster recovery plan. The following list includes examples of data protection solutions:

* OpenShift API for Data Protection
* Veeam Kasten K10
* Storware Backup and Recovery
* IBM Spectrum Protect Plus
* Pure Storage Portworx Backup

### **Backing Up Application Resources**

A Kubernetes application has many resources. The export and import include the following steps:

* List all required resources for the application.
* Export the listed resources.
* Clean the exported resource files.
* Deploy the cleaned resource files.

The first step to export an application is to list all the application resources.

### Note

The oc get all command lists only a subset of resources in a project and does not show resources such as secrets, configuration maps, and so on.

You do not need to list all resources in a project. For example, a MySQL application might require the following resources:

* Deployment
* Service
* Secret

You can list these resources by using the oc get command.

[user@host ~]$ **oc get deployment,svc,secret**

NAME READY UP-TO-DATE AVAILABLE AGE

**deployment.apps/mysql** 1/1 1 1 112s

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

**service/mysql** ClusterIP 172.30.210.241 <none> 9001/TCP 115s

NAME TYPE DATA AGE

secret/builder-dockercfg-skqdg kubernetes.io/dockercfg 1 2m16s

secret/builder-token-cpmxf kubernetes.io/service-account-token 4 2m16s

...output omitted...

**secret/mysql-credentials** Opaque 3 60s

A MySQL application might use persistent volume claim for data. The required resources for application export depend on the application.

#### **Export Application Resources**

You can export an application object to a YAML file by using the oc get command.

[user@host ~]$ **oc -n prod get deployment/mysql -o yaml > backup\_deployment.yaml**

The backup\_deployment.yaml file has all deployment details such as specifications, metadata, and status.

apiVersion: apps/v1

kind: **Deployment**

**metadata:**

annotations:

deployment.kubernetes.io/revision: "2"

labels:

app: mysql

name: mysql

namespace: prod

...output omitted...

**spec:**

progressDeadlineSeconds: 600

replicas: 1

selector:

matchLabels:

app: mysql

...output omitted...

**status:**

availableReplicas: 1

...output omitted...

You can export all other application resources by using the same oc get command.

#### **Import Application Resources**

You can create the application resources in a new project by using the backup YAML files. You can remove the metadata.namespace field and use the oc create -f command to create each resource in the new project.

Apart from the metadata.namespace field, all backup resource files have runtime and status information such as metadata.annotations, metadata.creationTimestamp, metadata.resourceVersion, metadata.generation, and status. You can remove these fields and keep cleaned backup resource files to create resources in a new project.

The following example demonstrates importing some required resources for an application:

[user@host ~]$ **oc create -f clean\_backup\_deployment.yaml -n prod-backup**

deployment.apps/mysql created

[user@host ~]$ **oc create -f service-mysql.yaml -n prod-backup**

The Service "mysql" is invalid: **spec.clusterIPs**: Invalid value: []string{"172.30.210.241"}: failed to allocate IP **172.30.210.241: provided IP is already allocated**

The mysql deployment was created successfully in the prod-backup project by using the backup YAML file.

However, the metadata and status removal does not work for all resources. Some resources require additional modifications.

The service resource creation failed because the IP address is already allocated to the mysql service in the prod project. The service backup YAML file requires more modification for a successful creation.

The modification depends on the resource type. Service resource creation requires the removal of not only the metadata and status fields but also the spec.clusterIP field from the YAML file.

You can use a text editor or other tools to remove these fields. For example, you can use the yq tool to process the YAML file and remove a specific field.

[user@host ~]$ **cat service-mysql.yaml \**

**| yq d - metadata.namespace \**

**| yq d - spec.clusterIP\* > clean-service-mysql.yaml**

[user@host ~]$ **oc create -f clean-service-mysql.yaml -n prod-backup**

service/mysql created

### Note

The yq tool is a command-line YAML processor that is similar to the jq tool for JSON files. The d option removes the specified field from the output.

See the references section for more information about the yq tool.

Similar to the service resource, the route resource creation requires the removal of the metadata.namespace and spec.host fields from the YAML file.

[user@host ~]$ **cat route-frontend.yaml \**

**| yq d - metadata.namespace \**

**| yq d - spec.host > clean-route-frontend.yaml**

[user@host ~]$ **oc create -f clean-route-frontend.yaml -n prod-backup**

route.route.openshift.io/etherpad created

You do not need to create all the exported resources. The new project creates some resources, such as service accounts, secrets, and role bindings. An application also creates some resources. The deployment resource creates the replicasets resource, and the deploymentconfig resource creates the replicationcontrollers resource.

### **Backing Up Container Images**

You can export container images by using container tools, such as podman or skopeo, to copy the images from one registry to another. For more details about Podman and Skopeo, refer to the DO188: Red Hat OpenShift Development I: Introduction to Containers with Podman training course.

Because the OpenShift internal registry is accessible only from within the cluster by default, you can use a Kubernetes job to export the images to a remote location. To access the registry from outside the cluster or to export images from your local machine, an OpenShift administrator must expose the registry externally.

#### **Expose OpenShift Internal Registry**

You can configure the OpenShift internal registry operator to expose the registry externally with the following command:

[user@host ~]$ **oc patch \**

**configs.imageregistry.operator.openshift.io/cluster \**

**--patch '{"spec":{"defaultRoute":true}}' \**

**--type merge**

### Note

This action requires the cluster-admin role.

The modification to the image registry operator triggers a redeployment of the OpenShift API server. It can take up to 10 minutes for the cluster to stabilize.

The operator creates a default-route route to expose externally the registry that uses the following URL format:

default-route-openshift-image-registry.***apps-domain***

### Note

You can use a custom hostname for the registry if needed. See the references section for more information about exposing the internal registry.

You can use the following command to retrieve the registry URL from the created route and save it in an environment variable for later use:

[user@host ~]$ **REGISTRY=$(oc get \**

**route default-route \**

**-n openshift-image-registry \**

**--template '{{.spec.host}}')**

OpenShift users who do not have access to the openshift-image-registry namespace can retrieve the registry URL from any image stream:

[user@host ~]$ **oc -n openshift get is/cli \**

**-ojsonpath="{.status.publicDockerImageRepository}{'\n'}"**

**default-route-openshift-image-registry.apps.ocp4.example.com**/openshift/cli

You can then log in to the internal registry by using your OpenShift username and authentication token with the following command:

[user@host ~]$ **podman login \**

**-u $(oc whoami) \**

**-p $(oc whoami -t) \**

**--tls-verify=false \**

**$REGISTRY**

### Note

If the cluster's default ingress certificate is not trusted, then you must use the --​tls-verify=false option to skip the certificate verification.

To configure a trusted certificate with the Ingress Operator, refer to the Setting a custom default certificate section in the Ingress Operator in OpenShift Container Platform chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#nw-ingress-setting-a-custom-default-certificate_configuring-ingress>

Alternatively, you can use the oc registry login command to log in to the internal registry without the need to specify your credential and registry URL. The oc registry login command automatically uses your authentication token and the internal registry URL from the OpenShift cluster.

The oc command stores the credentials in the ${HOME}/.docker/config.json file in Base64 format. The Podman, Skopeo, and Docker clients can use the authentication details from that file to access an image registry.

[user@host ~]$ **oc registry login**

info: Using registry public hostname default-route-openshift-image-registry.apps.ocp4.example.com

**Warning: the default reading order of registry auth file will be changed from "${HOME}/.docker/config.json" to podman registry config locations in the future version of oc. "${HOME}/.docker/config.json" is deprecated, but can still be used for storing credentials as a fallback**. See https://github.com/containers/image/blob/main/docs/containers-auth.json.5.md for the order of podman registry config locations.

Saved credentials for default-route-openshift-image-registry.apps.ocp4.example.com

### Note

The use of ${HOME}/.docker/config.json file by the oc command is deprecated and will be changed to the ${XDG\_RUNTIME\_DIR}/containers/auth.json file to store credentials in a future version. Although Podman and Skopeo can use the credentials from both files, the Docker client uses only the first file.

You can safely ignore the warning from the oc registry login command that mentions this deprecation, because the rest of the chapter uses only Podman and Skopeo.

#### **Export Container Images**

To export the image from outside the cluster, expose the internal registry and use the skopeo command to copy the image to a remote registry:

[user@host ~]$ **skopeo copy \**

**docker://${REGISTRY}/project\_name/imagestream:tag \**

**docker://remote-registry.example.com/path/to/image:remotetag**

|  |  |
| --- | --- |
|  | Fully qualified source image in the OpenShift internal registry |
|  | Destination registry URL with image and tag information |

Skopeo can copy images to other locations such as a local directory or a .tar archive. See the references section for detailed use of the skopeo copy command.

### Note

You can use the skopeo sync command to copy all the available tags in an image. See the references section for more information about the skopeo sync command.

If Skopeo is not available on your system, then you can use Podman or Docker to pull the image in the local container registry. You can then export the image as a .tar file with the podman save command or push the image to a remote registry.

[user@host ~]$ **podman pull ${REGISTRY}/project\_name/imagestream:tag**

[user@host ~]$ **podman save ${REGISTRY}/project\_name/imagestream:tag \**

**| bzip2 > image\_backup.tar.bz2**

Alternatively, you can use the oc image mirror command to copy images to a local or remote location, similar to the skopeo copy command:

[user@host ~]$ **oc image mirror ${REGISTRY}/project\_name/imagestream:\* \**

**remote-registry.example.com/path/to/image**

|  |  |
| --- | --- |
|  | You can use the wildcard character (\*) to copy all the tags to the destination registry. |

### Note

The oc client that matches your OpenShift version is available in the openshift/cli:latest container image that is included in the OpenShift internal registry.

See the references section for additional examples of the oc image mirror command.

To export a container image from within the cluster, you can use any available container tools in a pod to copy the image to the location of your choice, such as a persistent volume on NFS storage, S3 storage, or a remote registry.

You can use the following YAML file as an example to create a Kubernetes job that exports the container image to a persistent volume by using the OpenShift client:

apiVersion: batch/v1

kind: Job

metadata:

name: backup-image

namespace: application

labels:

app: backup

spec:

backoffLimit: 1

template:

metadata:

labels:

app: backup

spec:

containers:

- name: backup

image: image-registry.openshift-image-registry.svc:5000/openshift/cli:latest

env:

- name: REGISTRY\_AUTH\_FILE

value: /tmp/dockercfg.json

command: ["/bin/bash", "-c"]

args:

- |

**oc registry login**

**oc image mirror \**

**image-registry.openshift-image-registry.svc:5000/application/myapp:\* \**

**file://myapp --dir /backup**

volumeMounts:

- mountPath: **/backup**

name: backup

...output omitted...

volumes:

- name: backup

persistentVolumeClaim:

claimName: **backup-volume**

|  |  |
| --- | --- |
|  | Log in to the internal registry and export all tags of the myapp image stream to the /backup path. |
|  | Volume mount definition for the backup location. |
|  | Volume definition for the backup persistent volume claims (PVC). |

### Note

You need the system:image-puller role on the OpenShift project to pull images from any image streams in that project. Project users and administrators already have this permission, as well as the default service account.

#### **Import Container Images**

You can import an image to an image stream in any of your projects by using the same container tools as for the export. Similar to the export, you can use the following skopeo command to import an image from a remote repository:

[user@host ~]$ **skopeo copy \**

**docker://remote-registry.example.com/myimage:latest \**

**docker://${REGISTRY}/project\_name/mynewimagestream:latest**

### Note

If the image stream does not exist, then OpenShift creates it when the image is pushed.

You can also use the oc image mirror command to copy images in a similar way to the skopeo copy command:

[user@host ~]$ **oc image mirror \**

**remote-registry.example.com/myimage:latest \**

**${REGISTRY}/project\_name/mynewimagestream:latest**

You can use Podman or Docker to import an image, although the process uses more steps, because you must import the image to the local container engine registry and rename the image to use the new registry name.

The following commands import an image in the Podman local registry from a compressed archive and push it to the OpenShift internal registry:

[user@host ~]$ **podman load -i image\_backup.tar.bz2**

...output omitted...

Storing signatures

Loaded image(s): registry.apps.ocp4.example.com/application/myapp:1.2.3

If needed, rename the image with the podman tag command to match the new OpenShift project and image stream:

[user@host ~]$ **podman tag \**

**registry.apps.ocp4.example.com/application/myapp:1.2.3 \**

**${REGISTRY}/newproject/myapp:1.2.3**

Then, send the image to the OpenShift internal registry:

[user@host ~]$ **podman push ${REGISTRY}/newproject/myapp:1.2.3**

...output omitted...

Writing manifest to image destination

Storing signatures

### Note

You need the system:image-pusher role on the OpenShift project to push images to any image streams in that project. Project users and administrators already have this permission, as well as the builder service account.

### **Backing Up Application Data**

Several methods of data backup exist, depending on the application. Some applications provide dedicated tools or procedures to achieve the most reliable data protection and consistency. Different consistency levels can be achieved depending on the backup method.

**Inconsistent backup**

A backup is called inconsistent when the application alters data during the backup process. Traditional data copying when the application is running creates inconsistent backups.

**Crash-consistent backup**

A crash-consistent backup is created by suspending disk I/O during the backup, either by using snapshot technology or specialized tools, to ensure data consistency on disk. Application data in memory or pending I/O operations are not captured. The state of the application is kept as if the application was suddenly shut down due to power loss, or crashed.

**Application-consistent backup**

Application-consistent backup is the most reliable type of backup because it ensures that all in-memory data and pending I/O operations are written on disk before creating the backup.

Some applications provide a set of tools to flush memory data to disk and to pause file system operations on demand. These tools provide an application-consistent backup without any downtime by using snapshots, and is also known as hot backup.

For example, a MySQL database provides the FLUSH TABLES WITH READ LOCK statement to flush all operations to disk and to lock all tables before taking a snapshot. You can then unlock tables with the UNLOCK TABLES statement after the snapshot is created.

Database applications often come with specialized tools to create and restore backups without stopping the application or using volume snapshots. The following list includes examples of specialized tools:

* mysqldump for MySQL and MariaDB
* pg\_dump for PostgreSQL

### Note

Creating backups with specialized tools is out of the scope of this course.

A more universal way to create application-consistent backup is to stop the application, copy the data to another location, and then restart the application. This method is also called cold backup, because the application is down during the operation.

Depending on the amount of data, the application can be unavailable for a long time during the backup operation. By using volume snapshot, you can reduce this downtime to only a few minutes and use the cloned volume to back up while the application is back online.

#### **Volume Snapshot**

Volume snapshot capability is available only with Container Storage Interface (CSI) drivers. However, snapshot functions are not implemented for all CSI drivers. The following table includes examples of CSI drivers with the snapshot capability:

| **Storage provider** | **CSI driver** |
| --- | --- |
| AWS Elastic Block Storage | efs.csi.aws.com |
| Azure Disk | disk.csi.azure.com |
| CephFS | cephfs.csi.ceph.com |
| Ceph RBD | rbd.csi.ceph.com |
| NetApp | csi.trident.netapp.io |

Kubernetes provides similar API resources to PersistentVolume and PersistentVolumeClaim to create and manage volume snapshots.

**VolumeSnapshotClass**

Similar to the storage class for a persistent volume claim, a volume snapshot class describes the CSI driver and associated settings to create a volume snapshot.

### Note

The VolumeSnapshotClass driver must match the StorageClass provisioner of the source PVC.

The following commands list all available storage and volume snapshot classes:

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

NAME DRIVER

ocs-storagecluster-cephfsplugin-snapclass **openshift-storage.cephfs.csi.ceph.com**

ocs-storagecluster-rbdplugin-snapclass **openshift-storage.rbd.csi.ceph.com**

[user@host ~]$ **oc get storageclasses | egrep "^NAME|csi"**

NAME PROVISIONER

ocs-external-storagecluster-cephfs **openshift-storage.cephfs.csi.ceph.com**

ocs-external-storagecluster-ceph-rbd **openshift-storage.rbd.csi.ceph.com**

**VolumeSnapshot**

Similar to the PersistentVolumeClaim resource, a VolumeSnapshot resource requests the creation of a snapshot.

The following example is a YAML file for creating a volume snapshot:

apiVersion: snapshot.storage.k8s.io/v1

kind: VolumeSnapshot

metadata:

name: **my-snapshot**

namespace: **application**

spec:

volumeSnapshotClassName: **ocs-storagecluster-rbdplugin-snapclass**

source:

persistentVolumeClaimName: **application-data**

|  |  |
| --- | --- |
|  | Name of the volume snapshot. |
|  | Namespace of the volume snapshot. It must be the same as the source PVC. |
|  | Snapshot class name for the volume snapshot. |
|  | Name of the source PVC that is used for the snapshot. |

The following command lists volume snapshots. A snapshot is successfully created when the READYTOUSE attribute is set to true and a VolumeSnapshotContent resource is created:

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

NAME READYTOUSE SOURCEPVC ... SNAPSHOTCONTENT

my-snapshot **true** application-data ... **snapcontent-798...cf6**

### Note

When creating an application-consistent backup, the application must be quiesced or scaled down before the snapshot creation. The application can safely be resumed or scaled up after the snapshot is created and ready to use.

**VolumeSnapshotContent**

Similar to the PersistentVolume resource, a VolumeSnapshotContent resource represents a snapshot that a VolumeSnapshot resource created.

After the snapshot is created, it can function as a source to create a PVC and for a pod to use it. The following example is a YAML file for creating a persistent volume from a snapshot:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: **my-snapshot-volume**

namespace: **application**

spec:

storageClassName: **ocs-external-storagecluster-ceph-rbd**

accessModes:

- ReadWriteOnce

dataSource:

apiGroup: snapshot.storage.k8s.io

kind: VolumeSnapshot

name: **my-snapshot**

resources:

requests:

storage: **1Gi**

|  |  |
| --- | --- |
|  | Name of the PVC. |
|  | Namespace of the PVC. It must be the same as the snapshot namespace. |
|  | Storage class name for the PVC. |
|  | Name of the snapshot. |
|  | Size of the new volume. Must be equal to or greater than the snapshot size. |

#### **Export Application Data**

If your application provides specialized backup tools, you can use them to export the data to your chosen location. The following example is a cron job definition to back up a MariaDB database and to store the backup file on AWS S3 storage:

apiVersion: batch/v1

kind: CronJob

metadata:

name: backup-db

spec:

schedule: "​0 0 \* \* \*​"

jobTemplate:

spec:

template:

spec:

initContainers:

- envFrom:

- secretRef:

name: mariadb

image: quay.io/redhattraining/mariadb:10.5

command: ["/bin/bash", "-c"]

args:

- >

**mariadb-dump -u "${MARIADB\_USER}" -p"${MARIADB\_PASSWORD}"**

**-h mariadb "${MARIADB\_DATABASE}"**

**| bzip2 > /backup/backup-$(date '+%Y%m%d-%H%M').sql.bz2;**

**ls -al /backup**

name: backup

volumeMounts:

- mountPath: /backup

name: backup

containers:

- image: docker.io/amazon/aws-cli:latest

command: ["/bin/bash", "-c"]

args:

- >

**aws s3 cp --no-progress /backup/backup\* s3://backup/**

name: s3cli

volumeMounts:

- mountPath: /backup

name: backup

- mountPath: /root/.aws

name: aws-creds

volumes:

- name: backup

emptyDir: {}

- name: aws-creds

secret:

secretName: s3config

|  |  |
| --- | --- |
|  | Use the mariadb-dump tool to export the database to a compressed file in an ephemeral volume. |
|  | Use the AWS CLI tool to send the backup file to AWS S3-compatible storage. |

If the application does not provide backup tools, then you can use volume snapshot to back up the application data. Depending on the storage provider that is available in the OpenShift cluster, you might rather export the snapshot content to an external storage location.

The following example is a job definition that uses an existing snapshot volume to archive the snapshot content, and exports the backup file to a remote S3 bucket:

apiVersion: batch/v1

kind: Job

metadata:

name: backup

namespace: application

labels:

app: backup

spec:

backoffLimit: 1

template:

metadata:

labels:

app: backup

spec:

containers:

- name: backup

image: docker.io/d3fk/s3cmd:latest

command: ["/bin/sh", "-c"]

args:

- |

**tar czf -C /snapshot /tmp/mybackup.tar.gz .**

**s3cmd cp /tmp/mybackup.tar.gz s3://backup/**

volumeMounts:

- mountPath: **/snapshot**

name: snapshot

...output omitted...

volumes:

- name: snapshot

persistentVolumeClaim:

claimName: **my-snapshot-volume**

|  |  |
| --- | --- |
|  | Archive the snapshot content and copy the archive to a remote S3 bucket. |
|  | Volume mount definition for the snapshot data in the container. |
|  | Volume definition for the snapshot PVC. |

### Important

If the volume storage class does not support volume snapshot, then you can mount the application volume instead to export the data. In this case, you must ensure that no other pods are using the volume during the backup, to avoid data inconsistencies.

Another option is to export the snapshot content locally from a pod by using the oc cp command. The following example is a pod definition, where the snapshot data is mounted, which you can use to export the data to your workstation:

apiVersion: v1

kind: Pod

metadata:

name: export

spec:

containers:

- image: registry.access.redhat.com/ubi9:latest

command: ["/bin/bash", "-c"]

args:

- **sleep infinity**

...output omitted...

volumeMounts:

- mountPath: **/snapshot**

name: snapshot

volumes:

- name: snapshot

persistentVolumeClaim:

claimName: **my-snapshot-volume**

|  |  |
| --- | --- |
|  | The sleep infinity command ensures that the pod stays alive during the manual export. |
|  | Volume mount definition for the snapshot data in the container. |
|  | Volume definition for the snapshot PVC. |

You can use the following oc command to copy the snapshot data from the export pod to your local machine:

[user@host ~]$ **oc cp export:/snapshot /tmp/backup**

### Note

The tar binary must be installed in the remote container for the oc cp command to work.

After the snapshot content is exported to a remote location, you can safely remove the snapshot PVC and the volume snapshot, to free up space on the storage back end.

### Important

If your application uses more than one persistent volume, then you must track the backup names and the volumes that they belong to. You need that information to restore each backup to the correct persistent volume.

#### **Import Application Data**

To import the data to another cluster or namespace, create a pod or a job where the application volume is mounted, and copy the exported data to the pod.

The following example is a YAML file for creating a job that fetches a remote backup from an S3 bucket and extracts the archive to the application volume:

apiVersion: batch/v1

kind: Job

metadata:

name: restore

namespace: application

labels:

app: restore

spec:

backoffLimit: 1

template:

metadata:

labels:

app: restore

spec:

containers:

- name: restore

image: docker.io/d3fk/s3cmd:latest

command: ["/bin/bash", "-c"]

args:

- |

**s3cmd cp s3://backup/mybackup.tar.gz /tmp/**

**tar xvzf /tmp/mybackup.tar.gz -C /data**

volumeMounts:

- mountPath: **/data**

name: application-data

...output omitted...

volumes:

- name: application-data

persistentVolumeClaim:

claimName: **application-data**

|  |  |
| --- | --- |
|  | Install the s3cmd tool, fetch the backup from the S3 bucket, and extract the archive inside the application data volume. |
|  | Volume mount definition for the application data in the container. |
|  | Volume definition for the application PVC. |

You can restore the data to an existing volume, or create a new one. If you use a new volume, then you must update the application to use that new volume.

### Important

To avoid data corruption, ensure that the application that uses the volume is not running during the restoration procedure.

### References

For accessing the internal registry, refer to the Accessing the Registry chapter in the Red Hat OpenShift Container Platform 4.14 Registry documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/registry/index#accessing-the-registry>

For exposing the internal registry, refer to the Exposing the Registry chapter in the Red Hat OpenShift Container Platform 4.14 Registry documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/registry/index#securing-exposing-registry>

For more information about Kubernetes Volume Snapshots, refer to the Volume Snapshots section in the Kubernetes documentation at <https://kubernetes.io/docs/concepts/storage/volume-snapshots>

[oc image mirror Usage Examples](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#oc-image-mirror)

[skopeo-copy man page](https://github.com/containers/skopeo/blob/main/docs/skopeo-copy.1.md)

[skopeo-sync man page](https://github.com/containers/skopeo/blob/main/docs/skopeo-sync.1.md)

[yq documentation](https://mikefarah.gitbook.io/yq/v/v3.x/)

## **Guided Exercise: Export and Import Application Data and Settings**

Export all application resources and data for a project, and import them to another project to create a functional copy of a live application.

**Outcomes**

* Export an OpenShift application, and include the settings, container images, and data.
* Restore an application to a different namespace.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start backup-export**

**Instructions**

The development team published a new version of the etherpad application in the OpenShift internal registry and wants to deploy it in production. Before the deployment of the new version, your company requires you to create and restore a backup of the production application into a new stage project and to verify data integrity.

The application to back up is in the production project.

1. Connect to the production etherpad application and create a pad. You use this pad later in this exercise to validate the application restoration.
   1. Open a web browser and navigate to the Etherpad URL:
      * https://etherpad-production.apps.ocp4.example.com
   2. Create a pad named backup and click **OK**.

|  |
| --- |
|  |

* 1. Add a line to the pad with the current date and time, followed by Production backup started.
  2. The application automatically saves changes to the pad. Close the browser tab.

1. Review the resources in the production project.
   1. Open a terminal on the workstation and log in to the OpenShift cluster as the developer user with the developer password.

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

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

Login successful.

You have one project on this server: "production"

Using project "production".

* 1. List the resources in the production project with the oc get all command.

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

NAME READY STATUS RESTARTS AGE

pod/etherpad-6f9598bbb5-tmnt8 1/1 Running 0 109s

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

service/etherpad ClusterIP 172.30.210.241 <none> 9001/TCP 111s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/etherpad 1/1 1 1 109s

NAME DESIRED CURRENT READY AGE

replicaset.apps/etherpad-6f9598bbb5 1 1 1 109s

NAME ... TAGS UPDATED

imagestream.image.openshift.io/etherpad ... 1.8.18,1.9.1 109s

NAME ... PORT TERMINATION WILDCARD

route.route.openshift.io/etherpad ... http edge/Redirect None

### Note

The oc get all command shows only a subset of all available resources in a namespace, and might omit some resources that the application requires.

* 1. The etherpad application deployment requires the following resource types:
     + PersistentVolumeClaim
     + Service
     + Route
     + Deployment

Use the oc get command to list these resource types in the production project, and compare the result with the previous step.

[student@workstation ~]$ **oc get pvc,svc,route,deployment**

NAME STATUS ... AGE

persistentvolumeclaim/etherpad Bound ... 43m

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

service/etherpad ClusterIP 172.30.96.26 <none> 9001/TCP 43m

NAME ... PORT TERMINATION WILDCARD

route.route.openshift.io/etherpad ... http edge/Redirect None

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/etherpad 1/1 1 1 43m

1. Export the Kubernetes resources from the previous step and save them in the ~/DO380/labs/backup-export/production directory.
   1. Change to the ~/DO380/labs/backup-export directory.
   2. [student@workstation ~]$ **cd ~/DO380/labs/backup-export**

[student@workstation backup-export]$

* 1. Create a production directory.

[student@workstation backup-export]$ **mkdir production**

* 1. Export the persistent volume claim resource to a YAML file named 01-pvc.yml. Clean up the YAML file by removing the following fields:
     + metadata.annotations
     + metadata.creationTimestamp
     + metadata.namespace
     + metadata.finalizers
     + metadata.resourceVersion
     + metadata.uid
     + spec.volumeName
     + status
     + [student@workstation backup-export]$ **oc get pvc etherpad -o yaml \**
     + **| yq d - metadata.annotations \**
     + **| yq d - metadata.creationTimestamp \**
     + **| yq d - metadata.namespace \**
     + **| yq d - metadata.finalizers \**
     + **| yq d - metadata.resourceVersion \**
     + **| yq d - metadata.uid \**
     + **| yq d - spec.volumeName \**
     + **| yq d - status \**

**> production/01-pvc.yml**

* 1. Export the deployment resource to a YAML file named 02-deployment.yml. Clean up the YAML file by removing the following fields:
     + metadata.annotations
     + metadata.creationTimestamp
     + metadata.namespace
     + metadata.resourceVersion
     + metadata.uid
     + metadata.generation
     + status
     + [student@workstation backup-export]$ **oc get deployment etherpad -o yaml \**
     + **| yq d - metadata.annotations \**
     + **| yq d - metadata.creationTimestamp \**
     + **| yq d - metadata.namespace \**
     + **| yq d - metadata.resourceVersion \**
     + **| yq d - metadata.uid \**
     + **| yq d - metadata.generation \**
     + **| yq d - status \**

**> production/02-deployment.yml**

* 1. Export the service resource to a YAML file named 03-service.yml. Clean up the YAML file by removing the following fields:
     + metadata.annotations
     + metadata.creationTimestamp
     + metadata.namespace
     + metadata.resourceVersion
     + metadata.uid
     + spec.clusterIP
     + spec.clusterIPs
     + status
     + [student@workstation backup-export]$ **oc get svc etherpad -o yaml \**
     + **| yq d - metadata.annotations \**
     + **| yq d - metadata.creationTimestamp \**
     + **| yq d - metadata.namespace \**
     + **| yq d - metadata.resourceVersion \**
     + **| yq d - metadata.uid \**
     + **| yq d - spec.clusterIP\* \**
     + **| yq d - status \**

**> production/03-service.yml**

* 1. Export the route resource to a YAML file named 04-route.yml. Clean up the YAML file by removing the following fields:
     + metadata.annotations
     + metadata.creationTimestamp
     + metadata.namespace
     + metadata.resourceVersion
     + metadata.uid
     + spec.host
     + status
     + [student@workstation backup-export]$ **oc get route etherpad -o yaml \**
     + **| yq d - metadata.annotations \**
     + **| yq d - metadata.creationTimestamp \**
     + **| yq d - metadata.namespace \**
     + **| yq d - metadata.resourceVersion \**
     + **| yq d - metadata.uid \**
     + **| yq d - spec.host \**
     + **| yq d - status \**

**> production/04-route.yml**

* 1. Review the exported resource files.

[student@workstation backup-export]$ **tree production**

production/

├── 01-pvc.yml

├── 02-deployment.yml

├── 03-service.yml

└── 04-route.yml

0 directories, 4 files

1. As the admin user, expose the internal registry to enable users to export and import container images.
   1. Log in to the OpenShift cluster as the admin user with the redhatocp password.

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

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

Login successful.

...output omitted...

* 1. Expose the internal registry.

[student@workstation backup-export]$ **oc patch \**

**configs.imageregistry.operator.openshift.io/cluster \**

**--patch '{"spec":{"defaultRoute":true}}' --type merge**

config.imageregistry.operator.openshift.io/cluster patched

* 1. Wait until the openshift-apiserver operator is redeployed. It can take a couple of minutes. Press **Ctrl**+**C** to exit the watch command.

[student@workstation backup-export]$ **watch -n10 oc get co openshift-apiserver**

NAME VERSION AVAILABLE PROGRESSING ... MESSAGE

openshift-apiserver 4.14.12 True True ... APIServerDeploymentProgressing: deployment/apiserver.openshift-apiserver: 1/3 pods have been updated to the latest generation

openshift-apiserver 4.14.12 True True ... APIServerDeploymentProgressing: deployment/apiserver.openshift-apiserver: 2/3 pods have been updated to the latest generation

openshift-apiserver 4.14.12 True **False** ...

**^C**

1. As the developer user, export all container images that are referenced in the etherpad image stream and save them in the ~/DO380/labs/backup-export/production directory.
   1. Log in to the OpenShift cluster as the developer user with the developer password.

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

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

Login successful.

You have one project on this server: "production"

Using project "production".

* 1. Log in to the internal registry by using the oc registry login command.

[student@workstation backup-export]$ **oc registry login**

info: Using registry public hostname default-route-openshift-image-registry.apps.ocp4.example.com

Saved credentials for default-route-openshift-image-registry.apps.ocp4.example.com into /run/user/1000/containers/auth.json

* 1. Identify the fully qualified image name of the etherpad image and save it as the IMAGE environment variable.

[student@workstation backup-export]$ **IMAGE=$(oc get is etherpad \**

**--template '{{.status.publicDockerImageRepository}}')**

[student@workstation backup-export]$ **echo ${IMAGE}**

default-route-openshift-image-registry.apps.ocp4.example.com/production/etherpad

* 1. Export all container images from the etherpad image stream by using the oc image mirror command.

[student@workstation backup-export]$ **oc image mirror \**

**${IMAGE} file://etherpad --dir=production**

...output omitted...

sha256:057c...2513 file://etherpad:1.9.1

sha256:7265...8ec1 file://etherpad:1.8.18

info: Mirroring completed in 2.96s (116.3MB/s)

### Note

The oc image mirror stores the exported images in the v2 directory.

* 1. Review the exported images in the production directory.

[student@workstation backup-export]$ **tree production**

production/

...output omitted...

└── v2

└── etherpad

├── blobs

│ ├── sha256:057c...a2513

...output omitted...

└── manifests

├── 1.8.18 -> sha256:7265...28ec1

├── 1.9.1 -> sha256:057c...2513

├── sha256:057c...2513

└── sha256:7265...28ec1

4 directories, 38 files

1. Create a snapshot of the persistent volume to limit the downtime of the application during the backup operation.
   1. List the persistent volumes and identify the associated storage class.

[student@workstation backup-export]$ **oc get pvc**

NAME STATUS ... STORAGECLASS AGE

**etherpad** Bound ... **ocs-external-storagecluster-ceph-rbd** 25m

* 1. Identify the driver that is used for the ocs-external-storagecluster-ceph-rbd storage class.

[student@workstation backup-export]$ **oc get storageclass \**

**ocs-external-storagecluster-ceph-rbd**

NAME PROVISIONER ...

ocs-external-storagecluster-ceph-rbd **openshift-storage.rbd.csi.ceph.com** ...

* 1. Get the volume snapshot storage class name from the openshift-storage.rbd.csi.ceph.com driver.

[student@workstation backup-export]$ **oc get volumesnapshotclasses \**

**| egrep "^NAME|openshift-storage.rbd.csi.ceph.com"**

NAME ...

**ocs-external-storagecluster-rbdplugin-snapclass** ...

* 1. Scale down the application.

[student@workstation backup-export]$ **oc scale deployment/etherpad --replicas 0**

deployment.apps/etherpad scaled

* 1. Verify that no application pods are running.

[student@workstation backup-export]$ **oc get pods**

No resources found in production namespace.

* 1. Create a volume snapshot of the etherpad persistent volume by using the volume snapshot class from the previous steps. You can use the template in the ~/DO380/labs/backup-export/volumesnapshot.yml path.

apiVersion: snapshot.storage.k8s.io/v1

kind: VolumeSnapshot

metadata:

name: **etherpad**

spec:

volumeSnapshotClassName: **ocs-external-storagecluster-rbdplugin-snapclass**

source:

persistentVolumeClaimName: **etherpad**

### Note

You can use the solution in the ~/DO380/solutions/backup-export/volumesnapshot-etherpad.yml file.

* 1. Create the volume snapshot resource.

[student@workstation backup-export]$ **oc apply -f volumesnapshot.yml**

volumesnapshot.snapshot.storage.k8s.io/etherpad created

* 1. Verify that the volume snapshot is created and ready to use.

[student@workstation backup-export]$ **oc get volumesnapshot**

NAME READYTOUSE SOURCEPVC ... CREATIONTIME AGE

etherpad **true** etherpad ... 12s 13s

* 1. Scale up the application.

[student@workstation backup-export]$ **oc scale deployment/etherpad --replicas 1**

deployment.apps/etherpad scaled

1. Export the application data from the volume snapshot and save it in the ~/DO380/labs/backup-export/production directory.
   1. Create a persistent volume claim by using the data from the volume snapshot. You can use the template in the ~/DO380/labs/backup-export/pvc-snapshot.yml path.

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: etherpad-snapshot

labels:

app: etherpad-snapshot

spec:

storageClassName: **ocs-external-storagecluster-ceph-rbd**

accessModes:

- ReadWriteOnce

dataSource:

apiGroup: **snapshot.storage.k8s.io**

kind: **VolumeSnapshot**

name: **etherpad**

resources:

requests:

storage: 1Gi

### Note

You can use the solution in the ~/DO380/solutions/backup-export/pvc-snapshot.yml file.

* 1. Create the PVC snapshot resource.

[student@workstation backup-export]$ **oc apply -f pvc-snapshot.yml**

persistentvolumeclaim/etherpad-snapshot created

* 1. Verify that the persistent volume claim status is Bound.

[student@workstation backup-export]$ **oc get pvc etherpad-snapshot**

NAME STATUS ... STORAGECLASS AGE

etherpad-snapshot **Bound** ... ocs-external-storagecluster-ceph-rbd 117s

* 1. Mount the snapshot volume on /snapshot in a new pod named export-snapshot. You can use the template in the ~/DO380/labs/backup-export/pod-snapshot.yml file.

apiVersion: v1

kind: Pod

metadata:

name: export-snapshot

labels:

app: export-snapshot

spec:

containers:

...output omitted...

volumeMounts:

- mountPath: **/snapshot**

name: **snapshot**

terminationGracePeriodSeconds: 2

volumes:

- name: **snapshot**

persistentVolumeClaim:

claimName: **etherpad-snapshot**

### Note

You can use the solution in the ~/DO380/solutions/backup-export/pod-snapshot.yml file.

* 1. Create the pod snapshot resource.

[student@workstation backup-export]$ **oc apply -f pod-snapshot.yml**

pod/export-snapshot created

* 1. Verify that the export-snapshot pod is running.

[student@workstation backup-export]$ **oc get pods export-snapshot**

NAME READY STATUS RESTARTS AGE

**export-snapshot** 1/1 **Running** 0 31s

* 1. Copy the snapshot data from the pod to the production/data directory in the workstation machine.

[student@workstation backup-export]$ **oc cp \**

**export-snapshot:/snapshot production/data**

tar: Removing leading '/' from member names

* 1. Review the exported data.

[student@workstation backup-export]$ **tree production**

production/

...output omitted...

├── data

│ ├── dirty.db

│ ├── lost+found

│ ├── minified\_1f08...0f7e

│ ├── minified\_1f08...0f7e.gz

│ ├── minified\_5a47...57ca

│ ├── minified\_5a47...57ca.gz

│ ├── minified\_c24d...5b91

│ └── minified\_c24d...5b91.gz

...output omitted...

6 directories, 45 files

* 1. Delete the export-snapshot pod.

[student@workstation backup-export]$ **oc delete pod/export-snapshot**

pod "export-snapshot" deleted

1. Import the application container to the stage project.
   1. Create the stage project.

[student@workstation backup-export]$ **oc new-project stage**

Now using project "stage" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Import the etherpad container images to the stage project by using the oc image mirror command.

[student@workstation backup-export]$ **oc image mirror \**

**--dir=production file://etherpad \**

**default-route-openshift-image-registry.apps.ocp4.example.com/stage/etherpad**

...output omitted...

sha256:7265...8ec1 default-route.../stage/etherpad:1.8.18

sha256:057c...2513 default-route.../stage/etherpad:1.9.1

info: Mirroring completed in 4.98s (69.07MB/s)

* 1. Review the etherpad image stream.

[student@workstation backup-export]$ **oc describe is/etherpad**

...output omitted...

Image Repository: default-route.../stage/etherpad

Image Lookup: local=false

Unique Images: 2

Tags: 2

**1.9.1**

no spec tag

\* **image-registry...stage/etherpad@sha256:057c...2513**

24 seconds ago

**1.8.18**

no spec tag

\* **image-registry...stage/etherpad@sha256:7265...8ec1**

38 seconds ago

1. Import the production application data to the stage project.
   1. Import the persistent volume claim definition.

[student@workstation backup-export]$ **oc apply -n stage -f production/01-pvc.yml**

persistentvolumeclaim/etherpad created

* 1. Verify that the persistent volume claim status is Bound.

[student@workstation backup-export]$ **oc get pvc**

NAME STATUS VOLUME ... AGE

etherpad **Bound** pvc-1d2139df-8969-4dc8-affd-c473e301480f ... 4s

* 1. Mount the volume on /data in a new pod named restore-snapshot. You can use the template in the ~/DO380/labs/backup-export/pod-restore.yml file.

apiVersion: v1

kind: Pod

metadata:

name: restore-snapshot

labels:

app: restore-snapshot

spec:

containers:

...output omitted...

volumeMounts:

- mountPath: **/data**

name: **data**

terminationGracePeriodSeconds: 2

volumes:

- name: **data**

persistentVolumeClaim:

claimName: **etherpad**

### Note

You can use the solution in the ~/DO380/solutions/backup-export/pod-restore.yml file.

* 1. Create the restore-snapshot pod.

[student@workstation backup-export]$ **oc apply -f pod-restore.yml**

pod/restore-snapshot created

* 1. Verify that the restore-snapshot pod is running.

[student@workstation backup-export]$ **oc get po restore-snapshot**

NAME READY STATUS RESTARTS AGE

**restore-snapshot** 1/1 **Running** 0 65s

* 1. Copy the backup data from the workstation machine to the /data directory in the pod.

[student@workstation backup-export]$ **oc rsync --no-perms \**

**./production/data/ restore-snapshot:/data/**

sending incremental file list

dirty.db

minified\_5dbe...03cb

minified\_5dbe...03cb.gz

minified\_8214...cacf

minified\_8214...cacf.gz

minified\_bd66...b58f

minified\_bd66...b58f.gz

sent 776,429 bytes received 150 bytes 1,553,158.00 bytes/sec

total size is 775,441 speedup is 1.00

* 1. Review the imported data.

[student@workstation backup-export]$ **oc exec restore-snapshot -- \**

**ls -1 /data**

dirty.db

lost+found

minified\_5dbe...03cb

minified\_5dbe...03cb.gz

minified\_8214...cacf

minified\_8214...cacf.gz

minified\_bd66...b58f

minified\_bd66...b58f.gz

* 1. Delete the restore-snapshot pod.

[student@workstation backup-export]$ **oc delete pod/restore-snapshot**

pod "restore-snapshot" deleted

1. Import the application Kubernetes resources to the stage project.
   1. Create a copy of the deployment YAML file and name it stage-deployment.yml.

[student@workstation backup-export]$ **cp production/02-deployment.yml \**

**stage-deployment.yml**

* 1. Modify the stage-deployment.yml file with the following parameters:

| **Parameter** | **Value** |
| --- | --- |
| Container image | image-registry.openshift-image-registry.svc:5000/stage/etherpad:1.8.18 |
| TITLE environment variable | DO380 - stage etherpad |

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app.kubernetes.io/name: etherpad

name: etherpad

spec:

...output omitted...

spec:

containers:

- env:

- name: **TITLE**

value: **DO380 - stage etherpad**

- name: DEFAULT\_PAD\_TEXT

value: Add some content and write your ideas

- name: SUPPRESS\_ERRORS\_IN\_PAD\_TEXT

value: "true"

- name: EXPOSE\_VERSION

value: "true"

image: **image-registry.openshift-image-registry.svc:5000/stage/etherpad:1.8.18**

...output omitted...

initContainers:

- args:

...output omitted...

image: **image-registry.openshift-image-registry.svc:5000/stage/etherpad:1.8.18**

imagePullPolicy: IfNotPresent

name: clean

...output omitted...

### Note

* 1. The TITLE environment variable defines the instance name for this application. It is also the name of the browser window when accessing the application.
  2. Import the deployment definition to the stage project.
  3. [student@workstation backup-export]$ **oc apply -n stage -f stage-deployment.yml**

deployment.apps/etherpad created

* 1. Import the service definition to the stage project.
  2. [student@workstation backup-export]$ **oc apply -n stage \**
  3. **-f production/03-service.yml**

service/etherpad created

* 1. Import the route definition to the stage project.
  2. [student@workstation backup-export]$ **oc apply -n stage \**
  3. **-f production/04-route.yml**

route.route.openshift.io/etherpad created

* 1. Review all imported resources.
  2. [student@workstation backup-export]$ **oc get pvc,is,svc,route,deployment**
  3. NAME STATUS ...
  4. persistentvolumeclaim/etherpad Bound ...
  5. NAME ... TAGS UPDATED
  6. imagestream.image.openshift.io/etherpad ... 1.8.18,1.9.1 4 minutes ago
  7. NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
  8. service/etherpad ClusterIP 172.30.222.62 <none> 9001/TCP 29s
  9. NAME HOST/PORT ...
  10. route.route.openshift.io/etherpad etherpad-stage.apps.ocp4.example.com ...
  11. NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/etherpad 1/1 1 1 103s

1. Connect to the stage etherpad application and open the backup pad.
   1. Open a web browser and navigate to the stage Etherpad application URL. Confirm the instance name in the browser window.
      * https://etherpad-stage.apps.ocp4.example.com
   2. Open and review the pad named backup.

If the restoration is successful, then you should see the same content from the beginning of this section.

* 1. Close the browser tab.

1. Clean up the resources.
   1. Change to the student HOME directory.
   2. [student@workstation backup-export]$ **cd**

[student@workstation ~]$

* 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. Remove the exposed route from the OpenShift internal registry.
  2. [student@workstation ~]$ **oc patch \**
  3. **configs.imageregistry.operator.openshift.io/cluster \**
  4. **--patch '{"spec":{"defaultRoute":false}}' --type merge**

config.imageregistry.operator.openshift.io/cluster patched

### Note

The cluster image registry patch triggers a kube-apiserver deployment rollout. The lab finish command might take up to 10 minutes for the cluster to stabilize.

* 1. Delete the stage project.
  2. [student@workstation ~]$ **oc delete project stage**

project.project.openshift.io "stage" deleted

**Finish**

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

[student@workstation ~]$ **lab finish backup-export**

## **OADP Operator Deployment and Features**

### **Objectives**

* Deploy the OADP operator and describe its features and dependencies.

### **OpenShift API for Data Protection**

The OpenShift API for Data Protection (OADP) product provides a native backup solution for applications that run on OpenShift.

OADP uses several components such as Velero and Kopia to back up all the Kubernetes resources, container images from the internal registry, and persistent volumes that are associated with an application.

|  |
| --- |
|  |

The following list describes the main components that OADP uses:

**Velero**

Velero is the main upstream component of OADP. It provides the backup API to the users and uses plug-ins to add extended capabilities to OADP.

**Data Mover**

This feature enables exporting snapshot content to object storage by using Kopia and the CSI plug-in.

**Kopia**

Open source backup tool that Velero and Data Mover use to back up persistent volumes.

Those components are discussed in more detail later in this chapter. See the references section for more information about those components.

The OADP framework is initially designed for partners to provide the needed infrastructure for a complete backup solution. OpenShift administrators can use OADP as a basic backup solution, although it might not implement some required features, such as a graphical user interface or multi-tenancy support. The following examples are backup solutions that use OADP:

* Catalogic CloudCasa
* IBM Spectrum Protect Plus
* Dell EMC PowerProtect Data Manager

#### **Kubernetes Resources Backup**

OADP uses Velero to back up all the Kubernetes resources to cloud object storage. You can filter objects to back up and restore by namespace, resource type, and label.

Velero not only exports and imports resources; it also performs additional processing on the resources. The following list describes some of this additional processing:

* Determine the correct restore order for each resource.
* Apply any user-provided filters.
* Modify the destination namespace, if it is different from the backup.
* Remove or alter resource attributes such as auto-assigned node ports, IP addresses, or hostnames.
* Skip managed resources such as a replica set that deployment owns.

You can add backup and export logic by using Velero plug-ins.

#### **Data Volume Snapshot**

OAPD can back up volumes by using cloud native snapshot API or the Kubernetes Container Storage Interface (CSI) snapshot API.

Red Hat recommends the use of snapshots when possible, because this feature enables fast and consistent backup.

#### **File System Backup**

For volumes that are not supported by either a cloud-native snapshot API or the CSI snapshot API, OADP provides a file-system backup solution.

File System Backup (FSB), or Pod Volume Backup, enables backing up almost any kind of Kubernetes volume that does not support the snapshot capability, such as NFS and local storage.

### Warning

FSB does not support hostPath volumes.

OADP uses the open source Kopia tool to back up the application volume file system.

Kopia accesses the volume file system from the volume mount point on the cluster node. For that reason, the application pod must be running during the backup.

During the file-system backup, the application can still alter the file system and corrupt the backup. To prevent such a scenario, the application must be quiesced to prevent any write operations while the backup is in progress.

#### **Application-Consistent Backup**

You can use backup and restore hooks to run commands before and after the backup and restore operation. By using a pre-hook, it is possible to quiesce an application, such as a transactional database, to flush pending I/O operations to the storage back end and pause future application writing operations to avoid any data loss or corruption during the backup. When the backup is complete, a post-hook is used to resume the application operations.

If you use volume snapshots, then the hooks are executed just before and after the snapshot creation, which significantly reduces the time that the application is in read-only mode.

#### **OADP Data Mover**

Some CSI drivers might store the snapshot in the same storage location as the original volume and might be vulnerable to data loss in the event of a disaster. The storage solution that is used with Red Hat OpenShift Container Platform might not guarantee snapshot durability. In such scenarios, you can enable the OADP Data Mover to copy the snapshot content to a remote backup storage location and remove the snapshot from the storage back end.

Data Mover uses Kopia to upload and encrypt snapshot content to a unified backup repository on the object storage. The Kopia unified backup repository is explained later in this chapter.

### Note

Earlier OADP versions than v1.3 use Restic instead of Kopia to back up persistent volumes. For OADP v1.3, Restic is still available at installation, but Data Mover requires you to enable Kopia.

#### **Velero Plug-ins**

You can extend the OADP capabilities and backup logic by using the Velero plug-in system. Support for various cloud providers, storage solutions, and additional CRDs are available with integrated plug-ins.

OADP includes the following plug-ins that you can enable during the installation:

**aws**

Stores and retrieves backups on AWS S3-compatible storage. Manages volume snapshots on AWS EBS volumes.

**gcp**

Stores and retrieves backups on Google Cloud Storage. Manages volume snapshots on Google Compute Engine Disks.

**azure**

Stores and retrieves backups on Azure Blob Storage. Manages volume snapshots on Azure Managed Disks.

**openshift**

Provides additional logic to back up and restore OpenShift resources, including container images from the internal registry.

**kubevirt**

Provides additional logic to back up and restore virtual machines and other OpenShift Virtualization resources.

**csi**

Provides volume snapshot support by using the Kubernetes CSI Snapshot API.

OADP can also use third-party plug-ins, such as the following examples, to extend support to additional storage solutions and custom Kubernetes resources.

* OpenStack plug-in for object storage on Swift and volume snapshots on Cinder.
* HPE plug-in for volume snapshots on HPE Nimble Storage.
* DigitalOcean plug-in for volume snapshots on DigitalOcean Volumes Block Storage.

### Important

Red Hat does not support third-party plug-ins.

#### **OADP API Resources**

The OADP operator provides the following API resources:

**DataProtectionApplication**

The dataProtectionApplication resource is the configuration for the OADP operator and its components.

**BackupStorageLocation**

The OADP operator creates and manages a backupStorageLocation resource for each storage location that is defined in the dataProtectionApplication configuration.

**VolumeSnapshotLocation**

The OADP operator creates and manages a volumeSnapshotLocation resource for each snapshot location that is defined in the dataProtectionApplication configuration.

**Backup**

The backup resource requests the Velero server to create a backup.

**Restore**

The restore resource requests the Velero server to restore a backup.

**Schedule**

The schedule resource creates a backup periodically on a given schedule by using the Cron format.

**BackupRepository**

The BackupRepository resource tracks the lifecycle of the backup repositories.

### **Requirements for OADP**

Red Hat OpenShift API for Data Protection requires the following resources.

#### **OpenShift User Permissions**

OADP configuration requires a user with the cluster-admin role. Backup and restore operations require administrative privileges in the OADP namespace (openshift-adp by default).

### Important

OADP deployment has administrative privileges on the cluster to back up and restore all resources in any namespaces.

#### **Object Storage**

OADP requires an object storage location to store the backups. The following list is an example of object storage providers that are available for storing backups:

* Amazon Web Services
* Microsoft Azure
* Google Cloud Platform
* Any AWS S3-compatible provider, such as MinIO or OpenShift Data Foundation

#### **Object Storage with OpenShift Data Foundation**

OpenShift Data Foundation (ODF) is the storage solution that is used in this classroom environment. ODF provides two object storage services: the MultiCloud Object Gateway (NooBaa MCG) and the Ceph RADOS Object Gateway (Ceph RGW). Both services are fully compatible with OADP by using the aws provider plug-in.

### Note

For more details about OpenShift Data Foundation, refer to the DO370: Enterprise Kubernetes Storage with Red Hat OpenShift Data Foundation training course.

Similar to a persistent volume claim, you can use an object bucket claim to request an S3-compatible bucket. ODF creates a secret and a configuration map with the same name as the object bucket claim with the credentials and information to access the bucket.

The following is an example of an object bucket claim definition:

apiVersion: objectbucket.io/v1alpha1

kind: ObjectBucketClaim

metadata:

name: my-bucket-claim

namespace: my-namespace

spec:

storageClassName: ***storage-class-name***

generateBucketName: my-bucket

|  |  |
| --- | --- |
|  | Storage class name for the object bucket. To use NooBaa MCG, set the value to openshift-storage.noobaa.io. To use the Ceph RGW gateway, set the value to ocs-external-storagecluster-ceph-rgw. |
|  | Prefix to add to the generated bucket name. |

A few minutes after the object bucket claim creation, the status is set to Bound when the bucket is created and ready to use.

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

NAME STORAGE-CLASS PHASE AGE

my-bucket-claim openshift-storage.noobaa.io **Bound** 5m

ODF creates a configuration map with the bucket information in the same namespace as the object bucket claim:

[user@host ~]$ **oc get configmap/my-bucket-claim -o yaml**

apiVersion: v1

data:

BUCKET\_HOST: **s3.openshift-storage.svc**

BUCKET\_NAME: **my-bucket-9ce46e22-2fb8-4a46-af95-f6949d87c3fd**

BUCKET\_PORT: **"443"**

BUCKET\_REGION: ""

BUCKET\_SUBREGION: ""

kind: ConfigMap

...output omitted...

|  |  |
| --- | --- |
|  | S3 API URL, for internal use only. |
|  | The generated name of the bucket. |
|  | The TCP port for the S3 endpoint. The port 443 is for https protocol. |
|  | The region name, if used by the storage class. |

If the BUCKET\_HOST uses an internal service URL that ends with .svc, then the URL can be reached only from a workload that is running inside the cluster. The s3 route in the openshift-storage namespace provides the external endpoint URL.

[user@host ~]$ **oc get route/s3 -n openshift-storage \**

**-o jsonpath='{.spec.host}{"\n"}'**

s3-openshift-storage.apps.ocp4.example.com

The S3 bucket credentials are stored in a secret in the same namespace as the object bucket claim:

[user@host ~]$ **oc extract secret/my-bucket-claim --to=-**

# AWS\_ACCESS\_KEY\_ID

**YEAsbMJnG3o1bGANZprt**

# AWS\_SECRET\_ACCESS\_KEY

**xjaeCDhskn7lfrdA7WqzoUxpiRYuyjc9uDaWlMw3**

#### **CSI Snapshot**

To use volume snapshots with CSI drivers, a volume snapshot class must be created to register the CSI driver to use to create snapshots. The driver name must be the same as the provisioner attribute of the volume storage class to back up.

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

NAME PROVISIONER

nfs-storage (default) k8s-sigs.io/nfs-subdir-external-provisioner

ocs-storagecluster-ceph-rbd **openshift-storage.rbd.csi.ceph.com**

ocs-storagecluster-ceph-rgw openshift-storage.ceph.rook.io/bucket

ocs-storagecluster-cephfs **openshift-storage.cephfs.csi.ceph.com**

openshift-storage.noobaa.io openshift-storage.noobaa.io/obc

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

NAME DRIVER

ocs-storagecluster-cephfsplugin-snapclass **openshift-storage.cephfs.csi.ceph.com**

ocs-storagecluster-rbdplugin-snapclass **openshift-storage.rbd.csi.ceph.com**

### Note

For storage classes that do not support snapshots, such as nfs-storage in the previous example, you can use the file-system backup.

### **OADP Installation and Configuration**

You can install the OADP operator from the OperatorHub or by using the oc command. See the references section for instructions to install an operator by using the OperatorHub or the oc command.

### Note

For more details about Kubernetes operators, refer to the DO280: Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster training course.

The dataProtectionApplication resource defines the configuration and components that the OADP operator manages. The following YAML file is an example of a dataProtectionApplication resource that stores backups on AWS S3 storage:

apiVersion: oadp.openshift.io/v1alpha1

kind: DataProtectionApplication

metadata:

name: oadp-backup

namespace: openshift-adp

spec:

configuration:

velero:

defaultPlugins:

- aws

- openshift

- csi

defaultSnapshotMoveData: true

nodeAgent:

enable: true

uploaderType: kopia

backupLocations:

- velero:

config:

profile: "default"

region: "us-east-1"

provider: aws

default: true

credential:

key: cloud

name: cloud-credentials

objectStorage:

bucket: my-bucket

prefix: oadp

The OADP configuration is composed of several sections.

#### **Components Configuration**

The configuration section describes the configuration OADP components, such as Velero and Kopia.

configuration:

velero:

defaultPlugins:

- aws

- openshift

- csi

defaultSnapShotMoveData: true

nodeAgent:

enable: true

uploaderType: kopia

podConfig:

resourceAllocations:

limits:

cpu: "1"

memory: 8Gi

requests:

cpu: 500m

memory: 256Mi

|  |  |
| --- | --- |
|  | Velero plug-ins to enable. The openshift plug-in is mandatory. The csi plug-in is necessary for snapshots and Data Mover. |
|  | Set this parameter to true to use the default Data Mover. |
|  | The agent that manages volume backups. |
|  | Enable the node agent if you use either FSB or Data Mover. |
|  | You can use either Kopia or Restic as the uploader. If you use the Data Mover, then you must use Kopia. |
|  | Optional pod configuration such as node selector, labels, and resource allocation for each component. |

#### **Backup Storage Location**

The backupLocation section describes the object storage location for backups. You can define multiple backup locations. The following example uses the aws provider plug-in:

backupLocations:

- velero:

config:

profile: "default"

region: "us-east-1"

s3Url: https://s3.openshift-storage.svc

s3ForcePathStyle: "true"

insecureSkipTLSVerify: "true"

provider: aws

default: true

credential:

key: cloud

name: cloud-credentials

objectStorage:

bucket: my-bucket

prefix: oadp

caCert: LLS0S0...LS0tLS0K

|  |  |
| --- | --- |
|  | Storage provider-specific configuration. In this example, the configuration is for the aws provider. |
|  | The credentials profile name that is defined in the Velero secret. |
|  | The storage provider plug-in name. |
|  | Use this backup location as the default one if none is specified in the backup and restore resources. |
|  | Velero secret with the storage provider credentials. |
|  | Bucket name from the storage provider. |
|  | Optional subdirectory in the bucket to store backups. If not specified, then the backups are stored at the root level of the bucket. |
|  | Optional CA bundle in Base64 format to verify TLS connections to the storage provider. |

With the aws provider plug-in, you can configure any S3-compatible object storage by setting the s3Url attribute with a custom S3 endpoint URL. OADP can access the S3 endpoint by using two addressing models:

* The path-style model, such as https://s3.mydomain.com/*bucket-name*/myfile.txt
* The DNS-style model (also known as virtual-hosted style), such as \https://*bucket-name*.s3.mydomain.com/myfile.txt This method is the default.

If the object storage solution does not support the DNS-style method, then you can configure OADP to use the path-style model instead with the s3ForcePathStyle option.

The region attribute must be set even if the storage solution does not use a region. You can then set this attribute to the standard AWS region, such as "us-east-1".

### Note

ImageStream backup does not work with the caCert option. If the S3 endpoint certificate is not trusted, then the insecureSkipTLSVerify attribute is required to back up container images.

The Velero secret contains the credentials to access the object storage. If the secret name is not specified in the configuration, then Velero uses the cloud-credentials default secret name. You can create the Velero secret with a configuration file that is specific to the provider plug-in. The following Velero configuration file is for the aws provider plug-in:

[user@host ~]$ **cat credentials-velero**

[myProfile]

aws\_access\_key\_id=***AWS\_ACCESS\_KEY\_ID***

aws\_secret\_access\_key=***AWS\_SECRET\_ACCESS\_KEY***

|  |  |
| --- | --- |
|  | The profile name for the credentials. The name must match the profile name in the backup storage location configuration. |

You can then create the Velero secret with the following oc create secret command:

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

**--from-file cloud=*credentials-velero***

### Note

The Velero secret must be created before the DataProtectionApplication object, or the installation will fail.

#### **Snapshot Storage Location**

The snapshotLocations section describes the storage configuration to use for volume snapshots when using the cloud provider native snapshot capability. For CSI snapshots, this section is not needed, because volume snapshot classes are used instead. With Data Mover and FSB, backups are stored in the defined object storage location in the backupLocations section.

You can configure multiple snapshot locations per provider. The following example defines one snapshot location by using the aws provider plug-in:

snapshotLocations:

- name: default

velero:

provider: aws

config:

region: us-west-2

profile: "default"

credential:

key: cloud

name: cloud-credentials

|  |  |
| --- | --- |
|  | AWS region to use to create the snapshots. |
|  | The credentials profile name that is defined in the Velero secret. |
|  | **Optional**: Velero secret with the storage provider credentials. |

If the secret name is not specified in the configuration, then Velero uses the cloud-credentials default secret name. The same secret can be shared with the snapshot location and the backup location.

### Note

Depending on the storage provider, the region to store snapshots might be limited to the same region where the original volume is. Cross-provider snapshots are not supported.

See the references section for configuring OADP with other cloud providers such as Microsoft Azure and Google Cloud Platform.

#### **CSI Snapshots**

For OADP to use a volume snapshot class, you must add the velero.io/csi-volumesnapshot-class: "true" label. Only one volume snapshot class per driver needs this label.

You can use the following command to add the required label:

[user@host ~]$ **oc label volumesnapshotclass *my-snapshot-storageclass* \**

**velero.io/csi-volumesnapshot-class="true"**

***my-snapshot-storageclass*** labeled

### **Verification and Troubleshooting**

When the configuration is complete, several OADP components are created depending on the enabled features.

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

NAME READY UP-TO-DATE AVAILABLE AGE

openshift-adp-controller-manager 1/1 1 1 24h

velero 1/1 1 1 25s

|  |  |
| --- | --- |
|  | OADP controller that is deployed on the OADP operator installation. |
|  | Velero deployment that manages backup and restore operations. |

If the Node Agent that manages volume backups is enabled, then the following daemon set is created:

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

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE

node-agent 3 3 3 3 3 <none> 23s

|  |  |
| --- | --- |
|  | The node-agent is deployed on all compute nodes, to back up volume file systems. |

The OADP operator creates and validates each backup storage location that is defined in the DataProtectionApplication object. OADP tries to connect to the object storage with the provided credentials and validates the configuration. If the configuration is correct, then the backup storage location enters the Available phase.

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

NAME PHASE LAST VALIDATED AGE DEFAULT

oadp-backup-1 Available 7s 2m44s true

If the backup storage location remains in the Unavailable phase, then you can use the oc describe command to get more information and error messages from the Status field:

[user@host ~]$ **oc -n openshift-adp describe \**

**backupstoragelocation oadp-backup-1**

...output omitted...

Status:

Last Synced Time: 2023-09-22T10:27:27Z

Last Validation Time: 2023-09-22T12:03:27Z

Message: BackupStorageLocation "oadp-backup-1" is unavailable: rpc error: code = Unknown desc = InvalidAccessKeyId: The AWS access key Id you provided does not exist in our records.

Phase: Unavailable

...output omitted...

In this example, the S3 credentials that are configured in the Velero secret are incorrect.

### Note

The OADP operator checks the validity of the backup storage location configuration every minute. Review the last validated field of the backup storage location to determine when the last check ran.

You can use the AWS CLI or the s3cmd tool to validate the S3 configuration and to browse the content of the S3 bucket.

To configure the s3cmd tool, create a .s3cfg file in your home directory with your S3 credentials. You can use the following configuration template:

access\_key = ***AWS\_ACCESS\_KEY\_ID***

secret\_key = ***AWS\_SECRET\_ACCESS\_KEY***

host\_base = **s3.mydomain.com**

host\_bucket = **s3.mydomain.com/%(bucket)s**

signature\_v2 = **True**

|  |  |
| --- | --- |
|  | S3 access key and secret key. |
|  | S3 endpoint URL, if the AWS S3 endpoint is not used. |
|  | S3 bucket address model, if your storage provider does not support the DNS-style model. |
|  | If your storage provider does not support the new AWS v4 signature, then you can use the AWS v2 signature instead. OpenShift Data Foundation requires this setting. |

To list the contents of a bucket and to validate the configuration, use the s3cmd ls command to list all available buckets and the s3cmd la command to list all objects in all buckets:

### Note

If the S3 bucket is empty, then the s3cmd la command returns an empty line.

[user@host ~]$ **s3cmd ls**

2023-09-22 12:33 s3://my-bucket

[user@host ~]$ **s3cmd la**

2023-09-22 12:31 143 s3://my-bucket/somefile.txt

...output omitted...

### References

For more information about installing operators by using the OperatorHub, refer to the Installing from OperatorHub Using the Web Console section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-installing-from-operatorhub-using-web-console_olm-adding-operators-to-a-cluster>

For more information about installing operators by using the command line, refer to the Installing from OperatorHub Using the CLI section in the Administrator Tasks chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/operators/index#olm-installing-operator-from-operatorhub-using-cli_olm-adding-operators-to-a-cluster>

For more information about configuring OADP with other cloud object storage, refer to the Installing and Configuring OADP section in the Application Backup and Restore chapter in the Red Hat OpenShift Container Platform 4.14 Backup and Restore documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/backup_and_restore/index#installing-and-configuring-oadp>

For more information about the OADP v1.3 Data Mover, refer to the OADP 1.3 Data Mover section in the Application Backup and Restore chapter in the Red Hat OpenShift Container Platform 4.14 Backup and Restore documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/backup_and_restore/index#oadp-1-3-data-mover>

[Velero](https://velero.io/)

[Kopia](https://kopia.io/)

[S3cmd: Command Line S3 Client](https://s3tools.org/s3cmd)

## **Guided Exercise: OADP Operator Deployment and Features**

Deploy the OADP operator and perform a backup to validate that OADP is functional in a cluster.

**Outcomes**

* Install and configure the OpenShift API for Data Protection operator.
* Back up an application by using OADP to validate the configuration.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start backup-oadp**

**Instructions**

Install and configure the OADP operator with the CSI snapshot and the OADP Data Mover feature.

Create an S3-compatible object storage bucket with OpenShift Data Foundation and configure it as a backup storage location.

To validate the configuration, back up the application in the production project by using OADP and verify that the backup is successfully created in the S3 bucket.

### Important

All other exercises in this chapter depend on your cluster being correctly configured to use OpenShift API for Data Protection.

If you cannot complete this exercise, then delete and re-create your lab environment. After re-creating your lab environment, you can either attempt this exercise again, or the start script for another exercise can configure OADP for you.

1. As the admin user, locate and go to the OpenShift web console.
   1. Log in to your OpenShift cluster as the admin user.

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

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

Login successful.

...output omitted...

* 1. Identify the URL for the OpenShift web console.
  2. [student@workstation ~]$ **oc whoami --show-console**

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

* 1. Open a web browser and go to https://console-openshift-console.apps.ocp4.example.com.
  2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.

1. Install the OADP operator.
   1. Click **Operators** → **OperatorHub**. In the **Filter by keyword** field, type OADP to locate the OADP operator, and then click **OADP Operator**.

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

* 1. The web console displays information about the OADP operator. Select the stable-1.3 channel and the 1.3.1 version. Click **Install** to proceed to the **Install Operator** page.

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

* 1. Review the default configuration and click **Install** to install the operator.
  2. Wait until the installation is complete and the web console displays ready for use.

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

1. Create an S3-compatible object storage bucket for OADP to store backups.
   1. From the command line, change to the openshift-adp project.
   2. [student@workstation ~]$ **oc project openshift-adp**

Now using project "openshift-adp" on server "https://api.ocp4.example.com:6443".

* 1. Change to the ~/DO380/labs/backup-oadp directory.
  2. [student@workstation ~]$ **cd ~/DO380/labs/backup-oadp**

[student@workstation backup-oadp]$

* 1. Create an object bucket claim by using the following values:

| **Parameter** | **Value** |
| --- | --- |
| Name | backup |
| Namespace | openshift-adp |
| Storage class name | openshift-storage.noobaa.io |
| Generate bucket name | backup |

* 1. You can use the resource definition in the ~/DO380/labs/backup-oadp/obc-backup.yml path.

apiVersion: objectbucket.io/v1alpha1

kind: ObjectBucketClaim

metadata:

name: **backup**

namespace: **openshift-adp**

spec:

storageClassName: **openshift-storage.noobaa.io**

generateBucketName: **backup**

### Important

* 1. This environment uses ODF as a back end for both application data storage and backup storage for simplicity. In a disaster situation, both data and backups can be lost at the same time.
  2. For production environment, Red Hat recommends using a different storage location for backups and for application data.
  3. Create the backup OBC.

[student@workstation backup-oadp]$ **oc apply -f obc-backup.yml**

objectbucketclaim.objectbucket.io/backup created

* 1. Verify that the object bucket claim is created and in the Bound phase.

[student@workstation backup-oadp]$ **oc get obc**

NAME STORAGE-CLASS PHASE ...

backup openshift-storage.noobaa.io **Bound** ...

1. Retrieve the S3 bucket information and credentials. You can use a second terminal for this step to make it easier to copy and paste values in the remainder of this section.

When an object bucket claim is created, Red Hat OpenShift Data Foundation creates a matching secret and configuration map with the same name that contains the bucket information and credentials.

* 1. Retrieve the bucket name and bucket host from the generated configuration map.

[student@workstation backup-oadp]$ **oc extract --to=- cm/backup**

# BUCKET\_HOST

**s3.openshift-storage.svc**

# BUCKET\_NAME

**backup-7d9...f4c**

# BUCKET\_PORT

443

# BUCKET\_REGION

# BUCKET\_SUBREGION

### Note

The s3.openshift-storage.svc service uses a TLS certificate that is signed with the self-signed service CA that OpenShift manages. To prevent a certificate signed by unknown authority error, you must include the CA certificate in the OADP configuration.

* 1. Retrieve the service CA certificate for the s3.openshift-storage.svc endpoint. This certificate is available in the openshift-service-ca.crt configuration map in any namespace.

Encode the certificate in Base64 format and save the value for the next step.

[student@workstation backup-oadp]$ **oc get cm/openshift-service-ca.crt \**

**-o jsonpath='{.data.service-ca\.crt}' | base64 -w0; echo**

...output omitted...

* 1. Retrieve the bucket credentials from the generated secret.

[student@workstation backup-oadp]$ **oc extract --to=- secret/backup**

# AWS\_ACCESS\_KEY\_ID

**JNmbmhD0AQ3BFMABtXC4**

# AWS\_SECRET\_ACCESS\_KEY

**xjkwp8bXeJazTgC4u/WJTbzgiD0tfWGt8OtdADLz**

### Note

The values might be different in your environment.

* 1. Identify the public URL for the S3 endpoint from the s3 route in the openshift-storage namespace. You use this URL to connect to the S3 bucket from the workstation machine in the next step.

[student@workstation backup-oadp]$ **oc get route s3 -n openshift-storage**

NAME HOST/PORT ...

s3 **s3-openshift-storage.apps.ocp4.example.com** ...

1. Configure the s3cmd command to browse and validate the object storage configuration.
   1. Create a ~/.s3cfg configuration file in the student home directory with the S3 credentials and S3 endpoint URL from the previous step.

You can use the sample configuration file in the ~/DO380/labs/backup-oadp/s3cfg path.

[student@workstation backup-oadp]$ **vim ~/.s3cfg**

access\_key = ***AWS\_ACCESS\_KEY\_ID***

secret\_key = ***AWS\_SECRET\_ACCESS\_KEY***

host\_base = s3-openshift-storage.apps.ocp4.example.com

host\_bucket = s3-openshift-storage.apps.ocp4.example.com/%(bucket)s

signature\_v2 = True

### Note

Replace the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY values with the values from the previous step.

* 1. List the content of the bucket by using the s3cmd command to validate the configuration.

[student@workstation backup-oadp]$ **s3cmd la**

### Note

Because the bucket is empty at this stage, the command returns an empty line. The command returns an error message if the configuration is incorrect.

1. Prepare the DataProtectionApplication configuration in a new dpa-backup.yml YAML file.

Enable the csi plug-in and the Data Mover feature for the CSI snapshot.

Use the S3 bucket information from the previous steps to configure the default backup storage location.

### Note

You can find an example of the resource definition in the ~/DO380/labs/backup-oadp/dpa-backup.yml file.

apiVersion: oadp.openshift.io/v1alpha1

kind: DataProtectionApplication

metadata:

name: **oadp-backup**

namespace: **openshift-adp**

spec:

configuration:

nodeAgent:

enable: **true**

uploaderType: kopia

velero:

defaultPlugins:

- aws

- openshift

- **csi**

defaultSnapshotMoveData: **true**

backupLocations:

- velero:

config:

profile: "default"

region: "us-east-1"

s3Url: **https://s3.openshift-storage.svc**

s3ForcePathStyle: "true"

insecureSkipTLSVerify: "true"

provider: aws

default: true

credential:

key: cloud

name: cloud-credentials

objectStorage:

bucket: **backup-9e2...20b**

prefix: oadp

caCert: **LLS0S0...LS0tLS0K**

|  |  |
| --- | --- |
|  | The s3ForcePathStyle attribute must be set to true when using ODF. |
|  | Secret with the S3 credentials. You create the cloud-credentials secret in a following step. |
|  | Use the bucket name from the previous step. |
|  | Use the service CA certificate in Base64 format from the previous step. |

1. Create the cloud-credentials secret in the openshift-adp namespace with the backup object bucket credentials.
   1. Create a cloud-credentials file with the object bucket credentials. You can use the configuration example in the ~/DO380/labs/backup-oadp/cloud-credentials file.
   2. [default]
   3. aws\_access\_key\_id=***AWS\_ACCESS\_KEY\_ID***

aws\_secret\_access\_key=***AWS\_SECRET\_ACCESS\_KEY***

### Note

Replace the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY values with the values from the previous step.

* 1. Create the cloud-credentials secret with the cloud-credentials file content.
  2. [student@workstation backup-oadp]$ **oc create secret generic \**
  3. **cloud-credentials \**
  4. **-n openshift-adp \**
  5. **--from-file cloud=cloud-credentials**

secret/cloud-credentials created

1. Apply the OADP configuration by using the dpa-backup.yml YAML file from an earlier step.
2. [student@workstation backup-oadp]$ **oc apply -f dpa-backup.yml**

dataprotectionapplication.oadp.openshift.io/oadp-backup created

1. Verify that both the velero deployment object and the kopia node-agent daemon set are created in the openshift-adp namespace and in the Ready state.
2. [student@workstation backup-oadp]$ **oc get deploy,daemonset -n openshift-adp**
3. NAME READY UP-TO-DATE ...
4. deployment.apps/openshift-adp-controller-manager 1/1 1 ...
5. deployment.apps/velero 1/1 1 ...
6. NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE

daemonset.apps/node-agent 3 3 3 3 3

1. Verify that the backup storage location object is created and in the Available phase.
2. [student@workstation backup-oadp]$ **oc get BackupStorageLocation**
3. NAME PHASE LAST VALIDATED AGE DEFAULT

oadp-backup-1 **Available** 1m 10m true

### Note

It can take a minute for the backup storage location resource to enter the Available phase. If the resource is stuck in the Unavailable phase, then use the oc describe BackupStorageLocation command to get the status and error messages that relate to the configuration.

1. Configure all available volume snapshot classes for OADP.
   1. List all available volume snapshot classes.

[student@workstation backup-oadp]$ **oc get volumesnapshotclass**

NAME ...

ocs-external-storagecluster-cephfsplugin-snapclass ...

ocs-external-storagecluster-rbdplugin-snapclass ...

* 1. For each volume snapshot class, set the velero.io/csi-volumesnapshot-class label to true.

[student@workstation backup-oadp]$ **oc label volumesnapshotclass \**

**velero.io/csi-volumesnapshot-class="true" --all**

.../ocs-external-storagecluster-cephfsplugin-snapclass labeled

.../ocs-external-storagecluster-rbdplugin-snapclass labeled

1. Back up the production project to validate the OADP configuration.
   1. Modify the resource definition in the ~/DO380/labs/backup-oadp/backup.yml file to match the following specification.

apiVersion: velero.io/v1

kind: Backup

metadata:

name: **backup-production**

namespace: openshift-adp

spec:

includedNamespaces:

- **production**

* 1. Create the backup resource.
  2. [student@workstation backup-oadp]$ **oc apply -f backup.yml**

backup.velero.io/backup-production created

1. Wait a couple of minutes and check the backup completion status.
   1. Verify that the backup object is in the Completed phase.

[student@workstation backup-oadp]$ **oc describe backup backup-production**

...output omitted...

Status:

Backup Item Operations Attempted: 1

Backup Item Operations Completed: 1

Completion Timestamp: 2023-09-07T16:29:25Z

Expiration: 2023-10-07T16:28:10Z

Format Version: 1.1.0

Phase: Completed

Progress:

Items Backed Up: 42

Total Items: 42

Start Timestamp: 2023-09-07T16:28:10Z

Version: 1

Events: <none>

* 1. Use the s3cmd command to review the content of the S3 storage.

[student@workstation backup-oadp]$ **s3cmd la -r**

2024-05-31 14:20 2597 s3://backup-78...1a/docker/registry/v2/.../data

...output omitted...

2024-05-31 14:20 77529 s3://backup-78...1a/oadp/backups/...tar.gz

...output omitted...

2024-05-31 14:20 286923 s3://backup-78...1a/oadp/kopia/...

...output omitted...

|  |  |
| --- | --- |
|  | The /docker/registry path contains the container images. |
|  | The /oadp/backups path contains the Kubernetes resources and the logs of the backup. |
|  | The /oadp/kopia path is the Kopia backup repository that contains backups of all data volumes in a project. |

1. Clean up the resources.
   1. Change to the student HOME directory.

[student@workstation backup-oadp]$ **cd**

[student@workstation ~]$

* 1. Use the velero command to remove the backup and all associated resources.

[student@workstation ~]$ **oc exec deployment/velero -c velero -it -- \**

**./velero delete backup backup-production**

Are you sure you want to continue (Y/N)? **y**

Request to delete backup "backup-production" submitted successfully.

The backup will be fully deleted after all associated data (disk snapshots, backup files, restores) are removed.

* 1. Wait a couple of minutes and verify that the backup object is deleted.

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

No resources found in openshift-adp namespace.

* 1. Use the s3cmd command to clean up the object storage bucket. Use the bucket name from previous steps.

[student@workstation ~]$ **s3cmd rm -r --force \**

**s3://*backup-7d9d5169-6c8d-4b40-bbba-931ddaeb6f4c*/**

...output omitted...

* 1. Delete the backuprepository resource in the openshift-adp namespace.

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

NAME AGE REPOSITORY TYPE

***production-oadp-backup-1-kopia-qfs64*** 7m59s kopia

[student@workstation ~]$ **oc delete backuprepository \**

***production-oadp-backup-1-kopia-qfs64* -n openshift-adp**

backuprepository.velero.io "production-oadp-backup-1-kopia-qfs64" deleted

**Finish**

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

[student@workstation ~]$ **lab finish backup-oadp**

## **Backup and Restore with OADP**

### **Objectives**

* Configure one-time and scheduled backups with OADP and restore from them.

### **OADP Custom Resources**

Backup and restore are initiated by creating the corresponding Kubernetes resources in the openshift-adp namespace. Administrative access to the openshift-adp namespace, or to the cluster-admin role, is required to create those resources.

OADP provides the following custom resources for backup and restore:

**Backup**

The backup resource initiates a single backup attempt. This resource defines the namespaces and resources to include in the backup, and can also include a list of commands to run before or after the backup.

The backup resource definition and the backup information, such as backup logs and the list of included backup resources, are stored in the object storage with the backup.

OADP synchronizes backup definitions between the object storage and the OpenShift cluster to enable restoring backups to a different cluster with the same backup storage location.

If a backup resource exists in the OpenShift cluster but is deleted from the object storage, then OADP deletes the Kubernetes resource. Conversely, if a backup exists in the object storage, but not in OpenShift, then OADP creates the matching backup resource in the cluster.

### Note

Only backups with a Completed state are synchronized. The object storage synchronization does not automatically create or remove backup resources with a Failed or PartiallyFailed state.

**Restore**

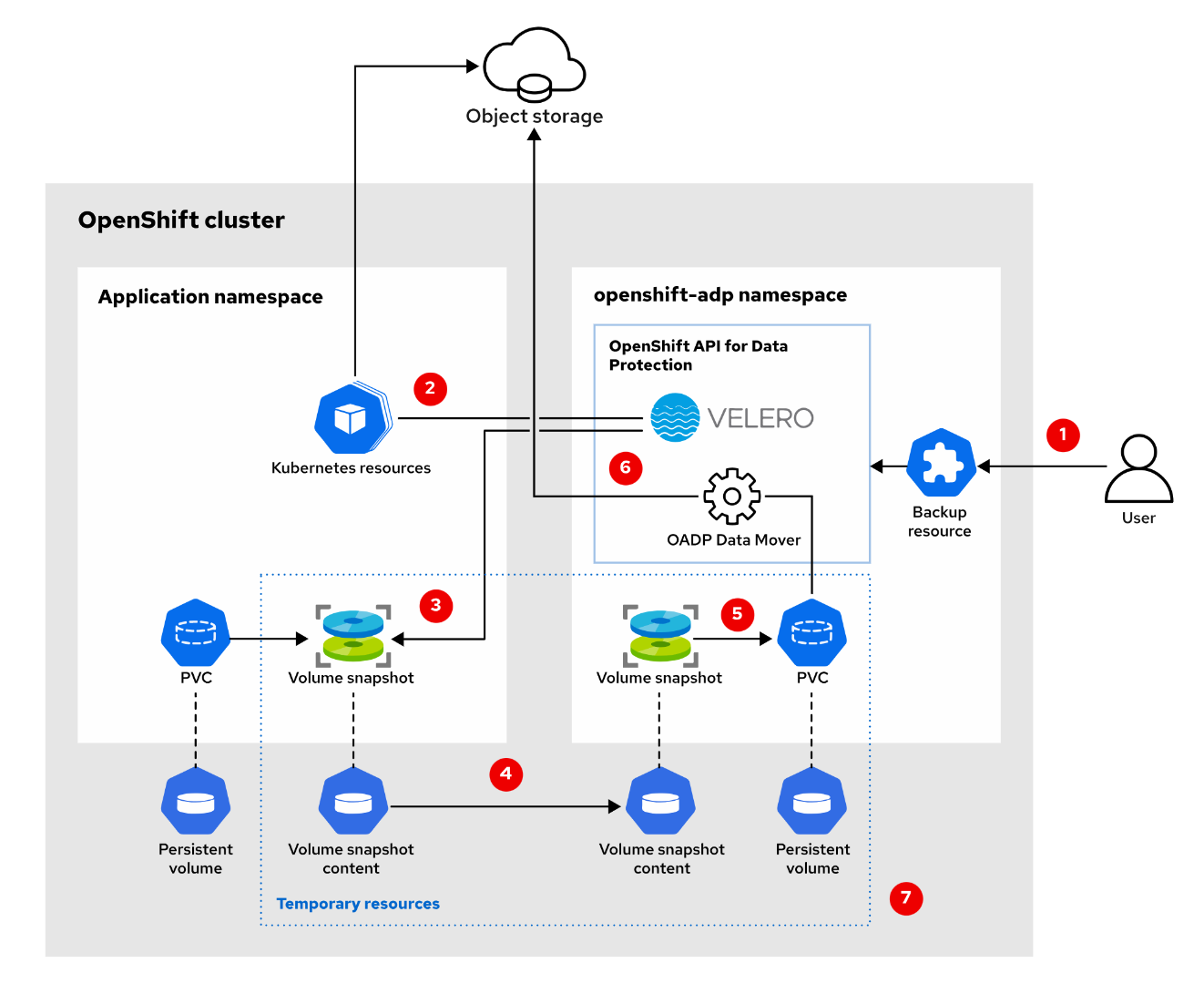
The restore resource starts restoring an existing backup resource. The restore result, the list of restored resources, and the restore logs are stored in the object storage.

**Schedule**

The schedule resource starts a backup on a given schedule that is written in Cron format. A schedule resource is similar to a cron job resource. A schedule defines a backup template to create a backup resource at a recurring interval.

### **Backing up and Restoring an OpenShift Application**

OADP backs up OpenShift applications by using the following process:



|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The administrator creates a backup resource in the openshift-adp namespace, which triggers the backup process. |
|  | Velero exports all Kubernetes resources from the application namespace to the backup storage location. |
|  | The Velero CSI plug-in creates a CSI snapshot of the application volume. |
|  | Data Mover clones the volumeSnapshotContent and volumeSnapshot resources to the openshift-adp namespace. |
|  | Data Mover creates a PVC from the volume snapshot. |
|  | Data Mover uses Kopia to transfer the volume data to the backup storage location. |
|  | After the backup completes, Data Mover deletes all volume snapshots and the PVCs that were created during the backup process. |

For restoring backups, OADP uses the following process:

A diagram of a software diagram

Description automatically generated

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The administrator creates a restore resource in the openshift-adp namespace, which triggers the restore process. |
|  | Velero imports the Kubernetes resources from the backup storage location to the application namespace. |
|  | Data Mover creates a temporary PVC in the openshift-adp namespace, and transfers the application data from the backup storage location to the new volume. |
|  | Velero creates the PVC in the application namespace with the same PV as the one that Data Mover is restoring. The PVC stays in the pending state until Data Mover completes the restore. |
|  | After the transfer is complete, Data Mover releases the PV from the temporary PVC in the openshift-adp namespace, and binds that PV to the final PVC in the application namespace. |

#### **Backing up and Restoring a Stateless Application**

You can start backing up a namespace by using the following backup definition:

apiVersion: velero.io/v1

kind: Backup

metadata:

name: my-app-backup

namespace: openshift-adp

spec:

includedNamespaces:

- my-app-project

ttl: 720h0m0s

labelSelector:

matchLabels:

app: my-app

includedResources:

- deployments

- configmaps

- secrets

- services

- routes

|  |  |
| --- | --- |
|  | Name of the backup |
|  | List of namespaces to back up |
|  | Amount of time before the automatic deletion of the backup. If the ttl value is not specified, then the default is 30 days. |
|  | Labels that are required for the resources to be included in the backup |
|  | List of resource types to back up |

The following example shows a stateless application, followed by the needed definition to back it up in the website namespace. The application is a static website that is built with Hugo and that uses a Source-to-Image (S2I) container image. This application uses the following resources that are created with the app=hugo label:

* Image Stream
* Build Configuration
* Deployment
* Service
* Route

You can back up this application with the following backup definition:

apiVersion: velero.io/v1

kind: Backup

metadata:

name: website

namespace: openshift-adp

spec:

includedNamespaces:

- **website**

labelSelector:

matchLabels:

**app: hugo**

includedResources:

- **imagestreams**

- **buildconfigs**

- **deployments**

- **services**

- **routes**

You can review the configuration and the backup status with the oc describe command:

[user@host ~]$ **oc -n openshift-adp describe backup/website**

Name: website

Namespace: openshift-adp

...output omitted...

Spec:

Csi Snapshot Timeout: 10m0s

Default Volumes To Fs Backup: false

Included Namespaces:

website

Included Resources:

imagestreams

buildconfigs

deployments

services

routes

Item Operation Timeout: 1h0m0s

Label Selector:

Match Labels:

App: hugo

Snapshot Move Data: true

Storage Location: oadp-config-1

Ttl: 720h0m0s

Status:

Backup Item Operations Attempted: 1

Backup Item Operations Completed: 1

Completion Timestamp: 2023-12-11T14:01:42Z

Expiration: 2024-01-10T14:01:27Z

Format Version: 1.1.0

Phase: Completed

Progress:

Items Backed Up: 7

Total Items: 7

Start Timestamp: 2023-12-11T14:01:27Z

Version: 1

|  |  |
| --- | --- |
|  | Backup completion time. |
|  | Backup expiration date, which is set with the ttl setting. The backup is automatically removed from both the OpenShift cluster and the object storage after this date. |
|  | Status of the backup. |
|  | The backup progress, with the number of backed up items so far and the total items to back up. |
|  | Backup start time. |

The backup resource can have the following statuses during its lifetime:

**New**

Initial status when a backup is created. If the backup definition is incorrect, then the backup is aborted and its status changes to FailedValidation. You can review the validationErrors status field for more information about the error.

If the backup definition is valid, then the status changes to InProgress.

**InProgress**

Status when the backup is in progress. During this phase, OADP backs up the resources that are specified in the backup definition and runs the backup hooks.

When OADP uses additional plug-ins, such as OADP Data Mover, the backup enters the WaitingForPluginOperations status. After the plug-in processes are complete, the backup enters the Finalizing status where OADP saves all the remaining backup items, such as backup logs and metadata, to the object storage.

If backing up some resources fails, then the status changes in turn to WaitingForPluginOperationsPartiallyFailed, FinalizingPartiallyFailed, and then PartiallyFailed.

**PartiallyFailed and Failed**

The final status of a backup, with some missing resources because of a backup failure. A partially failed backup can still restore a project, but incompletely. A backup with the Failed status cannot be restored.

You can review the failureReason status field for more information about the error.

**Completed**

Final status of a successfully completed backup. All the data is in the object storage and the backup is ready to use in a restore.

You can initiate a restore by using the following restore definition:

apiVersion: velero.io/v1

kind: Restore

metadata:

name: ***restore-name***

namespace: openshift-adp

spec:

backupName: ***backup-name***

|  |  |
| --- | --- |
|  | Name of the restore resource |
|  | Name of the backup resource to restore |

OADP restores only the resources that are not already in the destination namespace. The destination namespace is the same as in the backup, unless you use the namespaceMapping field to specify a different namespace.

For example, you can use the following restore definition to restore the website project from the previous example to a new website-stage project:

apiVersion: velero.io/v1

kind: Restore

metadata:

name: website-stage

namespace: openshift-adp

spec:

backupName: website

namespaceMapping:

**website: website-stage**

After creating the restore resource, you can review the configuration and monitor the progress of the restore with the oc describe command:

[user@host ~]$ **oc -n openshift-adp describe restore/website-stage**

Name: website-stage

Namespace: openshift-adp

...output omitted...

Spec:

Backup Name: website

Excluded Resources:

nodes

events

events.events.k8s.io

backups.velero.io

restores.velero.io

resticrepositories.velero.io

csinodes.storage.k8s.io

volumeattachments.storage.k8s.io

backuprepositories.velero.io

Item Operation Timeout: 1h0m0s

Namespace Mapping:

Website: website-stage

Status:

Completion Timestamp: 2023-12-12T09:36:23Z

Phase: Completed

Progress:

Items Restored: 7

Total Items: 7

Restore Item Operations Attempted: 1

Restore Item Operations Completed: 1

Start Timestamp: 2023-12-12T09:36:14Z

|  |  |
| --- | --- |
|  | OADP automatically excludes some resources from the restore, such as Kubernetes events and OADP custom resources. You can modify this list with the excludedResources field in the restore definition. |
|  | Restore completion time. |
|  | Status of the restore. |
|  | The restore progress, with the number of restored items so far and the total items to restore. |
|  | Restore start time. |

The possible statuses of the restore are the same as for the backup resource. A successful restore goes through the following statuses during its lifetime:

* New
* InProgress
* WaitingForPluginOperations
* Completed

During the restore, OADP adds two labels to the imported resources: A velero.io/restore-name label with the restore name, and a velero.io/backup-name with the backup name that the resource came from. You can use those labels to identify the resources that were restored from a specific backup or restore name.

For example, you can list the resources with the velero.io/restore-name=website-stage label that are created by the restore from the previous example:

[user@host ~]$ **oc -n website-stage get all \**

**-l velero.io/restore-name=website-stage**

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

service/hugo ClusterIP 172.30.61.176 <none> 8080/TCP,8443/TCP 65m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/hugo 1/1 1 1 65m

NAME TYPE FROM LATEST

buildconfig.build.openshift.io/hugo Source Git 0

NAME HOST/PORT ...

route.route.openshift.io/hugo hugo-website-stage.apps.ocp4.example.com ...

OADP stores in the object storage detailed information about each backup and restore attempt, such as logs, the list of resources that are backed up or restored, and a summary of the errors or warnings that occur during the backup and restore process. This information is not stored in the OADP Kubernetes custom resources, and only limited information such as the runtime status of the backup and restore is available.

The following excerpt shows the object storage layout after the backup and restore from the previous examples:

s3

├── docker

│ └── registry

│ └── v2

│ ├── blobs

│ └── repositories

│ └── website

│ └── hugo

└── oadp

├── backups

│ └── website

│ ├── website-logs.gz

│ ├── website-results.gz

│ └── website.tar.gz

└── restores

└── website-stage

├── restore-website-stage-logs.gz

└── restore-website-stage-results.gz

|  |  |
| --- | --- |
|  | The /docker/registry/v2 path contains a Docker registry with the container images that are included in all backups. |
|  | The /oadp/backups path contains all the information about each backup, including the backed-up Kubernetes resources and the backup logs. Each subdirectory is unique to a single backup attempt and relates to the matching backup resource in the cluster. |
|  | The /oadp/restores path contains all the information about each restore, including the restore logs. Each subdirectory is unique to a single restore attempt and relates to the matching restore resource in the cluster. |

#### **Introducing the Velero Tool**

OADP provides the velero command-line tool that can retrieve backup and restore information from both the object storage and the OpenShift cluster.

The Velero CLI tool is available from the velero deployment in the openshift-adp namespace. You can define an alias to access the velero binary by using the following command:

[user@host ~]$ **alias velero='\**

**oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'**

### Note

The remainder of this section uses the velero alias on the command line to refer to the Velero CLI tool.

The velero command can use the same syntax as the kubectl or oc commands, but is limited to OADP Kubernetes custom resources such as the backup, restore, and schedule resources.

The oc get command displays limited information about OADP resources. However, the velero get command provides the same runtime information from the Kubernetes resources as the oc describe command:

[user@host ~]$ **oc -n openshift-adp get backup,restore**

NAME AGE

backup.velero.io/website 24h

NAME AGE

restore.velero.io/website-stage 4h27m

[user@host ~]$ **velero get backup**

NAME STATUS ERRORS WARNINGS CREATED EXPIRES STORAGE LOCATION SELECTOR

website Completed 0 0 ... 29d oadp-config-1 app=hugo

[user@host ~]$ **velero get restore**

NAME BACKUP STATUS STARTED COMPLETED ERRORS WARNINGS ...

website-stage website Completed ... ... 0 0 ...

The Velero tool provides a describe command that is similar to the oc describe command, and adds a --details option that retrieves additional information about the resource from the object storage:

[user@host ~]$ **velero describe backup website --details**

Name: website

Namespace: openshift-adp

...output omitted...

Phase: Completed

Namespaces:

Included: website

Excluded: <none>

Resources:

Included: imagestreams, buildconfigs, deployments, services, routes

Excluded: <none>

Cluster-scoped: auto

Label selector: app=hugo

...output omitted...

Started: 2023-12-11 14:01:27 +0000 UTC

Completed: 2023-12-11 14:01:42 +0000 UTC

Expiration: 2024-01-10 14:01:27 +0000 UTC

Total items to be backed up: 7

Items backed up: 7

Resource List:

apps/v1/Deployment:

- website/hugo

build.openshift.io/v1/BuildConfig:

- website/hugo

image.openshift.io/v1/ImageStream:

- website/hugo

- website/nginx-122

route.openshift.io/v1/Route:

- website/hugo

v1/Service:

- website/hugo

Velero-Native Snapshots: <none included>

|  |  |
| --- | --- |
|  | Status of the backup. |
|  | Definition of the namespaces and resources to back up. |
|  | Resources that are included in the backup. The --details option adds this information, which comes from the object storage. |

If the backup or restore resource has errors or warnings, then the --details option adds them to the command output.

In the following example, the website project is backed up without filtering by resource type. OADP backs up all resources with the app=hugo label in the website namespace.

apiVersion: velero.io/v1

kind: Backup

metadata:

name: website-label

namespace: openshift-adp

spec:

includedNamespaces:

- website

labelSelector:

matchLabels:

app: hugo

You can use the velero create restore command to restore the previous backup in a new website-dev namespace, as follows:

[user@host ~]$ **velero create restore website-dev \**

**--from-backup=website-label --namespace-mappings=website:website-dev**

Restore request "website-dev" submitted successfully.

Run 'velero restore describe website-dev' or 'velero restore logs website-dev' for more details.

The restore completes with one warning about an existing resource in the target namespace:

[user@host ~]$ **velero describe restore website-dev --details**

Name: website-dev

Namespace: openshift-adp

...output omitted...

Phase: **Completed**

Total items to be restored: 11

Items restored: 11

Started: 2023-12-12 10:21:21 +0000 UTC

Completed: 2023-12-12 10:21:27 +0000 UTC

Warnings:

Velero: <none>

Cluster: <none>

Namespaces:

website-dev: **could not restore, Endpoints "hugo" already exists**. Warning: **the in-cluster version is different than the backed-up version**.

Backup: website-label

...output omitted...

Resource List:

apps/v1/Deployment:

- website-dev/hugo(created)

build.openshift.io/v1/Build:

- website-dev/hugo-1(skipped)

build.openshift.io/v1/BuildConfig:

- website-dev/hugo(created)

discovery.k8s.io/v1/EndpointSlice:

- website-dev/hugo-5hq2x(created)

image.openshift.io/v1/ImageStream:

- website-dev/hugo(skipped)

image.openshift.io/v1/ImageStreamTag:

- website-dev/hugo:latest(skipped)

image.openshift.io/v1/ImageTag:

- website-dev/hugo:latest(skipped)

route.openshift.io/v1/Route:

- website-dev/hugo(created)

v1/Endpoints:

- **website-dev/hugo(failed)**

v1/Namespace:

- website-dev(created)

v1/Service:

- website-dev/hugo(created)

In this example, OADP tries unsuccessfully to restore the hugo endpoint. The hugo service in the backup automatically creates this resource.

Avoid including resources that other resources manage, such as builds, endpoints, or replica sets. It is unnecessary and can cause issues during the restore. You must filter to include in the backup only those resources that your application requires for a successful deployment.

#### **Backing up a Stateful Application with Backup Hooks**

OADP creates crash-consistent backups of your application by using volume snapshots. To back up persistent volumes, you must include the persistentvolumeclaims and persistentvolumes resource types in the backup definition.

OpenShift assigns to each namespace a unique UID and GID that the application pod uses to write data to persistent volumes. If you restore an application to a new namespace, then OpenShift assigns a new set of UIDs and GIDs that prevent the application from accessing its data.

### Note

For more details about user and group ID assignments, refer to the DO180: Red Hat OpenShift Administration I: Operating a Production Cluster training course.

So that OADP can restore the UID and GID, include the namespace resource type in the backup definition. If you use a label selector in the backup definition, then you must add the corresponding label to the namespace. Alternatively, you can use the kubernetes.io/metadata.name label that Kubernetes automatically sets on all namespaces.

To improve the consistency of a backup, you can use backup hooks to specify a list of commands to execute in the application pod before and after the backup is created. You can then use those hooks to quiesce the application and perform an application-consistent backup.

Some applications require additional steps when using volume snapshots to create a usable backup. For more details about the backup procedure, refer to the application documentation. To configure backup hooks, you must specify the target pod (by using labels), the container name, and the commands to run on that container.

You can configure the following hook types according to your needs:

**Pre backup hooks**

A pre backup hook is executed before any other backup action on the pod. If the command fails, then the backup stops immediately with the Failed status.

As an example, you can use this type of hook to quiesce and prepare the application for backup.

**Post backup hooks**

A post backup hook is executed after the backup of the pod and its attached volumes. If the command fails, then the backup stops immediately with the PartiallyFailed status.

As an example, you can use this type of hook to resume or unlock the application after the backup is complete.

**Init restore hooks**

An init restore hook is executed after the pod and its attached volumes are restored, but before any container on that pod starts. The init restore hook defines one or more init containers that follow the same specification as the init container in a pod definition.

OADP does not monitor the status of the init container. Therefore, if the command fails, then the restore continues without any error or warning, but the application pod is in error with the Init:Error status.

As an example, you can use this type of hook to restore a database that the application requires and that is external to the OpenShift cluster.

**Post restore hooks**

A post restore hook is executed when the application pod is restored and running. If the command fails, then the error is logged and the restore continues.

As an example, you can use this type of hook to run an integrity check on the restored database.

The following example is a backup definition for a MongoDB database. It uses backup hooks to flush all pending writes to the disk and to lock the database to prevent any writes during the backup.

apiVersion: velero.io/v1

kind: Backup

metadata:

name: mongodb

namespace: openshift-adp

spec:

includedNamespaces:

- mongodb

orLabelSelectors:

- matchLabels:

app: mongodb

- matchLabels:

kubernetes.io/metadata.name: mongodb

includedResources:

- deployments

- services

- secret

- pvc

- pv

- pods

- namespace

hooks:

resources:

- name: mongodb-lock

labelSelector:

matchLabels:

app: mongodb

pre:

- exec:

container: mongodb

command:

- /usr/bin/mongosh

- --eval

- db.fsyncLock();

post:

- exec:

container: mongodb

command:

- /usr/bin/mongosh

- --eval

- db.fsyncUnlock();

|  |  |
| --- | --- |
|  | Resources with the app: mongodb or kubernetes.io/metadata.name: mongodb label are included in the backup. |
|  | The pvc and pv resource types must be specified in the includedResources key to back up the application volume. The pods resource type must also be specified for the backup hooks to be executed. The namespace resource type must be specified to preserve the UID and GID that are used in the application volume. |
|  | The hook runs on all pods that match the label selector. |
|  | The pre backup hook executes the db.fsyncLock() MongoDB command in the mongodb container to lock the database before the volume snapshot. |
|  | The post backup hook executes the db.fsyncUnlock() MongoDB command to unlock the database after the backup is completed. |

### Important

Because backup and restore hooks are executed only on pods that are included in the backup, you must include the pod resource type in the backup.

If the hooks are configured to run on a pod that is not included in the backup, then the hooks are ignored without any error or warning.

The following restore resource definition restores the MongoDB backup from the previous example:

apiVersion: velero.io/v1

kind: Restore

metadata:

name: mongodb

namespace: openshift-adp

spec:

backupName: mongodb

hooks:

resources:

- name: mongodb-unlock

labelSelector:

matchLabels:

app: mongodb

postHooks:

- init:

initContainers:

- name: remove-lock

image: mongodb/mongodb-community-server:7.0-ubi8

volumeMounts:

- name: mongodb-data

mountPath: /data/db

command:

- /usr/bin/rm

- /data/db/mongod.lock

|  |  |
| --- | --- |
|  | The init restore hook removes the database lock from the backup before the database starts. |

OADP Data Mover stores the volume backup in the object storage inside a Kopia unified repository.

A Kopia unified repository is a backup repository that Kopia manages for storing volume backups. Kopia encrypts, deduplicates, and compresses the data in the backup repository. OADP uses a dedicated backup repository for each namespace to store all volume backups for that namespace. OADP uses this unified repository to store backups from both volume snapshots and file-system backups.

When the first backup of a namespace is created, OADP initializes a Kopia repository in the object storage, and creates a matching BackupRepository resource in the openshift-adp namespace.

OADP uses a BackupRepository custom resource to store information about the backup repository such as the repository encryption keys and object storage information.

The following excerpt shows the object storage layout after the previous backup and restore examples:

s3

└── oadp

├── backups

│ └── mongodb

│ ├── mongodb-logs.gz

│ └── ...

├── kopia

│ └── mongodb

│ ├── kopia.repository

│ └── ...

└── restores

└── mongodb

├── restore-mongodb-logs.gz

└── ...

|  |  |
| --- | --- |
|  | The /oadp/kopia path contains the backup repositories with data from volume backups. Each subdirectory is a unique Kopia repository for a single namespace in the cluster and contains all backups from all volumes in that namespace. |

### **Scheduling a Recurring Backup**

You can back up an application at a recurring interval by using the schedule resource. A schedule resource is similar to a cron job resource. A schedule resource requires a schedule in Cron format and a definition template of the backup resource to create, at the specified time.

The following example is a schedule definition of a daily backup for the previous static website example:

apiVersion: velero.io/v1

kind: Schedule

metadata:

name: website-daily

namespace: openshift-adp

labels:

app: hugo

spec:

schedule: "0 7 \* \* \*"

paused: false

template:

includedNamespaces:

- website

labelSelector:

matchLabels:

app: hugo

includedResources:

- imagestreams

- buildconfigs

- deployments

- services

- routes

ttl: 720h0m0s

|  |  |
| --- | --- |
|  | The labels that you set on the schedule resource are automatically copied to the backup resources that the schedule creates. |
|  | Specifies the schedule for the job in Cron format. |
|  | You can disable the schedule by setting the paused parameter to true. |
|  | Sets the backup definition template by using the same settings as in a backup resource. |

See the references section for more information about the schedule API definition.

To get detailed information about a schedule, use the velero get schedule command:

[user@host ~]$ **velero get schedule**

NAME STATUS CREATED SCHEDULE BACKUP TTL LAST BACKUP SELECTOR PAUSED

website-daily Enabled ... 0 7 \* \* \* 720h0m0s ... app=hugo false

The status of an activated schedule is Enabled. If the schedule is disabled with the paused parameter, then its status is New. The LAST BACKUP column shows the time of the latest backup that the schedule created.

With the Velero tool, you can create a one-time backup by using the same definition as in an existing schedule. To back up an application on demand, you can create a disabled schedule as a backup template. You can then start a new backup with this schedule when you need it.

In this example, you are creating a pre-upgrade-1.1 backup from the website-daily schedule:

[user@host ~]$ **velero create backup *pre-upgrade-1.1* \**

**--from-schedule website-daily**

Creating backup from schedule, all other filters are ignored.

Backup request "***pre-upgrade-1.1***" submitted successfully.

...output omitted...

Backups that are created from a schedule inherit the schedule's labels. In addition, the velero.io/schedule-name label is set on the backup resources with the schedule name. You can use those labels to identify the schedules and backups for your application on the openshift-adp namespace:

[user@host ~]$ **velero get backup -l app=hugo**

NAME STATUS ERRORS WARNINGS ...

post-upgrade-1.1 Finalizing 0 0 ...

pre-upgrade-1.1 Completed 0 0 ...

website-daily-20231215100856 Completed 0 0 ...

website-daily-20231215091728 Completed 0 0 ...

Because the openshift-adp namespace contains the backup and restore resources for all applications that are running on the cluster, it is important to use labels to identify OADP resources that relate to your application.

### **Cleaning Backups**

Because OADP synchronizes backup definitions with the object storage, OADP automatically re-creates any backup that you delete with the oc command. To permanently delete a backup, you must delete it from the object storage.

Use the velero command to delete backup and restore information from the object storage, and all the associated resources from the cluster:

[user@host ~]$ **velero delete backup *backup-name***

Are you sure you want to continue (Y/N)? **y**

Request to delete backup "***backup-name***" submitted successfully.

The backup will be fully deleted after all associated data (disk snapshots, backup files, restores) are removed.

The backup status changes to Deleting, and OADP removes all restore resources that are attached to this backup from both the OpenShift cluster and the object storage. Then the backup itself is removed from the cluster and the object storage.

You can instruct OADP to delete a backup automatically after a specified elapsed time by using the TTL (Time To Live) setting on the backup and schedule definition. By default, OADP deletes backups after 30 days. The minimum lifetime of a backup is 1 hour.

### Important

OADP does not immediately delete data from the backup repository on the object storage. OADP occasionally runs repository maintenance, such as to delete data that backups no longer need. OADP might start deleting data from the object storage up to 24 hours after a backup is deleted.

### **Backing up Volumes with File System Backup**

With the File System Backup (FSB) feature, you can back up volumes that are not compatible with snapshots.

OADP uses the following process to back up OpenShift applications with FSB:

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The administrator creates a backup resource in the openshift-adp namespace that triggers the backup process. |
|  | Velero exports all Kubernetes resources from the application namespace to the backup storage location. |
|  | The node-agent daemon set, which runs on the same node as the application pod, exports the volume data from the volume mount point on the cluster node to the backup storage location. |

### Important

OADP copies the data from the application file system while the application is still running. You must use backup hooks to ensure that the application data is not altered during the backup process.

The FSB restore process is as follows:

A diagram of a software cluster

Description automatically generated

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The administrator creates a restore resource in the openshift-adp namespace, which triggers the restore process. |
|  | Velero imports the Kubernetes resources from the backup storage location to the application namespace. |
|  | The node-agent daemon set, which runs on the same node as the application pod, imports the application data from the backup storage location to the mount point for the application volume on the cluster node. |

You can instruct OADP to back up all volumes in a backup definition with FSB by using the defaultVolumesToFsBackup option:

apiVersion: velero.io/v1

kind: Backup

metadata:

name: <backup-name>

namespace: openshift-adp

spec:

**defaultVolumesToFsBackup: true**

You can also annotate the application pod to specify which volumes to back up with FSB by using the backup.velero.io/backup-volumes annotation. If your application uses multiple volumes from different storage classes, then OADP backs up the volumes in the annotation with FSB, and all other volumes with snapshots.

In the following example, the application to back up is an Nginx web server that serves a website from an Amazon Elastic File System (AWS EFS). Because the AWS EFS CSI driver does not support volume snapshots, you must back up the volume with FSB.

To enable FSB for the wwwdata volume, you must set the backup.velero.io/backup-volumes annotation in the deployment, as follows:

apiVersion: apps/v1

kind: Deployment

metadata:

name: website-nginx

namespace: org-website

spec:

template:

metadata:

annotations:

**backup.velero.io/backup-volumes: wwwdata**

spec:

containers:

name: web

image: registry.access.redhat.com/ubi9/nginx-120

...output omitted...

volumeMounts:

- mountPath: /opt/app-root/src

name: wwwdata

...output omitted...

volumes:

- name: **wwwdata**

persistentVolumeClaim:

claimName: nginx-wwwdata

...output omitted...

|  |  |
| --- | --- |
|  | List of volumes to back up with FSB. You must use the same volume name that is defined in the pod definition. |

### Note

You can specify multiple volumes with the backup.velero.io/backup-volumes annotation, with a comma to separate each volume. For example, backup.velero.io/backup-volumes: volume1,volume2,volume3

### Important

Only volumes that are compatible with volume snapshots are included in the backup by default. You must enable FSB to include volumes in your backup that are not compatible with volume snapshots.

OADP stores file-system backups in the same unified backup repository on the object storage that Data Mover uses for volume snapshots.

### **Troubleshooting Backups and Restores**

If a backup or restore fails, you can get more information about the failure with the velero command:

[user@host ~]$ **velero get backup *mybackup***

NAME STATUS ERRORS WARNINGS ...

mybackup **PartiallyFailed** 1 0 ...

Use the velero describe command for more details about the errors and warnings:

[user@host ~]$ **velero describe backup *mybackup***

Name: mybackup

Namespace: openshift-adp

Labels: velero.io/storage-location=oadp-backup-1

Annotations: velero.io/source-cluster-k8s-gitversion=v1.25.7+eab9cc9

velero.io/source-cluster-k8s-major-version=1

velero.io/source-cluster-k8s-minor-version=25

Phase: **PartiallyFailed** (run `velero backup logs mybackup` for more information)

Errors:

Velero: <none>

Cluster: <none>

Namespaces:

database: **resource: /pods name: /mariadb-757c5bdc88-mrwhb error: /command terminated with exit code 1**

...output omitted...

You can view the backup logs with the velero backup logs command:

[user@host ~]$ **velero backup logs *mybackup***

...output omitted...

Because the logs are stored in the object storage, you can also download them with the s3cmd command, and view them with any text editor.

The logs are stored in the s3://*bucket-name*/oadp/backups/*backup-name*/*backup-name*-logs.gz path for backups and in the s3://*bucket-name*/oadp/restores/*restore-name*/*restore-name*-logs.gz path for restores.

You can use the following command to download the log file to the current directory with s3cmd:

[user@host ~]$ **s3cmd get \**

**s3://*backup-bucket*/oadp/backups/*mybackup*/*mybackup*-logs.gz**

download: 's3://***backup-bucket***/oadp/backups/***mybackup***/***mybackup***-logs.gz' ...

7454 of 7454 100% in 0s 96.14 KB/s done

For a backup that uses hooks, you can search the log with the hookPhase keyword to review the status of the hooks. The standard and error outputs of the hook command are recorded in the stdout and stderr lines, respectively.

[user@host ~]$ **velero backup logs *mybackup* | grep hookPhase**

... msg="running exec hook" hookCommand="[/bin/bash -c mariadb -u root -e \"set global read\_only=1;flush tables\"]" ... hookPhase=pre

... msg="stdout: " ... hookPhase=pre

... msg=**"stderr: ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)\n"** ... hookPhase=pre

... msg="Error executing hook" **error="command terminated with exit code 1"** ... hookPhase=pre

In the previous example, the backup hook was misconfigured and is missing the password to connect to the database.

### References

For more information about backing up applications with OADP, refer to the OADP Backing Up section in the OADP Application Backup and Restore chapter in the Red Hat OpenShift Container Platform 4.14 Backup and Restore documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/backup_and_restore/index#oadp-backing-up>

For more information about troubleshooting OADP, refer to the Troubleshooting section in the OADP Application Backup and Restore chapter in the Red Hat OpenShift Container Platform 4.14 Backup and Restore documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/backup_and_restore/index#troubleshooting>

[Velero API Definitions](https://velero.io/docs/v1.12/api-types/)

## **Guided Exercise: Backup and Restore with OADP**

Perform a crash-consistent and an application-consistent backup of a database, and restore each backup to a different project.

**Outcomes**

* Back up and restore an OpenShift project.
* Schedule a backup of an OpenShift project.
* Remove previous backups.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start backup-restore**

**Instructions**

Create a crash-consistent backup of the database project by using OADP and CSI snapshots, and restore it to a new database-crash project.

Schedule a weekly backup of the database project. Use hooks to quiesce the database and produce an application-consistent backup.

Trigger a backup from the schedule and restore the application-consistent backup to a new database-backup project.

Remove the backups and all related resources.

1. As the admin user, review the content of the database project.
   1. Log in to your OpenShift cluster as the admin user.

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

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

Login successful.

...output omitted...

* 1. Change to the database project.

[student@workstation ~]$ **oc project database**

Now using project "database" on server "https://api.ocp4.example.com:6443".

* 1. List the resources with the app=mariadb label in the database project.

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

**pvc,svc,deployment,secret,configmap -l app=mariadb**

NAME ... STORAGECLASS

persistentvolumeclaim/mariadb ... ocs-external-storagecluster-ceph-rbd

NAME TYPE ... PORT(S)

service/mariadb LoadBalancer ... 3306:31402/TCP

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/mariadb 1/1 1 1 5m51s

NAME TYPE DATA AGE

secret/mariadb Opaque 4 5m54s

NAME DATA AGE

configmap/database 1 5m53s

* 1. Review the UID and GID assignment for the database namespace. The values might be different in your environment.

[student@workstation ~]$ **oc get namespace database -o yaml**

...output omitted...

metadata:

annotations:

...output omitted...

openshift.io/sa.scc.supplemental-groups: **1000810000/10000**

openshift.io/sa.scc.uid-range: **1000810000/10000**

labels:

kubernetes.io/metadata.name: database

...output omitted...

* 1. Use the DO380/labs/backup-restore/view-db.sh script to connect to the MariaDB database in the mariadb deployment, and list the rows in the application\_logs table from the application database.

[student@workstation ~]$ **DO380/labs/backup-restore/view-db.sh**

+ oc exec -c mariadb deploy/mariadb -- bash -c 'mariadb -u ${MARIADB\_USER} \

-p"${MARIADB\_PASSWORD}" \

${MARIADB\_DATABASE} -te \

"SELECT \* FROM application\_logs ORDER BY id DESC LIMIT 5;"'

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

| id | time | app | message |

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

| 25 | 2023-12-05 10:48:12 | db-loader | ad42...785d |

| 24 | 2023-12-05 10:48:02 | db-loader | 26b3...be3e |

| 23 | 2023-12-05 10:47:52 | db-loader | ed74...6ed8 |

| 22 | 2023-12-05 10:47:42 | db-loader | f9f5...3270 |

| 21 | 2023-12-05 10:47:32 | db-loader | 2c65...b85c |

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

### Note

The db-loader application is deployed in the application project. This application creates a record in the application\_logs table every 10 seconds to simulate database activity during the backup operations.

1. Create a crash-consistent backup of the database project.
   1. Change to the ~/DO380/labs/backup-restore directory.

[student@workstation ~]$ **cd ~/DO380/labs/backup-restore**

[student@workstation backup-restore]$

* 1. Create a db-manual backup resource in the openshift-adp namespace to back up the database project. Use resource filtering to back up only the listed MariaDB resources in the previous step.

Modify the partial resource definition in the ~/DO380/labs/backup-restore/backup-db-manual.yml file as follows:

apiVersion: velero.io/v1

kind: Backup

metadata:

name: **db-manual**

namespace: **openshift-adp**

spec:

includedNamespaces:

- **database**

orLabelSelectors:

- matchLabels:

**app: mariadb**

- matchLabels:

**kubernetes.io/metadata.name: database**

includedResources:

- **namespace**

- **deployments**

- **configmaps**

- **secrets**

- **pvc**

- **pv**

- **services**

### Important

So that OADP can restore the UID and GID assigned to the database namespace, include the namespace resource type and the kubernetes.io/metadata.name: database label in the backup definition.

* 1. Apply the configuration for the backup resource.

[student@workstation backup-restore]$ **oc apply -f backup-db-manual.yml**

backup.velero.io/db-manual created

1. Review the status of the backup.
   1. Create an alias to access the velero binary from the Velero deployment in the openshift-adp namespace.

[student@workstation backup-restore]$ **alias velero='\**

**oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'**

* 1. Use the velero command to get the status of the db-manual backup. Monitor the output to verify that the backup status is Completed. The backup process takes several minutes. Note the backup creation time for the next step.

[student@workstation backup-restore]$ **velero get backup db-manual**

NAME STATUS ERRORS WARNINGS CREATED ...

db-manual **Completed** 0 0 ***2023-12-04 12:16:40*** ...

* 1. Use the view-db.sh script to review the content of the application\_logs table from the application database in the mariadb deployment. Use the backup creation time from the previous step as a script argument to list the rows that were created before the backup time. You use the result to compare with the database that you restore in a later step.

[student@workstation backup-restore]$ **./view-db.sh "*2023-12-04 12:16:40*"**

+ oc exec -c mariadb deploy/mariadb -- bash -c 'mariadb -u ${MARIADB\_USER} -p"${MARIADB\_PASSWORD}" ${MARIADB\_DATABASE} -te "SELECT \* FROM application\_logs WHERE time <= \"***2023-12-04 12:16:40***\" ORDER BY id DESC LIMIT 5;"'

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

| id | time | app | message |

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

| 36 | 2023-12-05 10:50:03 | db-loader | 2232...4341 |

| 35 | 2023-12-05 10:49:53 | db-loader | b8a2...6ddf |

| 34 | 2023-12-05 10:49:43 | db-loader | 9cb1...a6d1 |

| 33 | 2023-12-05 10:49:33 | db-loader | c0e5...5e8b |

| 32 | 2023-12-05 10:49:23 | db-loader | e3c9...1929 |

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

1. Restore the db-manual backup to a new database-crash project.
   1. Create a db-crash restore resource in the openshift-adp namespace to restore the db-manual backup. Restore the backup to the database-crash project.

Modify the partial resource definition in the ~/DO380/labs/backup-restore/restore-db-crash.yml file as follows:

apiVersion: velero.io/v1

kind: Restore

metadata:

name: **db-crash**

namespace: **openshift-adp**

spec:

backupName: **db-manual**

restorePVs: true

namespaceMapping:

**database: database-crash**

* 1. Apply the configuration for the restore resource.
  2. [student@workstation backup-restore]$ **oc apply -f restore-db-crash.yml**

restore.velero.io/db-crash created

* 1. Use the velero command to get the status of the db-crash restore resource. Monitor the output to verify that the backup status is Completed. The restoration process takes several minutes.
  2. [student@workstation backup-restore]$ **velero get restore db-crash**
  3. NAME BACKUP STATUS ... ERRORS WARNINGS

db-crash db-manual **Completed** ... 0 0

1. Review the content of the database-crash project and verify the database integrity.
   1. Change to the database-crash project.
   2. [student@workstation backup-restore]$ **oc project database-crash**

Now using project "database-crash" on server "https://api.ocp4.example.com:6443".

* 1. List the resources in the database-crash project and compare them with the resources in the database project from the previous step.

[student@workstation backup-restore]$ **oc get \**

**pvc,svc,deployment,secret,configmap -l app=mariadb**

NAME ... STORAGECLASS

persistentvolumeclaim/mariadb ... ocs-external-storagecluster-ceph-rbd

NAME TYPE ... PORT(S)

service/mariadb LoadBalancer ... 3306:31402/TCP

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/mariadb 1/1 1 1 5m51s

NAME TYPE DATA AGE

secret/mariadb Opaque 4 5m54s

NAME DATA AGE

configmap/database 1 5m53s

* 1. Review the UID and GID assignment for the database-crash namespace. Verify that the numbers are the same as for the database namespace from the previous step.

[student@workstation ~]$ **oc get namespace database-crash -o yaml**

...output omitted...

metadata:

annotations:

...output omitted...

openshift.io/sa.scc.supplemental-groups: **1000810000/10000**

openshift.io/sa.scc.uid-range: **1000810000/10000**

labels:

kubernetes.io/metadata.name: database-crash

...output omitted...

* 1. Use the mariadb-check command to check the integrity and to repair the database tables.

[student@workstation backup-restore]$ **oc exec -c mariadb deploy/mariadb -- \**

**bash -c \**

**'mariadb-check -u root -p"${MARIADB\_ROOT\_PASSWORD}" -A --auto-repair'**

application.application\_logs

warning : 1 client is using or hasn't closed the table properly

warning : Size of datafile is: 13296 Should be: 13120

error : Wrong bytesec: 0-0-0 at linkstart: 13120

**error : Corrupt**

mysql.column\_stats OK

...output omitted...

Repairing tables

application.application\_logs

info : Wrong bytesec: 0- 0- 0 at 13120; Skipped

status : OK

### Note

In this example, the application\_logs table is corrupted and some data might be lost after the repair. The result might be different in your lab environment.

* 1. Use the view-db.sh script to list the records in the application\_logs table from the application database in the database-crash project. Compare the result with the content of the database in the database project from the previous step.

[student@workstation backup-restore]$ **./view-db.sh**

+ oc exec -c mariadb deploy/mariadb -- bash -c 'mariadb -u ${MARIADB\_USER} \

-p"${MARIADB\_PASSWORD}" \

${MARIADB\_DATABASE} -te \

"SELECT \* FROM application\_logs ORDER BY id DESC LIMIT 5;"'

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

| id | time | app | message |

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

| 35 | 2023-12-05 10:49:53 | db-loader | b8a2...6ddf |

| 34 | 2023-12-05 10:49:43 | db-loader | 9cb1...a6d1 |

| 33 | 2023-12-05 10:49:33 | db-loader | c0e5...5e8b |

| 32 | 2023-12-05 10:49:23 | db-loader | e3c9...1929 |

| 31 | 2023-12-05 10:49:13 | db-loader | dc45...b7d8 |

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

### Note

Compare the latest row ID with the ID from the previous step. Rows can be missing. This outcome can result from the database corruption that was detected in the previous step, or because the records were not written to the disk before the backup started.

1. Delete the crash-consistent backup and all related resources.
   1. Review the content of the S3 bucket by using the s3cmd command.

[student@workstation backup-restore]$ **s3cmd la -r**

...output omitted...

... s3://backup-...**/oadp/backups/db-manual**/velero-backup.json

... s3://backup-.../oadp/backups/db-manual/....gz

...output omitted...

... s3://backup-...**/oadp/kopia/database**/kopia.repository

... s3://backup-.../oadp/kopia/database/551...c2b

...output omitted...

... s3://backup-...**/oadp/restores/db-crash**/restore-db-crash-logs.gz

... s3://backup-.../oadp/restores/db-crash/....gz

...output omitted...

|  |  |
| --- | --- |
|  | The /oadp/backups/*backup\_name* path contains the backup of the Kubernetes resources and the logs of the backup. |
|  | The /oadp/kopia/*project\_name* path is the Kopia backup repository that contains backups of all data volumes in a project. |
|  | The /oadp/restores/*restore\_name* path contains the restore logs. Note The database uses a container image that is stored on an external registry and is therefore not included in the backup. |

* 1. Delete the db-manual backup by using the velero command.

[student@workstation backup-restore]$ **velero delete backup db-manual**

Are you sure you want to continue (Y/N)? **y**

Request to delete backup "db-manual" submitted successfully.

The backup will be fully deleted after all associated data (disk snapshots, backup files, restores) are removed.

### Note

Velero deletes all backup and restore resources that are associated with the specified backup in the OpenShift cluster. Velero also deletes the backup and restore files from the object storage.

The Kopia backup repository is never deleted, even if all backups from the project are removed. Kopia removes expired and unused volume backups from the object storage during the repository maintenance which happens occasionally.

* 1. Monitor the status of the db-manual backup with the velero command and wait until OADP deletes the backup.

[student@workstation backup-restore]$ **velero get backup db-manual**

NAME STATUS ERRORS WARNINGS ...

db-manual **Deleting** 0 0 ...

### Note

If the command returns the following error, then the backup is removed from both OpenShift and the object storage:

An error occurred: backups.velero.io "db-manual" not found

command terminated with exit code 1

* 1. Verify that the backup and restore resources are removed from the openshift-adp namespace. It can take a few minutes for the OADP operator to remove the resources.

[student@workstation backup-restore]$ **oc -n openshift-adp get backup,restore**

No resources found in openshift-adp namespace.

* 1. Review the content of the S3 bucket. The db-manual backup and restore are automatically deleted and only the Kopia backup repository for the database project remains.

[student@workstation backup-restore]$ **s3cmd la -r**

... s3://backup-.../oadp/kopia/database/\_log\_2024...5f5

... s3://backup-.../oadp/kopia/database/kopia.repository

... s3://backup-.../oadp/kopia/database/551...c2b

...output omitted...

1. Schedule a recurring application-consistent backup of the database project.

Use backup hooks to ensure that all database operations are flushed to the file system and that the database is locked in read-only mode during the backup.

Backups must be automatically deleted after 15 days.

* 1. Create a db-backup schedule resource in the openshift-adp namespace to back up the database project every week.

Set the schedule to 7 AM every Sunday. For this exercise, disable the schedule with the paused field to prevent unexpected backups from starting during the hands-on activity. You manually trigger a backup from this schedule in a later step.

Modify the partial resource definition in the ~/DO380/labs/backup-restore/schedule-db-backup.yml file as follows:

apiVersion: velero.io/v1

kind: Schedule

metadata:

name: **db-backup**

namespace: **openshift-adp**

spec:

schedule: **"0 7 \* \* 0"**

paused: true

template:

ttl: **360h0m0s**

includedNamespaces:

- **database**

orLabelSelectors:

- matchLabels:

**app: mariadb**

- matchLabels:

**kubernetes.io/metadata.name: database**

includedResources:

- **namespace**

- **deployments**

- **configmaps**

- **secrets**

- **pvc**

- **pv**

- **services**

- **pods**

hooks:

resources:

- name: mariadb-readonly

pre:

- exec:

container: mariadb

command:

- /bin/bash

- -c

- |

**mariadb -u "root" \**

**-p"${MARIADB\_ROOT\_PASSWORD}" \**

**${MARIADB\_DATABASE} -e \**

**"set global read\_only=1; \**

**BACKUP STAGE START; \**

**BACKUP STAGE BLOCK\_COMMIT;";**

**sync;**

post:

- exec:

container: mariadb

command:

- /bin/bash

- -c

- |

**mariadb -u "root" \**

**-p"${MARIADB\_ROOT\_PASSWORD}" \**

**${MARIADB\_DATABASE} -e \**

**"set global read\_only=0;";**

|  |  |
| --- | --- |
|  | The set global read\_only=1 command locks the database in read-only mode to prevent any writes during the backup. |
|  | The BACKUP STAGE commands ensure that all database tables are flushed to the disk. |
|  | The sync command ensures that all cached writes are flushed to the persistent storage. |
|  | The set global read\_only=0 command removes the database lock after the backup is completed. |

### Warning

Because backup hooks are executed only on pods that are included in the backup, you must include the pod resource in the includedResources field.

* 1. Apply the configuration for the schedule resource.

[student@workstation backup-restore]$ **oc apply -f schedule-db-backup.yml**

schedule.velero.io/db-backup created

* 1. Verify the status of the db-backup schedule.

[student@workstation backup-restore]$ **velero get schedule**

NAME STATUS ... SCHEDULE BACKUP TTL LAST BACKUP SELECTOR PAUSED

db-backup New ... 0 7 \* \* 0 360h0m0s n/a <none> true

* 1. Use the velero command to trigger a new backup from the db-backup schedule. Save the backup name for a later step.

[student@workstation backup-restore]$ **velero create backup \**

**--from-schedule=db-backup**

...output omitted...

Creating backup from schedule, all other filters are ignored.

Backup request "***db-backup-20231205114600***" submitted successfully.

...output omitted...

* 1. Use the velero command to monitor the status of the backup and wait until the backup is in the Completed state. The backup process takes several minutes. Note the creation time of the backup from the CREATED field.

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

[student@workstation backup-restore]$ **watch -n 10 oc -n openshift-adp \**

**exec deployment/velero -c velero -it -- ./velero get backup \**

**-l velero.io/schedule-name=db-backup**

NAME STATUS ERRORS WARNINGS CREATED ...

**db-backup-2023...00** **Completed** 0 0 ***2023-12-05 11:46:00*** ...

* 1. Use the view-db.sh script to identify the rows immediately before the backup time in the application\_logs table from the application database in the mariadb deployment. Use the date and time from the previous step as argument of the script.

[student@workstation backup-restore]$ **./view-db.sh -n database "*2023-12-05 11:46:00*"**

+ oc exec -n database -c mariadb deploy/mariadb -- bash -c 'mariadb -u ${MARIADB\_USER} -p"${MARIADB\_PASSWORD}" ${MARIADB\_DATABASE} -te "SELECT \* FROM application\_logs WHERE time <= \"***2023-12-05 11:46:00***\" ORDER BY id DESC LIMIT 5;"'

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

| id | time | app | message |

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

| 371 | 2023-12-05 11:45:58 | db-loader | 325ef...f337 |

| 370 | 2023-12-05 11:45:48 | db-loader | 2505f...3215 |

| 369 | 2023-12-05 11:45:38 | db-loader | 77ba0...561a |

| 368 | 2023-12-05 11:45:27 | db-loader | 6bffb...8f5f |

| 367 | 2023-12-05 11:45:17 | db-loader | a895d...3dfd |

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

1. Restore the application-consistent backup to the database-backup project.
   1. Create a db-backup restore resource in the openshift-adp namespace. Restore the backup from the previous step to the database-backup project.

Modify the partial resource definition in the ~/DO380/labs/backup-restore/restore-db-backup.yml file as follows:

apiVersion: velero.io/v1

kind: Restore

metadata:

name: **db-backup**

namespace: **openshift-adp**

spec:

backupName: ***db-backup-20231205114600***

restorePVs: true

namespaceMapping:

**database: database-backup**

* 1. Apply the configuration for the restore resource.

[student@workstation backup-restore]$ **oc apply -f restore-db-backup.yml**

* 1. Use the velero command to get the status of the db-backup restore resource. Monitor the output to verify that the status is Completed. The restoration process takes several minutes.

[student@workstation backup-restore]$ **velero get restore db-backup**

NAME BACKUP STATUS ... ERRORS WARNINGS

db-backup db-backup-20231205114600 **Completed** ... 0 0

1. Review the content of the database-backup project.
   1. Change to the database-backup project.

[student@workstation backup-restore]$ **oc project database-backup**

Now using project "database-backup" on server "https://api.ocp4.example.com:6443".

* 1. Use the mariadb-check command to verify the integrity of the database tables.

[student@workstation backup-restore]$ **oc exec -c mariadb deploy/mariadb -- \**

**bash -c 'mariadb-check -u root -p"${MARIADB\_ROOT\_PASSWORD}" -A'**

application.application\_logs **OK**

mysql.column\_stats OK

...output omitted...

mysql.transaction\_registry OK

### Note

All tables are OK and no corruption is detected.

* 1. Use the view-db.sh script to list the rows in the application\_logs table from the application database in the database-backup project. Compare the result with the content of the database in the database project from the previous step.

[student@workstation backup-restore]$ **./view-db.sh**

+ oc exec -c mariadb deploy/mariadb -- bash -c 'mariadb -u ${MARIADB\_USER} \

-p"${MARIADB\_PASSWORD}" \

${MARIADB\_DATABASE} -te \

"SELECT \* FROM application\_logs ORDER BY id DESC LIMIT 5;"'

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

| id | time | app | message |

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

| 371 | 2023-12-05 11:45:58 | db-loader | 325e...f337 |

| 370 | 2023-12-05 11:45:48 | db-loader | 2505...3215 |

| 369 | 2023-12-05 11:45:38 | db-loader | 77ba...561a |

| 368 | 2023-12-05 11:45:27 | db-loader | 6bff...8f5f |

| 367 | 2023-12-05 11:45:17 | db-loader | a895...3dfd |

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

### Note

Compare the latest row ID with the ID from the previous step. All rows until the start of the backup are present.

1. Clean up the resources.
   1. Change to the student HOME directory.

[student@workstation backup-restore]$ **cd**

[student@workstation ~]$

* 1. Delete the database-crash and database-backup projects.

[student@workstation ~]$ **oc delete project database-crash database-backup**

project.project.openshift.io "database-crash" deleted

project.project.openshift.io "database-backup" deleted

* 1. Delete the backup that you created in the previous step by using the velero command. Use the velero.io/schedule-name=db-backup label to delete all backups that were created from the db-backup schedule.

[student@workstation backup-restore]$ **velero delete backup \**

**-l velero.io/schedule-name=db-backup**

Are you sure you want to continue (Y/N)? **y**

Request to delete backup "db-backup-20231204141509" submitted successfully.

The backup will be fully deleted after all associated data (disk snapshots, backup files, restores) are removed.

**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 backup-restore**

## **Lab: Backup, Restore, and Migration of Applications with OADP**

Deploy the OADP operator and describe its features and dependencies.

Configure one-time and scheduled backups with OADP and restore from them.

**Outcomes**

* Configure the OpenShift API for Data Protection operator to back up OpenShift applications.
* Schedule a complete backup of an application.
* Create and restore an application-consistent backup by using backup hooks.
* Clean up previous backups.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Your team deployed a wiki application on OpenShift to host its operational documentation. You are asked to configure a scheduled daily backup by using OADP.

The application is deployed in the wiki project and is accessible at the following URL: https://mediawiki-wiki.apps.ocp4.example.com. The application uses the MediaWiki software and a PostgreSQL database with two volumes: uploaded images and documents are stored on a CephFS volume, and the database uses a Ceph RBD volume.

A colleague started installing OADP and performed the following tasks:

* Install the required operators for OADP.
* Create the S3 bucket and configure the s3cmd command with the S3 credentials.
* Create partial OADP configuration files in the ~/DO380/labs/backup-review path with the S3 information and credentials.

Review and complete the configuration of OADP. Ensure that all storage classes that the application is using can be backed up with CSI snapshots. Configure OADP to store volume backups on the S3 object storage.

Configure a scheduled daily backup of the wiki project. The backup must start at 11 PM every day. To ensure backup consistency, you must lock the application to prevent any writes during the backup, and prepare the database for a snapshot before the backup by using backup hooks.

All Kubernetes resources required for the application have the app.kubernetes.io/part-of=mediawiki label. Resources specific to the MediaWiki component have the app.kubernetes.io/name: mediawiki label. Resources specific to the PostgreSQL database have the app.kubernetes.io/name: postgresql label. The backup must include only the resources with those labels.

To lock the MediaWiki application in read-only mode, create a /data/images/lock\_yBgMBwiR file in the mediawiki container. This lock file must contain a single line with the "backup in progress" string to display the lock to the users in the application. To unlock the application after the backup is complete, remove the lock file.

### Note

The /data/images path is in the mediawiki volume. The lock file is therefore included in the backup.

To prepare the PostgreSQL database for a snapshot and to flush all in-memory transactions to the disk, use the psql -c "CHECKPOINT;" command in the postgresql container.

To prepare for an upcoming update of the software stack, you are asked to restore an exact copy of the wiki application to a new wiki-staging project, so your team can test the update in a staging environment first.

Trigger an immediate backup from the scheduled backup and restore it to the wiki-staging project. Ensure that the application is unlocked after the restore by deleting the MediaWiki lock file with a restore hook.

To distinguish the production environment from the staging environment, change the MediaWiki site name to DO380 Team Wiki Staging. This setting is controlled by the MEDIAWIKI\_SITE\_NAME environment variable on the mediawiki deployment.

Configure the MediaWiki URL on the mediawiki deployment with the MEDIAWIKI\_SITE\_SERVER environment variable to match the route URL in the wiki-staging project.

Finally, remove all manual backup and restore resources from OpenShift and from the object storage.

### Note

The S3 object storage in the lab environment uses a custom certificate that is signed with the OpenShift service CA. If you use the velero describe --​details or velero logs commands, you must specify the CA certificate with the --cacert parameter.

The CA certificate for the OpenShift service CA is available from the velero pod in the /run/secrets/kubernetes.io/serviceaccount/service-ca.crt path.

1. Complete the configuration files in the ~/DO380/labs/backup-review path and configure OADP with the following requirements:
   * Ensure that OADP can back up volumes with CSI snapshots.
   * Volume backups must be stored in the S3 bucket.
   * S3 credentials are in the credentials-velero configuration file.
   * Partial OADP configuration with S3 information is in the oadp-config.yml file.
   * Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* + Change to the ~/DO380/labs/backup-review directory.

[student@workstation ~]$ **cd ~/DO380/labs/backup-review**

* + Create the cloud-credentials secret in the openshift-adp namespace with the credentials-velero file content.

[student@workstation backup-review]$ **oc create secret generic \**

**cloud-credentials -n openshift-adp --from-file cloud=credentials-velero**

secret/cloud-credentials created

* + Complete the OADP configuration in the oadp-config.yml file to enable CSI snapshots and Data Mover.

apiVersion: oadp.openshift.io/v1alpha1

kind: DataProtectionApplication

metadata:

name: oadp-config

namespace: **openshift-adp**

spec:

...output omitted...

configuration:

nodeAgent:

**enable: true**

**uploaderType: kopia**

velero:

**defaultPlugins:**

**- openshift**

**- aws**

**- csi**

**defaultSnapshotMoveData: true**

* + Apply the OADP configuration by using the oadp-config.yml file.

[student@workstation backup-review]$ **oc apply -f oadp-config.yml**

dataprotectionapplication.oadp.openshift.io/oadp-config created

* + Verify that the BackupStorageLocation object is created and in the Available phase.

[student@workstation backup-review]$ **oc get -n openshift-adp \**

**BackupStorageLocations**

NAME PHASE LAST VALIDATED AGE DEFAULT

oadp-config-1 **Available** 18s 2m51s true

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Identify the storage classes that the application uses. Ensure that the application volumes can be backed up by using CSI snapshots. Configure the matching volume snapshot classes for OADP.
   * List the persistent volume claims in the wiki namespace and identify the storage classes.
   * [student@workstation backup-review]$ **oc -n wiki get pvc**
   * NAME ... STORAGECLASS AGE
   * mediawiki ... **ocs-external-storagecluster-cephfs** 5m45s

postgresql-postgresql-0 ... **ocs-external-storagecluster-ceph-rbd** 5m43s

* + Identify the provisioner for each storage class from the previous step.

[student@workstation backup-review]$ **oc get storageclasses \**

**ocs-external-storagecluster-ceph-rbd ocs-external-storagecluster-cephfs**

NAME PROVISIONER ...

ocs-external-storagecluster-ceph-rbd **openshift-storage.rbd.csi.ceph.com** ...

ocs-external-storagecluster-cephfs **openshift-storage.cephfs.csi.ceph.com** ...

* + List the available volume snapshot classes. Ensure that each storage class from the previous step has a matching volume snapshot class with the same driver.

[student@workstation backup-review]$ **oc get volumesnapshotclasses**

NAME DRIVER

ocs-...-cephfsplugin-snapclass **openshift-storage.cephfs.csi.ceph.com**

ocs-...-rbdplugin-snapclass **openshift-storage.rbd.csi.ceph.com**

* + For each volume snapshot class, set the velero.io/csi-volumesnapshot-class label to true.

[student@workstation backup-review]$ **oc label volumesnapshotclass \**

**velero.io/csi-volumesnapshot-class="true" --all**

.../ocs-external-storagecluster-cephfsplugin-snapclass labeled

.../ocs-external-storagecluster-rbdplugin-snapclass labeled

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create a scheduled daily backup of the wiki project. The backup must start at 11 PM every day. Use backup hooks to prepare the application for backup by using the provided commands on both the MediaWiki and PostgreSQL pods. Ensure that both CephFS and RBD volumes are backed up.

For this exercise, disable the schedule with the paused field to prevent unexpected backups from starting during the hands-on activity. You manually trigger a backup from this schedule in a later step.

Only the resources that the application uses or required by OADP should be in the backup. Use the application labels to filter the resources to back up. The application uses the following resource types:

* + Deployments
  + Stateful Sets
  + Secrets
  + Services
  + Routes
  + Persistent Volumes

You can use the partial resource definition files in the ~/DO380/labs/backup-review path.

* + Modify the partial resource definition in the ~/DO380/labs/backup-review/schedule.yml file as follows:

apiVersion: velero.io/v1

kind: Schedule

metadata:

name: **wiki-backup**

namespace: **openshift-adp**

spec:

schedule: **"0 23 \* \* \*"**

paused: true

template:

includedNamespaces:

- **wiki**

orLabelSelectors:

- matchLabels:

**app.kubernetes.io/part-of: mediawiki**

- matchLabels:

**kubernetes.io/metadata.name: wiki**

includedResources:

- **deployments**

- **statefulsets**

- **secrets**

- **pvc**

- **pv**

- **services**

- **routes**

- **pods**

- **namespace**

hooks:

resources:

- name: mediawiki-lock

labelSelector:

matchLabels:

**app.kubernetes.io/name: mediawiki**

pre:

- exec:

container: **mediawiki**

command:

- **/bin/bash**

- **-c**

- **echo "backup in progress" > /data/images/lock\_yBgMBwiR;**

post:

- exec:

container: **mediawiki**

command:

- **rm**

- **/data/images/lock\_yBgMBwiR**

- name: postgresql-checkpoint

labelSelector:

matchLabels:

**app.kubernetes.io/name: postgresql**

pre:

- exec:

container: **postgresql**

command:

- **psql**

- **-c**

- **"CHECKPOINT;"**

* + Apply the configuration for the schedule resource.
  + [student@workstation backup-review]$ **oc apply -f schedule.yml**

schedule.velero.io/wiki-backup created

* + Create an alias to access the velero binary from the Velero deployment in the openshift-adp namespace.

[student@workstation backup-review]$ **alias velero='\**

**oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'**

* + Verify the status of the schedule with the velero command.

[student@workstation backup-restore]$ **velero get schedule**

NAME STATUS CREATED SCHEDULE ...

wiki-backup New 2023-11-17 14:37:39 +0000 UTC 0 23 \* \* \* ...

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

1. Trigger an immediate backup from the scheduled backup, and restore it to the wiki-staging project. Use restore hooks to unlock the application.

The backup and restore should be completed without any errors or warnings.

* + Use the velero command to start a backup by using the schedule definition from the previous step. Note the name of the backup that the command creates, to use in the next step.

[student@workstation backup-review]$ **velero backup create \**

**--from-schedule wiki-backup**

...output omitted...

Creating backup from schedule, all other filters are ignored.

Backup request "***wiki-backup-20231115113447***" submitted successfully.

Run **velero backup describe *wiki-backup-20231115113447*** or **velero backup logs *wiki-backup-20231115113447*** for more details.

### Note

The S3 object storage that is configured in the lab environment uses a custom certificate signed with the OpenShift service CA. You must add the CA certificate to the velero backup describe --details and velero backup logs commands as follow:

[user@host]$ **velero backup logs \**

**--cacert=/run/secrets/kubernetes.io/serviceaccount/service-ca.crt \**

***wiki-backup-20231115113447***

* + Monitor the status of the backup and verify that the backup ends with the Completed status. The backup process takes several minutes.

[student@workstation backup-review]$ **velero get backup**

NAME STATUS ERRORS WARNINGS ...

***wiki-backup-20231115113447*** **Completed** 0 0 ...

* + Create the restore resource in the openshift-adp namespace to restore the wiki project to a new wiki-staging project. Use the backup name from the previous step. Configure a post-restore hook to remove the MediaWiki lock file from the mediawiki volume.

You can use the incomplete restore.yml file as the starting point.

apiVersion: velero.io/v1

kind: Restore

metadata:

name: **wiki-staging**

namespace: **openshift-adp**

spec:

backupName: ***wiki-backup-20231115113447***

namespaceMapping:

**wiki: wiki-staging**

hooks:

resources:

- name: mediawiki-unlock

labelSelector:

matchLabels:

**app.kubernetes.io/name: mediawiki**

postHooks:

- exec:

container: **mediawiki**

command:

- **rm**

- **/data/images/lock\_yBgMBwiR**

* + Create the resource by using the YAML manifest.
  + [student@workstation backup-review]$ **oc apply -f restore.yml**

restore.velero.io/wiki-staging created

* + Use the velero command to get the status of the restore. Monitor the output to verify that the restore status is Completed. The restore process takes several minutes.

[student@workstation backup-review]$ **velero get restore**

NAME BACKUP STATUS ... ERRORS WARNINGS

wiki-staging ***wiki-backup-20231115113447*** **Completed** ... 0 0

* + Review the restored resources in the wiki-staging project.

[student@workstation backup-review]$ **oc -n wiki-staging get all**

NAME READY STATUS RESTARTS AGE

pod/mediawiki-77dfffc8bd-nlc74 1/1 Running 0 67s

pod/postgresql-0 1/1 Running 0 66s

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

service/mediawiki ClusterIP 172.30.86.62 <none> 8080/TCP 67s

service/postgresql ClusterIP 172.30.216.157 <none> 5432/TCP 67s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/mediawiki 1/1 1 1 66s

NAME DESIRED CURRENT READY AGE

replicaset.apps/mediawiki-77dfffc8bd 1 1 1 66s

NAME READY AGE

statefulset.apps/postgresql 1/1 66s

NAME HOST/PORT

route.route.openshift.io/mediawiki mediawiki-wiki-staging.apps.ocp4.example.com

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

1. Change the MediaWiki site name and site URL in the wiki-staging project.
   * Review the mediawiki route URL in the wiki-staging project.

[student@workstation backup-review]$ **oc -n wiki-staging get route**

NAME HOST/PORT PATH SERVICES

mediawiki **mediawiki-wiki-staging.apps.ocp4.example.com** mediawiki

* + Set the MEDIAWIKI\_SITE\_SERVER environment variable on the mediawiki deployment to match the route URL from the previous step. Set the MEDIAWIKI\_SITE\_NAME environment variable to DO380 Team Wiki Staging.

[student@workstation backup-review]$ **oc -n wiki-staging \**

**set env deployment/mediawiki \**

**MEDIAWIKI\_SITE\_SERVER=https://mediawiki-wiki-staging.apps.ocp4.example.com \**

**MEDIAWIKI\_SITE\_NAME="DO380 Team Wiki Staging"**

deployment.apps/mediawiki updated

* + Verify that the mediawiki deployment is ready.

[student@workstation backup-review]$ **oc get deployment -n wiki-staging**

NAME READY UP-TO-DATE AVAILABLE AGE

mediawiki **1/1** 1 1 14m

* + Open a web browser and navigate to https://mediawiki-wiki-staging.apps.ocp4.example.com. Verify the MediaWiki site name in the browser window and that you can edit any page.

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

1. Remove the manual backup and restore resources from the previous steps.
   * Use the velero command to delete the backup. Use the backup name from the previous step.

[student@workstation backup-review]$ **velero delete \**

**backup *wiki-backup-20231115113447***

Are you sure you want to continue (Y/N)? **y**

Request to delete backup "***wiki-backup-20231115113447***" submitted successfully.

The backup will be fully deleted after all associated data (disk snapshots, backup files, restores) are removed.

* + Change to the student HOME directory.

[student@workstation backup-review]$ **cd**

[student@workstation ~]$

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

**Evaluation**

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

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

# **Chapter 3.  Cluster Partitioning**

[Node Pools](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03)

[Quiz: Node Pools](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s02)

[Node Configuration with the Machine Configuration Operator](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s03)

[Guided Exercise: Node Configuration with the Machine Configuration Operator](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s04)

[Node Configuration with Special Purpose Operators](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s05)

[Guided Exercise: Node Configuration with Special Purpose Operators](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s06)

[Lab: Cluster Partitioning](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s07)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch03s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Configure a subset of cluster nodes to be dedicated to a type of workload. |
| **Sections** | * Node Pools (and Quiz) * Node Configuration with the Machine Configuration Operator (and Guided Exercise) * Node Configuration with Special Purpose Operators (and Guided Exercise) |
| **Lab** | * Cluster Partitioning |

## **Node Pools**

### **Objectives**

* Illustrate methods of adding and configuring OpenShift cluster nodes, by using node labels to configure nodes and to schedule pods to those nodes.

### **Introduction**

The OpenShift cluster has control plane nodes and compute nodes. Compute nodes are also known as worker nodes. The control plane nodes manage the Red Hat OpenShift Container Platform cluster and orchestrate the distribution of the workloads on the compute nodes.

The applications that run in the OpenShift cluster have different specifications and requirements. Some applications might require compute nodes with graphics processing units (GPU) for graphical applications, artificial intelligence (AI) computations, or machine learning (ML) workloads. Some applications, such as simple websites, might not require specific hardware or specifications, and work smoothly on a small compute node.

For example, you run an application that is composed of many microservices that require high CPU capacity, and a database that does not have such a requirement.

|  |
| --- |
|  |

You can run both the CPU-intensive application and the database instances on compute nodes with high CPU capacity. In this case, the CPU-intensive application runs smoothly, but the database does not use the high CPU node to full capacity, and the billing cost increases.

In OpenShift, you can deploy both the CPU-intensive application and the database instances on compute nodes with low CPU capacity. In this case, the billing cost reduces but the application does not work correctly because of the low CPU compute node.

|  |
| --- |
|  |

The best solution is to deploy the CPU-intensive application instances on high CPU compute nodes and to deploy the database application on the low CPU compute nodes.

You need compute nodes with the different capacity and hardware to accommodate these applications, and a partition of these nodes to deploy the application. You can deploy these applications on specific nodes by using node pools.

In some cases, you might face a "noisy neighbor" issue where a application gives poor performance and high latency due to other applications. You can configure compute nodes to be part of node pools for critical applications, to avoid the noisy neighbor issue.

### **Node Pools**

A node pool is a logical group of OpenShift compute nodes. Compute nodes with similar hardware configurations can be organized into a node pool. Collecting these similar nodes into node pools provides a method of targeting placement for workload deployments.

You can configure multiple node pools for your workloads to ensure that your applications run on the required hardware.

An instance often requires a specific configuration for a compute node to run smoothly.

|  |
| --- |
|  |

In this scenario, you have an application that uses ML and that also uses another database application. The ML application requires nodes with a GPU to run smoothly, whereas a database can run on a standard compute node.

In this case, provision compute nodes with GPU hardware and configure GPU compute nodes as part of a gpu node pool by using the node-pool=gpu label. You can schedule the ML application pods on the gpu node pool, whereas database pods deploy on other compute nodes.

You might have critical applications without specific hardware requirements and that can run on a standard compute node. These critical applications might face the "noisy neighbor" issue.

For example, consider a messaging application. The application does not require specific hardware and can run on any standard node, but cannot afford high network latency. Other non-critical applications might run on the same cluster node and increase network traffic and latency for the messaging application.

Although the application would typically run adequately on that compute node, other non-critical applications can affect the network performance of the compute node. The messaging application faces another "noisy neighbor" issue. To avoid this issue, you can use a node pool to organize your compute nodes to run critical applications.

Defining node pools for your cluster hardware is specific to the approach for application deployments in your organization. In a scenario with multi-tiered infrastructure deployments, such as a development, staging, or production tier of hardware, administrators can organize node pools to correspond to these tiers. Earlier hardware can populate a "development" node pool, whereas later infrastructure can be designated for production workloads through a "production" node pool. With this approach, developers can target these node pools to separate development, staging, and production applications that run in the cluster.

#### **Node Provisioning**

Compute node provisioning depends on the method of cluster installation.

In a user-provisioned infrastructure (UPI), you must manually provision new instances. The strategy for provisioning nodes is specific to your data center and your IT processes. However, the base steps remain the same, as follows:

1. Update the compute node Ignition file with an updated TLS certificate.
2. Install Red Hat Enterprise Linux CoreOS (RHCOS) from an ISO image or by using a Preboot eXecution Environment (PXE) boot approach.
3. Add the new instance to the ingress load balancer.
4. Approve the Certificate Signing Requests (CSRs).

In an installer-provisioned infrastructure (IPI), the machine API automatically performs scaling operations for supported cloud providers. Thus, you can modify the specified number of replicas in a machine set resource, and OpenShift communicates with the cloud provider to provision or remove instances. You can scale up and scale down by using autoscaling based on the workload requirements.

For more details about Red Hat OpenShift installation and node provisioning, refer to the DO322: Red Hat OpenShift Installation Lab training course.

Some managed clusters have protected machine sets for direct modification and support node provisioning by using node pools. For example, Red Hat OpenShift on AWS (ROSA) clusters use machine pools. You create a machine pool and select an instance type that is specific to the workload. The machine set is protected against direct modification. You can scale a machine pool manually or you can use the cluster autoscaler.

For more details about ROSA and autoscaling, refer to the CS220: Creating and Configuring Production Red Hat OpenShift on AWS (ROSA) Clusters training course.

#### **Node Labels**

Use node labels to configure compute nodes to be a member of a specific node pool. Node labels are key-value pairs that are attached to the node.

You can add a label to multiple compute nodes to designate each as a member of one node pool. In UPI clusters, specify node labels for the members of a node pool within a MachineSet object since the node labels can be overridden by the machine or cluster API.

### Note

When applying node labels to exisiting members of a MachineSet, it is necessary to redeploy the nodes before the node label is available. For more information on defining and applying node labels within a MachineSet, see the following Red Hat article: <https://access.redhat.com/solutions/6955894>.

You can list the node and labels by using the oc get nodes command:

[user@host ~]$ **oc get nodes worker01 --show-labels**

NAME STATUS ROLES ... LABELS

worker01 Ready worker ... **beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster.ocs.openshift.io/openshift-storage=,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker01,kubernetes.io/os=linux,node-role.kubernetes.io/worker=,node.openshift.io/os\_id=rhcos**

Compute nodes can have many node labels. Add as many node labels to a compute node as needed for workload placement onto the intended hardware. You can add a custom node label to the compute nodes by using the oc label node command.

[user@host ~]$ **oc label node/worker01 node-pool=gpu**

**node/worker01 labeled**

[user@host ~]$ **oc get nodes worker01 --show-labels**

NAME STATUS ROLES ... LABELS

worker01 Ready worker ... beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cluster.ocs.openshift.io/openshift-storage=,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker01,kubernetes.io/os=linux,node-role.kubernetes.io/worker=,node.openshift.io/os\_id=rhcos,**node-pool=gpu**

|  |  |
| --- | --- |
|  | The oc label command adds the node-pool=gpu label to the worker01 compute node. |
|  | The oc get nodes command displays the node-pool=gpu label for the worker01 compute node. |

You can configure multiple GPU compute nodes to be part of a gpu node pool by adding the same node label to all GPU-enabled compute nodes.

[user@host ~]$ **oc label node/worker02 node-pool=gpu**

**node/worker02 labeled**

[user@host ~]$ **oc get nodes --selector node-pool=gpu**

NAME STATUS ROLES AGE VERSION

**worker01** Ready worker 13d v1.25.7+eab9cc9

**worker02** Ready worker 13d v1.25.7+eab9cc9

The oc get nodes command with the selector filter shows all nodes with the node-pool=gpu label. You can schedule the application to target a compute node in the gpu node pool by using node labels. The next chapter discusses pod scheduling in detail.

#### **Node Configuration**

The OpenShift cluster manages upgrades to the nodes by using the machine configuration operator (MCO). You can manage nodes by defining groups of these nodes, which are called machine configuration pools (MCP). The MCO uses the master and worker MCPs, by default. You can create custom MCPs for node pools. MCPs use labels to match one or more MCs to one or more nodes.

For example, you can create the gpu MCP. The gpu MCP selects nodes based on its assigned gpu node role.

Similarly, cloud providers also provide different compute and storage instances to meet customer requirements. For example etcd storage and I/O-intensive workloads require a low-latency storage solution for optimal performance. AWS has an io2 EBS volume type to support these workloads. You can use the gp2 EBS volume type for other workloads where intensive I/O is not required.

### References

For more information about the control plane architecture, refer to the Control Plane Architecture chapter in the Red Hat OpenShift Container Platform 4.14 Architecture documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/architecture/index#control-plane>

For more information about adding compute nodes to a cluster, refer to the About Adding RHEL Compute Nodes to a Cluster section in the Adding RHEL Compute Machines to an OpenShift Container Platform Cluster chapter in the Red Hat OpenShift Container Platform 4.14 Machine Management documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/machine_management/index#adding-rhel-compute>

For more information about the installation prerequisites, refer to the Prerequisites section in the Deploying Installer-provisioned Clusters on Bare Metal chapter in the Red Hat OpenShift Container Platform 4.14 Installing documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/installing/index#ipi-install-prerequisites>

For more details about ROSA and Microsoft Azure Red Hat OpenShift, refer to the DO120: Introduction to Red Hat OpenShift Service on AWS (ROSA) and DO121: Introduction to Microsoft Azure Red Hat OpenShift training courses respectively.

[Creating and Adding Additional Worker Nodes to the Cluster](https://access.redhat.com/solutions/4270972)

[Infrastructure Nodes in OpenShift 4](https://access.redhat.com/solutions/5034771)

[Kubernetes: Assign Pods to Nodes](https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes)

## **Node Configuration with the Machine Configuration Operator**

### **Objectives**

* Apply operating system settings to cluster nodes with the machine configuration operator.

### **The Machine Configuration Operator**

OpenShift uses Red Hat Enterprise Linux CoreOS (RHCOS) as the underlying operating system in the hosts. The entire operating system is updated as a single image, instead of on a package-by-package basis. The machine configuration operator (MCO) manages the RHCOS operating system upgrades and configuration changes.

Do not use traditional Red Hat Enterprise Linux management approaches, such as manually editing files or using system commands such as the systemctl command, for RHCOS operating system upgrades or configuration changes. Those changes can conflict with the MCO and the MCO can override them.

The MCO comprises the following pods:

**machine-config-operator**

These pods form the main operator workload that manages the rest of the MCO components.

**machine-config-controller**

The machine configuration controller (MCC) manages the synchronization of machine upgrades according to specified configurations through a machine configuration object. The MCC offers options to upgrade individual sets of machines.

**machine-config-server**

The machine configuration server (MCS) provides instance customizations to machines that join the cluster. The MCS uses Ignition to provide the instance customizations to cluster nodes that join the cluster. The RHCOS operating system downloads and processes Ignition files at boot time.

### Note

Installing an OpenShift cluster and using Ignition files to configure the cluster nodes is explained in detail in the DO322: Red Hat OpenShift Installation Lab course.

The MCO also requires the machine-config-daemon daemonset that RHCOS provides. The machine configuration daemon (MCD) implements updates to machine configurations and validates each machine's state in accordance with the requested configuration.

The MCO can manage the following resources:

* systemd services
* SSH keys
* Kernel arguments
* Files in the /var or /etc directories.

The MCO can also manage directories, such as the /opt and /usr/local directories, which can be writeable if symbolically linked to the /var or /etc directories.

The MCO defines two custom resources (CRs), which are part of the machineconfiguration.openshift.io/v1 API group.

**MachineConfig**

A machine configuration (MC) CR declares instance customizations by using the Ignition configuration format.

**MachineConfigPool**

A machine configuration pool (MCP) CR uses labels to match one or more MCs to one or more nodes by means of the machineConfigSelector and nodeSelector parameters, respectively. This resource creates a pool of nodes with the same configuration. The MCO uses the MCP to track status when the MCO applies MCs to the nodes.

OpenShift administrators set custom node configurations by declaring the MachineConfig and MachineConfigPool CRs.

The following diagram shows the relationship between the node, MC, and MCP labels:

|  |
| --- |
|  |

Figure 3.4: Relationship between the node, MC, and MCP labels

In the previous diagram, the MCP CR uses the nodeSelector parameter to select all the nodes with the worker role, and applies to them the MCs with the machineconfiguration.openshift.io/role: worker label that is selected by using the machineConfigSelector parameter.

The MCO also manages two custom resources (CRs) for modifying CRI-O container runtime settings and the Kubelet service: the ContainerRuntimeConfig and KubeletConfig CRs, respectively.

### **Machine Configurations**

MCs declare instance configurations by using the Ignition configuration format.

Ignition files encode file contents by using the Base64 encoding scheme. Other items, such as systemd units or SSH keys, do not use the Base64 encoding scheme. In a terminal, use the base64 and base64 -d commands to encode and decode files or standard input.

You can also use Butane to create Ignition files and MCs. Butane simplifies handling Base64 encoding.

### Note

For more information about how to use Butane to create MCs, refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/installing/index#installation-special-config-butane_installing-customizing>

The following example shows an MC to modify the configuration file for the journald service by using the Base64 encoding scheme:

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: worker

name: 60-journald

spec:

config:

ignition:

version: 3.2.0

storage:

files:

- contents:

source: data:text/plain;charset=utf-8;base64,VGVzdGl...E8gKItAo=

filesystem: root

mode: 0644

path: /etc/systemd/journald.conf

|  |  |
| --- | --- |
|  | The label for the MC. The MC uses the machineconfiguration.openshift.io/role MC role label. By default, OpenShift comes with preinstalled MCs for control plane and compute nodes. |
|  | Prefix the name with a two-digit number that specifies when to apply the configuration, relative to MCs that belong to the same MCP. Higher numbers have precedence. |
|  | Use the data URL format to embed escaped file content. Base64 encoding is common for Ignition files. |

The following example shows an MC to change the default kernel to a real-time kernel. This configuration does not use the Base64 encoding scheme:

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: worker

name: 99-kernel-realtime

spec:

kernelType: realtime

You can use the oc explain mc command to read the parameters that you can change by using an MC.

You can list the MCs on your cluster by using the oc get machineconfig or oc get mc commands. To list the MCs for a specific node label, use the --selector argument. The following example lists the MCs for the worker role in the cluster:

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

**--selector machineconfiguration.openshift.io/role=worker**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

00-worker 52fe...bf97 3.2.0 108d

01-worker-container-runtime 52fe...bf97 3.2.0 108d

01-worker-kubelet 52fe...bf97 3.2.0 108d

99-worker-chrony-conf-override 3.2.0 108d

99-worker-generated-registries 52fe...bf97 3.2.0 108d

99-worker-ssh 3.2.0 108d

The MCO reads MCs alphanumerically by the name, from 00-\* to 99-\*. OpenShift stores the resulting compilation of MCs in a rendered MC resource. Thus, if two or more MCs apply changes to the same file, then the MCO applies only the changes from the MC with a higher number. For two MCs with the same number, the MCO uses the last one alphabetically. For example, the 99-worker-ssh-new MC has precedence over the 99-worker-ssh-last MC.

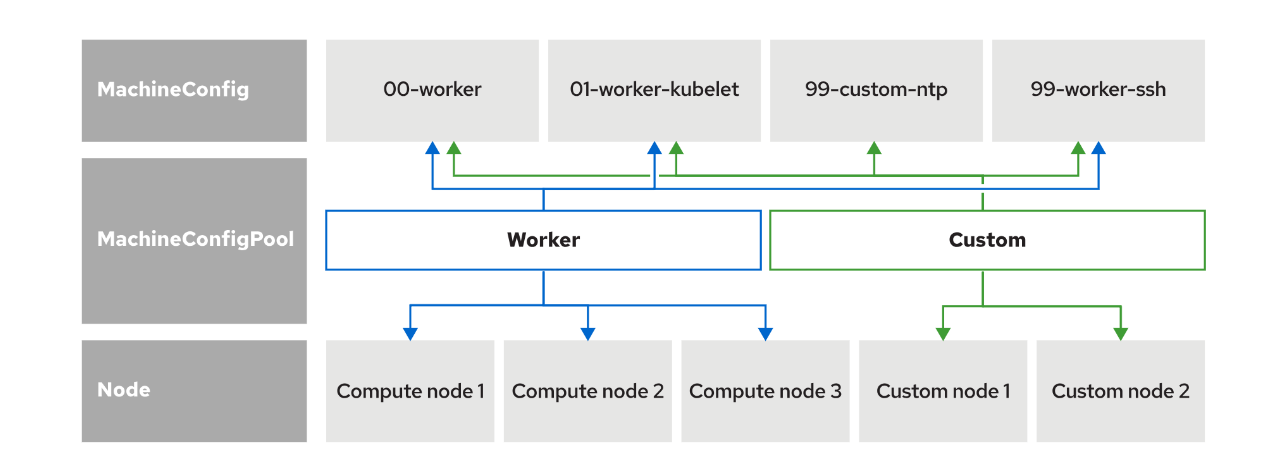
### **Machine Configuration Pools**

MCPs use labels to match one or more MCs to one or more nodes. By using multiple MCs, you can split the node configuration and focus every MC on one aspect of server configuration. For example, you can configure one MC for DNS resolution and another MC to synchronize time.

You can also use the same MCs in multiple MCPs. For example, although a time synchronization MC would apply to multiple MCPs, only the MCPs for a specific node pool would include the configuration that a special hardware accelerator card requires.

MCPs specify a machineConfigSelector MC label to select one or more MCs, and a nodeSelector node label to select one or more nodes.

By default, OpenShift includes MCPs for the master and worker roles. However, you can add custom MCPs to the cluster. Red Hat recommends creating custom MCPs as a composition of worker and custom MCs that are applied to the nodes. Assigning the worker role to the custom MCP is critical so that OpenShift applies operating system updates that are labeled as worker to the machines in the pool. Thus, the machineConfigSelector match expression selects both the worker role and the custom label. The nodes that are part of the custom pool use the MCs from the worker role with the additions from the custom label.



|  |
| --- |
|  |

Figure 3.5: Custom pool with the worker MCs and an additional MC

In the previous diagram, the custom MCP targets two nodes with the custom label, and the worker MCP targets three nodes with the worker label. The worker pool applies to the worker nodes the 00-worker, 01-worker-kubelet, and 99-worker-ssh MCs. The custom pool uses the MCs from the worker role, and adds the 99-custom-ntp MC to the configuration of the custom nodes.

### Important

Nodes with the worker role can be part of only one custom pool, and nodes with the master role cannot be part of a custom pool. In these cases, the MCO does not apply any changes that are specific to the custom pools, and shows an error in the MCC pod logs.

The following MCP specification demonstrates creating a separate pool for custom nodes:

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

name: ***custom***

spec:

machineConfigSelector:

matchExpressions:

- key: machineconfiguration.openshift.io/role

operator: In

values: [worker, ***custom***]

nodeSelector:

matchLabels:

node-role.kubernetes.io/***custom***: ""

|  |  |
| --- | --- |
|  | The name for the custom MCP. |
|  | The MC selector includes both the worker and custom MCs. |
|  | The node selector includes the nodes in the cluster with the custom role. |

### Note

You can find more information about labels and selectors in <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

You can list the MCPs in your cluster by using the oc get machineconfigpool or oc get mcp commands.

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

NAME CONFIG UPDATED UPDATING DEGRADED

MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT AGE

master rendered-master-716...267 True False False

3 3 3 3 108d

worker rendered-worker-573...3db True False False

3 3 3 3 108d

The previous command shows the following information for the MCPs:

**CONFIG**

The name for the rendered MC that is applied to the nodes in the MCP. During the first update, this field remains blank.

**UPDATED**

The True status indicates that the MCO applied the current MC to the nodes in that MCP. The False status indicates that nodes in the MCP are updating.

**UPDATING**

The True status indicates that the MCO is applying the intended MC. Nodes in the UPDATING state might not be available for scheduling. The False status indicates that all nodes in the MCP are updated.

**DEGRADED**

A True status indicates that the MCO is blocked from applying the current or intended MC to at least one node in that MCP. A possible reason is a detection of configuration drift. Configuration drift is explained later in this section. Nodes that are degraded might not be available for scheduling. A False status indicates that all nodes in the MCP are ready.

**MACHINECOUNT**

Indicates the total number of machines in the MCP.

**READYMACHINECOUNT**

Indicates the total number of machines in the MCP that are ready for scheduling.

**UPDATEDMACHINECOUNT**

Indicates the total number of machines in the MCP with the current MC.

**DEGRADEDMACHINECOUNT**

Indicates the total number of machines in that MCP that are degraded or irreconcilable.

### **Label Nodes**

You can add labels to your node by using the oc label command.

The following example adds the custom role to the worker03 node in the cluster.

[user@host ~]$ **oc label node/worker03 node-role.kubernetes.io/custom=**

node/worker03 labeled

You can verify the node roles in the cluster by using the oc get nodes command:

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

NAME STATUS **ROLES** AGE VERSION

master01 Ready control-plane,master 114d v1.25.7+eab9cc9

master02 Ready control-plane,master 114d v1.25.7+eab9cc9

master03 Ready control-plane,master 114d v1.25.7+eab9cc9

worker01 Ready worker 12d v1.25.7+eab9cc9

worker02 Ready worker 12d v1.25.7+eab9cc9

worker03 Ready **custom**,worker 12d v1.25.7+eab9cc9

### **Infrastructure Nodes**

One important node label is the infra role. Use the infra role for nodes that host only infrastructure components, such as cluster logging, cluster monitoring, or the integrated container image registry. Adding the infra role for nodes is recommended for larger clusters to ensure the performance and stability of OpenShift cluster services, such as the router or OAuth services, or to prevent the impact of heavy infrastructure components, such as metrics and logging, on user workloads.

Infrastructure nodes do not count towards the total number of required OpenShift subscriptions to run the environment. You can create a custom MCP to apply MCs to the infrastructure nodes.

### Note

For more information about creating infrastructure nodes, refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-nodes-creating-infrastructure-nodes>

For more information about using infrastructure nodes to separate maintenance and management, and to prevent incurring billing costs against subscription counts, refer to <https://access.redhat.com/solutions/5034771>

### **Machine Configuration Operator Updates**

The MCD reports the state of the node updates by using node annotations. You can use these annotations to assess the state of the update.

You can list the node annotations by using the oc describe command:

[user@host ~]$ **oc describe node worker01**

Name: worker01

Roles: worker

Labels: beta.kubernetes.io/arch=amd64

...output omitted...

node-role.kubernetes.io/worker=

node.openshift.io/os\_id=rhcos

**Annotations**: **machineconfiguration.openshift.io/currentConfig**: rendered-worker-370...bfd

**machineconfiguration.openshift.io/desiredConfig**: rendered-worker-370...bfd

machineconfiguration.openshift.io/reason:

**machineconfiguration.openshift.io/state**: Done

volumes.kubernetes.io/controller-managed-attach-detach: true

...output omitted...

|  |  |
| --- | --- |
|  | The current rendered MC that is applied to the node. |
|  | The intended rendered MC to be applied to the node. |
|  | The current node state regarding MCO. |

When the intended configuration does not match the current configuration, the MCD applies the intended rendered MC, drains all the pods from the node, and reboots the node.

#### **Configuration Drift**

A configuration drift is the state where the configuration on a node does not fully match what the currently applied rendered MC specifies.

The MCD checks for configuration drifts when a node boots, or when any of the specified files in the MC are modified outside the MC, or before a new MC is applied.

When the MCD detects a configuration drift, the MCD performs the following tasks:

* Logs an error message to the console.
* Generates a Kubernetes event.
* Stops additional drift detection on the affected node.
* Marks both the node and the MCP with the degraded state.

The MCO marks the node in the degraded state until an administrator corrects the node configuration. Although a degraded node is online and operational, you cannot update it.

You can correct configuration drift and return the node to the Ready state with one of the following remediations:

**Generate**

Generate a force file on the degraded node to bypass the configuration drift detection and reapply the current MC. To generate the force file, create a debug pod on the node with the degraded state and create the /run/machine-config-daemon-force file. Then, OpenShift skips the MC validation, restarts the node, and applies the current MC to the node. The force file does not force the node upgrade; it instead skips validation of configurations on the system and attempts an update regardless of the difference. Depending on the issue on your node, skipping the validation process might not help you move past your node error.

**Rewrite**

Rewrite the file contents or change the file permissions of the files on the node to match the MC configuration. This manual procedure requires you to review the logs and manually fix the conflicting file. This remediation does not require rebooting the node in a degraded state, and thus avoids possible downtime in your applications.

For information about configuration drift, refer to the Status field for the pool with the degraded node:

[user@host ~]$ **oc describe mcp *worker***

...output omitted...

Status:

Conditions:

...output omitted...

Last Transition Time: 2023-10-02T10:11:37Z

**Message: Node worker01 is reporting: "content mismatch for file '/etc/containers/registries.conf'"**

Reason: 1 nodes are reporting degraded status on sync

Status: True

**Type: NodeDegraded**

Last Transition Time: 2023-10-02T10:11:37Z

Message:

Reason:

Status: True

Type: Degraded

...output omitted...

In the previous example, the MCO detects a configuration drift for the /etc/containers/registries.conf file.

You can get the MCD pod name for a specific node by using a filter.

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

**--field-selector spec.nodeName=*worker01***

NAME READY STATUS RESTARTS AGE

***machine-config-daemon-jsrzm*** 2/2 Running 2 19d

You can also review the logs for the MCD that gives you more information about the configuration drift.

[user@host ~]$ **oc logs *machine-config-daemon-jsrzm* \**

**-n openshift-machine-config-operator**

...output omitted...

E1002 10:11:36.163676 2667 daemon.go:589] Preflight config drift check failed: content mismatch for file "/etc/containers/registries.conf"

E1002 10:11:36.163692 2667 writer.go:200] Marking Degraded due to: content mismatch for file "/etc/containers/registries.conf"

W1002 10:11:40.193224 2667 daemon.go:1763] current+desiredConfig is rendered-worker-d4f45006b2d83725d98af944c8296774 but state is Degraded

I1002 10:11:40.510208 2667 rpm-ostree.go:394] Running captured: rpm-ostree kargs

E1002 10:11:40.568388 2667 on\_disk\_validation.go:207] **content mismatch for file "/etc/containers/registries.conf" (-want +got)**:

[]uint8(

"""

**- unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]**

**+ unqualified-search-registries = ["docker.io"]**

short-name-mode = ""

... // 107 identical lines

"""

)

E1002 10:11:40.568446 2667 daemon.go:589] Preflight config drift check failed: content mismatch for file "/etc/containers/registries.conf"

E1002 10:11:40.568467 2667 writer.go:200] Marking Degraded due to: content mismatch for file "/etc/containers/registries.conf"

E1002 10:11:46.987190 2667 daemon.go:1077] **mode mismatch for file: "/etc/containers/registries.conf"; expected: -rw-r--r--/420/0644; received: -rwxrwxrwx/511/0777**

E1002 10:11:46.987222 2667 writer.go:200] Marking Degraded due to: mode mismatch for file: "/etc/containers/registries.conf"; expected: -rw-r--r--/420/0644; received: -rwxrwxrwx/511/0777

...output omitted...

In the previous example, the MCD marks the worker01 node in a degraded state due to the mismatches of content and permissions for the file.

### **Configure MCO-related Custom Resources**

The MCO provides the ContainerRuntimeConfig CR to modify CRI-O container runtime settings, and the KubeletConfig CR to manage the Kubelet service. You can use these CRs to configure a subset of CRI-O and Kubelet configuration parameters. Always use valid values for the configuration parameter, because invalid values might render cluster nodes unusable.

Although you can modify the CRI-O container runtime settings and the Kubelet service by using the MachineConfig CR, using either of these two special CRs simplifies node deployment and configuration management, provides API checking, and prevents misconfigurations. Moreover, because OpenShift does not support changing all the settings of the Kubelet service and the container runtime, these CRs provide only the configuration changes that OpenShift supports.

#### **Create a Custom Resource for Kubelet Configuration**

OpenShift provides a kubelet configuration controller to the MCC. You can use the KubeletConfig CR to edit the kubelet parameters.

The MCO can write the kubelet.conf configuration file and the kubelet.system systemd unit file to Ignition, so Ignition writes these two files to configure the kubelet agent when it starts on a node. If you create an MC to change the kubelet parameters, then the MCD reboots the nodes to write the new configuration. With this approach, OpenShift can restore the default kubelet configuration if you delete the KubeletConfig instance.

### Note

For a list of all the parameters that you can modify by using the KubeletConfig CR, you can refer to the KubeletConfiguration API object in Kubernetes that uses the same parameters. Refer to <https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration>

#### **Create a Custom Resource for Container Runtime Configuration**

You can change the settings for the OpenShift CRI-O runtime by using the ContainerRuntimeConfig CR. The MCO can write the crio.conf and storage.conf configuration files on the associated nodes with the updated values.

You can modify the following parameters by using a ContainerRuntimeConfig CR:

**Logging level**

The logLevel parameter sets the level of verbosity for logging messages. The default level is info. Other options include the fatal, panic, error, warn, debug, and trace options.

**Overlay size**

The overlaySize parameter sets the maximum size of a container image.

**Container runtime**

The defaultRuntime parameter sets the container runtime to either runc (the default) or crun.

### Important

Support for the crun container runtime is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements and might not be functionally complete. Red Hat does not recommend using Technology Preview features in production.

You can also use the ContainerRuntimeConfig CR to change the limit of PIDs or the maximum logging size. However, Red Hat recommends using the KubeletConfig CR to change these parameters, because they will likely be deprecated in a future version.

### References

For more information about the Machine Configuration Operator, refer to the Post-installation Machine Configuration Tasks section in the Red Hat OpenShift Container Platform 4.14 Post-installation Configuration documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/postinstallation_configuration/post-install-machine-configuration-tasks>

[How does Machine Config Pool work?](https://www.redhat.com/en/blog/openshift-container-platform-4-how-does-machine-config-pool-work)

## **Guided Exercise: Node Configuration with the Machine Configuration Operator**

Label a cluster node and apply a machine configuration to it, without affecting other cluster nodes.

**Outcomes**

* Use the machine configuration operator (MCO) to modify the Network Time Protocol (NTP) client configuration.
* Create a machine configuration pool (MCP) and a machine configuration (MC).
* Observe the status of MC updates.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start nodes-mco**

**Instructions**

Your company requires you to configure an NTP server for the nodes in the cluster to synchronize services among them. The new NTP server is 0.rhel.pool.ntp.org.

In production environments, it is common to change the default NTP configuration to refer to NTP servers in the corporate LAN, instead of to public NTP servers from the internet, to prevent clock skew. However, in this exercise, you apply a new NTP configuration only to the worker01 node, to eliminate waiting time for node reboots when applying MCs, and to show how to apply MCs only to a subset of nodes by using labels.

To configure the NTP server in the worker01 node, create the custom MCP. The custom MCP must apply the MCs with the worker and custom labels to the nodes with the custom role. Then, create an MC for the NTP configuration with the custom label.

1. Connect to the OpenShift cluster and verify the roles for the nodes in the cluster. List the MCs that apply to the worker and custom roles.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. Change to the nodes-mco project.

[student@workstation ~]$ **oc project nodes-mco**

Now using project "nodes-mco" on server "https://api.ocp4.example.com:6443".

* 1. Verify the nodes in the cluster and their roles.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 109d v1.27.10+28ed2d7

master02 Ready control-plane,master 109d v1.27.10+28ed2d7

master03 Ready control-plane,master 109d v1.27.10+28ed2d7

worker01 Ready worker 21d v1.27.10+28ed2d7

worker02 Ready worker 21d v1.27.10+28ed2d7

worker03 Ready worker 21d v1.27.10+28ed2d7

* 1. List the MCs that apply to the worker roles.

[student@workstation ~]$ **oc get machineconfig \**

**-l machineconfiguration.openshift.io/role=worker**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

00-worker 52fe2...ebf97 3.4.0 12d

01-worker-container-runtime 52fe2...ebf97 3.4.0 12d

01-worker-kubelet 52fe2...ebf97 3.4.0 12d

97-worker-generated-kubelet 52fe2...ebf97 3.4.0 12d

98-worker-generated-kubelet 52fe2...ebf97 3.4.0 12d

99-worker-chrony-conf-override 3.2.0 12d

99-worker-generated-registries 52fe2...ebf97 3.4.0 12d

99-worker-ssh 3.2.0 12d

* 1. List the MCs that apply to the custom roles.

[student@workstation ~]$ **oc get machineconfig \**

**-l machineconfiguration.openshift.io/role=custom**

No resources found

1. Label the worker01 node with the node-role.kubernetes.io/custom label. Verify that the node has the correct label.
   1. Apply the node-role.kubernetes.io/custom label to the worker01 node.

[student@workstation ~]$ **oc label node/worker01 node-role.kubernetes.io/custom=**

node/worker01 labeled

* 1. Verify that the node has the correct label.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 33d v1.27.10+28ed2d7

master02 Ready control-plane,master 33d v1.27.10+28ed2d7

master03 Ready control-plane,master 33d v1.27.10+28ed2d7

worker01 Ready **custom**,worker 12d v1.27.10+28ed2d7

worker02 Ready worker 12d v1.27.10+28ed2d7

worker03 Ready worker 12d v1.27.10+28ed2d7

1. Create an MCP for the node with the custom label. The MCP must include the MCs for the worker and custom labels.
   1. Change to the ~/DO380/labs/nodes-mco directory.

[student@workstation ~]$ **cd ~/DO380/labs/nodes-mco**

* 1. Create the MCP custom resource (CR) YAML file. You can find an example for the CR in the ~/DO380/labs/nodes-mco/custom-mcp.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

name: **custom**

spec:

machineConfigSelector:

matchExpressions:

- key: machineconfiguration.openshift.io/role

operator: In

values: **[worker,custom]**

nodeSelector:

matchLabels:

**node-role.kubernetes.io/custom: ""**

* 1. Apply the configuration for the MCP CR.

[student@workstation nodes-mco]$ **oc create -f custom-mcp.yml**

machineconfigpool.machineconfiguration.openshift.io/custom created

* 1. Verify that the custom MCP is correctly created. If the UPDATING field is marked as true for the custom MCP, then wait until the MCO updates the node.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False False **1** ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify that the worker01 node is assigned for the custom MCP, and that the MCO applies the rendered MC to the node.

[student@workstation nodes-mco]$ **oc get events -n \**

**openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=custom**

... REASON OBJECT MESSAGE

... RenderedConfigGenerated machineconfigpool/custom rendered-custom-2f56....81d0 successfully generated ...

... **SetDesiredConfig** machineconfigpool/custom **Targeted node worker01** to MachineConfig rendered-custom-2f56...81d0

1. Because an OpenShift preconfigured MC configures the NTP client, verify the name for the MCs that modify the chrony configuration for the nodes with the worker label. Create an MC for the chrony configuration that applies to the custom label, with higher precedence than the preconfigured MC. The name for the MC must be 99-zcustom-chrony to have higher precedence.
   1. Verify the names for the MCs that modify the chrony configuration for the nodes with the worker label.

[student@workstation nodes-mco]$ **oc get mc --selector \**

**machineconfiguration.openshift.io/role=worker \**

**-o jsonpath="{range .items[\*]}{.metadata.name} \**

**{': '}{range .spec.config.storage.files[\*]}{.path} \**

**{' '}{end}{'\n'}{end}" | grep chrony**

**99-worker-chrony-conf-override**: /etc/chrony.conf

* 1. Create a chrony configuration file. You can find an example for the CR in the ~/DO380/labs/nodes-mco/chrony-mod.conf file.

pool 0.rhel.pool.ntp.org iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Encode the chrony configuration file in Base64 format. You use the encoded configuration file in a later step. The encoded configuration file would differ on your system.

[student@workstation nodes-mco]$ **base64 -w0 chrony-mod.conf; echo**

***cG9vbCAwLnJoZWwucG9...C92YXIvbG9nL2Nocm9ueQo=***

* 1. Create the 99-zcustom-chrony MC CR YAML file. Use the encoded chrony configuration file from a previous step. You can find an example for the CR in the ~/DO380/labs/nodes-mco/99-zcustom-chrony.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: **custom**

name: **99-zcustom-chrony**

spec:

config:

ignition:

version: 3.2.0

storage:

files:

- contents:

compression: ""

source: data:;base64,***cG9vbCAwLnJoZWwucG9...C92YXIvbG9nL2Nocm9ueQo=***

mode: 420

overwrite: true

path: /etc/chrony.conf

* 1. Apply the configuration for the MC CR.

[student@workstation nodes-mco]$ **oc create -f 99-zcustom-chrony.yml**

machineconfig.machineconfiguration.openshift.io/99-zcustom-chrony created

* 1. Verify that the 99-zcustom-chrony MC is correctly created.

[student@workstation nodes-mco]$ **oc get mc**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

...output omitted...

99-master-generated-registries 52fe26136643a946ff... 3.2.0 109d

99-master-ssh 3.2.0 109d

99-worker-chrony-conf-override 3.2.0 109d

99-worker-generated-registries 52fe26136643a946ff... 3.2.0 109d

99-worker-ssh 3.2.0 109d

**99-zcustom-chrony** 3.2.0 2m5s

...output omitted...

* 1. Verify that the MCP starts updating the resources.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... False **True** False **1** ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify that the MCO applies the rendered MC to the worker01 node.

[student@workstation nodes-mco]$ **oc get events \**

**-n openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=custom**

... REASON OBJECT MESSAGE

... RenderedConfigGenerated machineconfigpool/custom rendered-custom-2f56...81d0 successfully generated ...

... SetDesiredConfig machineconfigpool/custom Targeted node worker01 to MachineConfig rendered-custom-2f56...81d0

... **RenderedConfigGenerated** machineconfigpool/custom **rendered-custom-009f...3e5d successfully generated** ...

... **SetDesiredConfig** machineconfigpool/custom **Targeted node worker01** to MachineConfig rendered-custom-009f...3e5d

* 1. Wait a few minutes until the MCO finishes the update.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False False 1 ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

1. Verify that the MCO applies the chrony configuration to the worker01 node, and that the worker02 and worker03 nodes remain unaltered.
   1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

Temporary namespace openshift-debug-vlrjg is created for debugging node...

Starting pod/worker01-debug ...

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

Pod IP: 192.168.50.13

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

sh-4.4#

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Verify the content for the chrony configuration file. The MC correctly applies the changes to the /etc/chrony.conf file.

sh-4.4# **cat /etc/chrony.conf**

pool 0.rhel.pool.ntp.org iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Remove the debug pod for the worker01 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

Removing debug pod ...

Temporary namespace openshift-debug-vlrjg was removed.

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

* 1. Create a debug pod for the worker02 node.

[student@workstation nodes-mco]$ **oc debug node/worker02**

...output omitted...

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Verify the content for the chrony configuration file. The MC does not change the content in the /etc/chrony.conf file.

sh-4.4# **cat /etc/chrony.conf**

# The Machine Config Operator manages this file.

server classroom.example.com iburst

stratumweight 0

driftfile /var/lib/chrony/drift

rtcsync

makestep 10 3

bindcmdaddress 127.0.0.1

bindcmdaddress ::1

...output omitted...

* 1. Remove the debug pod for the worker02 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

1. Manually modify the /etc/chrony.conf file on the worker01 node, and verify that the MCO detects a mismatch in the chrony configuration file.
   1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

...output omitted...

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Modify the content for the /etc/chrony.conf configuration file as follows. If the vi editor does not show the first line, then resize the window to show it.

pool **1.rhel.pool.ntp.org** iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Remove the debug pod for the worker01 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

* 1. Verify that the MCO marks the worker01 node with the degraded state.
  2. [student@workstation nodes-mco]$ **oc get mcp**
  3. NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...
  4. **custom** rendered-custom-... False True **True** 1 ...
  5. master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify the information for the custom MCP, and that the MCO detects a mismatch in the chrony configuration file.

[student@workstation nodes-mco]$ **oc describe mcp custom**

...output omitted...

Status:

Conditions:

...output omitted...

Last Transition Time: 2023-09-20T11:03:53Z

Message: **Node worker01 is reporting: "unexpected on-disk state validating against rendered-custom-009f7b72211eb0d37a8ba1d03a343e5d: content mismatch for file '/etc/chrony.conf'"**

Reason: 1 nodes are reporting degraded status on sync

Status: True

Type: NodeDegraded

Last Transition Time: 2023-09-20T11:03:53Z

Message:

Reason:

Status: True

Type: Degraded

...output omitted...

* 1. Get the name of the MCO daemon pod. The name for the daemon pod might be different on your system.

[student@workstation nodes-mco]$ **oc get pod -n openshift-machine-config-operator \**

**--field-selector spec.nodeName=worker01**

NAME READY STATUS RESTARTS AGE

***machine-config-daemon-jsrzm*** 2/2 Running 4 16d

* 1. Review the logs for the MCO daemon for more information about the configuration drift.

[student@workstation nodes-mco]$ **oc logs *machine-config-daemon-jsrzm* \**

**-n openshift-machine-config-operator**

...output omitted...

W0929 09:23:20.539413 2520 daemon.go:1763] current+desiredConfig is rendered-custom-11677dc3a3e84870c3a359ab1be4eafc but state is Degraded

I0929 09:23:20.858668 2520 rpm-ostree.go:394] Running captured: rpm-ostree kargs

E0929 09:23:20.905408 2520 on\_disk\_validation.go:207] **content mismatch for file "/etc/chrony.conf"** (-want +got):

bytes.Join({

"pool ",

**- "0",**

**+ "1",**

".rhel.pool.ntp.org iburst\ndriftfile /var/lib/chrony/drift\nmakest",

"ep 1.0 3\nrtcsync\nlogdir /var/log/chrony\n",

}, "")

E0929 09:23:20.905438 2520 daemon.go:589] Preflight config drift check failed: content mismatch for file "/etc/chrony.conf"

E0929 09:23:20.905457 2520 writer.go:200] Marking Degraded due to: content mismatch for file "/etc/chrony.conf"

* 1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

...output omitted...

sh-4.4# **chroot /host**

* 1. Fix the /etc/chrony.conf file.

pool **0.rhel.pool.ntp.org** iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Exit the debug pod.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

* 1. Verify that the MCO marks the node with the ready state, which might take a few minutes.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False **False** 1 ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

1. Remove the node-role.kubernetes.io/custom label for the worker01 node. After the MCO updates the node with the new configuration, delete the 99-zcustom-chrony MC and the custom-ntp MCP.
   1. Remove the node-role.kubernetes.io/custom label from the worker01 node.

[student@workstation nodes-mco]$ **oc label node/worker01 \**

**node-role.kubernetes.io/custom-**

node/worker01 unlabeled

* 1. Verify that no node uses the node-role.kubernetes.io/custom label.

[student@workstation nodes-mco]$ **oc get nodes -l node-role.kubernetes.io/custom=**

No resources found

* 1. Verify that the MCO applies the changes to the worker01 node. Wait until the MCO updates the node. The MCO finishes updating the node when the desiredConfig and currentConfig annotations point to the same rendered MC, and the state annotation is Done.

[student@workstation nodes-mco]$ **oc describe node worker01 \**

**| grep machineconfiguration**

machineconfiguration.openshift.io/controlPlaneTopology: HighlyAvailable

**machineconfiguration.openshift.io/currentConfig: rendered-custom-1167...eafc**

**machineconfiguration.openshift.io/desiredConfig: rendered-worker-d4f4...6774**

machineconfiguration.openshift.io/desiredDrain: drain-rendered-worker-...

machineconfiguration.openshift.io/lastAppliedDrain: drain-rendered-worker-...

machineconfiguration.openshift.io/reason:

**machineconfiguration.openshift.io/state: Working**

* 1. Verify that the MCP updated status is set to True.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... True False False **0** ...

master rendered-master-... True False False 3 ...

**worker** rendered-worker-... **True** False False **3** ...

* 1. Delete the 99-zcustom-chrony MC.

[student@workstation nodes-mco]$ **oc delete mc 99-zcustom-chrony**

machineconfig.machineconfiguration.openshift.io "99-zcustom-chrony" deleted

* 1. Delete the custom MCP.

### Warning

Remove the custom MCP only after the MCO applies the changes to the worker01 node.

If you remove the custom MCP before the MCO applies the changes to the node, then the MCP gets stuck in a degraded state, because the currentConfig MC of the node does not exist. Thus, none of the guided exercises from this point on will work.

To solve this issue, either recreate your entire cluster or apply the solution in <https://access.redhat.com/solutions/4970731>

[student@workstation nodes-mco]$ **oc delete mcp custom**

machineconfigpool.machineconfiguration.openshift.io "custom" deleted

* 1. Change to the student HOME directory.

[student@workstation nodes-mco]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish nodes-mco**

## **Guided Exercise: Node Configuration with the Machine Configuration Operator**

Label a cluster node and apply a machine configuration to it, without affecting other cluster nodes.

**Outcomes**

* Use the machine configuration operator (MCO) to modify the Network Time Protocol (NTP) client configuration.
* Create a machine configuration pool (MCP) and a machine configuration (MC).
* Observe the status of MC updates.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start nodes-mco**

**Instructions**

Your company requires you to configure an NTP server for the nodes in the cluster to synchronize services among them. The new NTP server is 0.rhel.pool.ntp.org.

In production environments, it is common to change the default NTP configuration to refer to NTP servers in the corporate LAN, instead of to public NTP servers from the internet, to prevent clock skew. However, in this exercise, you apply a new NTP configuration only to the worker01 node, to eliminate waiting time for node reboots when applying MCs, and to show how to apply MCs only to a subset of nodes by using labels.

To configure the NTP server in the worker01 node, create the custom MCP. The custom MCP must apply the MCs with the worker and custom labels to the nodes with the custom role. Then, create an MC for the NTP configuration with the custom label.

1. Connect to the OpenShift cluster and verify the roles for the nodes in the cluster. List the MCs that apply to the worker and custom roles.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. Change to the nodes-mco project.

[student@workstation ~]$ **oc project nodes-mco**

Now using project "nodes-mco" on server "https://api.ocp4.example.com:6443".

* 1. Verify the nodes in the cluster and their roles.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 109d v1.27.10+28ed2d7

master02 Ready control-plane,master 109d v1.27.10+28ed2d7

master03 Ready control-plane,master 109d v1.27.10+28ed2d7

worker01 Ready worker 21d v1.27.10+28ed2d7

worker02 Ready worker 21d v1.27.10+28ed2d7

worker03 Ready worker 21d v1.27.10+28ed2d7

* 1. List the MCs that apply to the worker roles.

[student@workstation ~]$ **oc get machineconfig \**

**-l machineconfiguration.openshift.io/role=worker**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

00-worker 52fe2...ebf97 3.4.0 12d

01-worker-container-runtime 52fe2...ebf97 3.4.0 12d

01-worker-kubelet 52fe2...ebf97 3.4.0 12d

97-worker-generated-kubelet 52fe2...ebf97 3.4.0 12d

98-worker-generated-kubelet 52fe2...ebf97 3.4.0 12d

99-worker-chrony-conf-override 3.2.0 12d

99-worker-generated-registries 52fe2...ebf97 3.4.0 12d

99-worker-ssh 3.2.0 12d

* 1. List the MCs that apply to the custom roles.

[student@workstation ~]$ **oc get machineconfig \**

**-l machineconfiguration.openshift.io/role=custom**

No resources found

1. Label the worker01 node with the node-role.kubernetes.io/custom label. Verify that the node has the correct label.
   1. Apply the node-role.kubernetes.io/custom label to the worker01 node.

[student@workstation ~]$ **oc label node/worker01 node-role.kubernetes.io/custom=**

node/worker01 labeled

* 1. Verify that the node has the correct label.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 33d v1.27.10+28ed2d7

master02 Ready control-plane,master 33d v1.27.10+28ed2d7

master03 Ready control-plane,master 33d v1.27.10+28ed2d7

worker01 Ready **custom**,worker 12d v1.27.10+28ed2d7

worker02 Ready worker 12d v1.27.10+28ed2d7

worker03 Ready worker 12d v1.27.10+28ed2d7

1. Create an MCP for the node with the custom label. The MCP must include the MCs for the worker and custom labels.
   1. Change to the ~/DO380/labs/nodes-mco directory.

[student@workstation ~]$ **cd ~/DO380/labs/nodes-mco**

* 1. Create the MCP custom resource (CR) YAML file. You can find an example for the CR in the ~/DO380/labs/nodes-mco/custom-mcp.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

name: **custom**

spec:

machineConfigSelector:

matchExpressions:

- key: machineconfiguration.openshift.io/role

operator: In

values: **[worker,custom]**

nodeSelector:

matchLabels:

**node-role.kubernetes.io/custom: ""**

* 1. Apply the configuration for the MCP CR.

[student@workstation nodes-mco]$ **oc create -f custom-mcp.yml**

machineconfigpool.machineconfiguration.openshift.io/custom created

* 1. Verify that the custom MCP is correctly created. If the UPDATING field is marked as true for the custom MCP, then wait until the MCO updates the node.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False False **1** ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify that the worker01 node is assigned for the custom MCP, and that the MCO applies the rendered MC to the node.

[student@workstation nodes-mco]$ **oc get events -n \**

**openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=custom**

... REASON OBJECT MESSAGE

... RenderedConfigGenerated machineconfigpool/custom rendered-custom-2f56....81d0 successfully generated ...

... **SetDesiredConfig** machineconfigpool/custom **Targeted node worker01** to MachineConfig rendered-custom-2f56...81d0

1. Because an OpenShift preconfigured MC configures the NTP client, verify the name for the MCs that modify the chrony configuration for the nodes with the worker label. Create an MC for the chrony configuration that applies to the custom label, with higher precedence than the preconfigured MC. The name for the MC must be 99-zcustom-chrony to have higher precedence.
   1. Verify the names for the MCs that modify the chrony configuration for the nodes with the worker label.

[student@workstation nodes-mco]$ **oc get mc --selector \**

**machineconfiguration.openshift.io/role=worker \**

**-o jsonpath="{range .items[\*]}{.metadata.name} \**

**{': '}{range .spec.config.storage.files[\*]}{.path} \**

**{' '}{end}{'\n'}{end}" | grep chrony**

**99-worker-chrony-conf-override**: /etc/chrony.conf

* 1. Create a chrony configuration file. You can find an example for the CR in the ~/DO380/labs/nodes-mco/chrony-mod.conf file.

pool 0.rhel.pool.ntp.org iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Encode the chrony configuration file in Base64 format. You use the encoded configuration file in a later step. The encoded configuration file would differ on your system.

[student@workstation nodes-mco]$ **base64 -w0 chrony-mod.conf; echo**

***cG9vbCAwLnJoZWwucG9...C92YXIvbG9nL2Nocm9ueQo=***

* 1. Create the 99-zcustom-chrony MC CR YAML file. Use the encoded chrony configuration file from a previous step. You can find an example for the CR in the ~/DO380/labs/nodes-mco/99-zcustom-chrony.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: **custom**

name: **99-zcustom-chrony**

spec:

config:

ignition:

version: 3.2.0

storage:

files:

- contents:

compression: ""

source: data:;base64,***cG9vbCAwLnJoZWwucG9...C92YXIvbG9nL2Nocm9ueQo=***

mode: 420

overwrite: true

path: /etc/chrony.conf

* 1. Apply the configuration for the MC CR.

[student@workstation nodes-mco]$ **oc create -f 99-zcustom-chrony.yml**

machineconfig.machineconfiguration.openshift.io/99-zcustom-chrony created

* 1. Verify that the 99-zcustom-chrony MC is correctly created.

[student@workstation nodes-mco]$ **oc get mc**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

...output omitted...

99-master-generated-registries 52fe26136643a946ff... 3.2.0 109d

99-master-ssh 3.2.0 109d

99-worker-chrony-conf-override 3.2.0 109d

99-worker-generated-registries 52fe26136643a946ff... 3.2.0 109d

99-worker-ssh 3.2.0 109d

**99-zcustom-chrony** 3.2.0 2m5s

...output omitted...

* 1. Verify that the MCP starts updating the resources.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... False **True** False **1** ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify that the MCO applies the rendered MC to the worker01 node.

[student@workstation nodes-mco]$ **oc get events \**

**-n openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=custom**

... REASON OBJECT MESSAGE

... RenderedConfigGenerated machineconfigpool/custom rendered-custom-2f56...81d0 successfully generated ...

... SetDesiredConfig machineconfigpool/custom Targeted node worker01 to MachineConfig rendered-custom-2f56...81d0

... **RenderedConfigGenerated** machineconfigpool/custom **rendered-custom-009f...3e5d successfully generated** ...

... **SetDesiredConfig** machineconfigpool/custom **Targeted node worker01** to MachineConfig rendered-custom-009f...3e5d

* 1. Wait a few minutes until the MCO finishes the update.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False False 1 ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

1. Verify that the MCO applies the chrony configuration to the worker01 node, and that the worker02 and worker03 nodes remain unaltered.
   1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

Temporary namespace openshift-debug-vlrjg is created for debugging node...

Starting pod/worker01-debug ...

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

Pod IP: 192.168.50.13

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

sh-4.4#

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Verify the content for the chrony configuration file. The MC correctly applies the changes to the /etc/chrony.conf file.

sh-4.4# **cat /etc/chrony.conf**

pool 0.rhel.pool.ntp.org iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Remove the debug pod for the worker01 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

Removing debug pod ...

Temporary namespace openshift-debug-vlrjg was removed.

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

* 1. Create a debug pod for the worker02 node.

[student@workstation nodes-mco]$ **oc debug node/worker02**

...output omitted...

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Verify the content for the chrony configuration file. The MC does not change the content in the /etc/chrony.conf file.

sh-4.4# **cat /etc/chrony.conf**

# The Machine Config Operator manages this file.

server classroom.example.com iburst

* 1. stratumweight 0

driftfile /var/lib/chrony/drift

rtcsync

makestep 10 3

bindcmdaddress 127.0.0.1

bindcmdaddress ::1

...output omitted...

* 1. Remove the debug pod for the worker02 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

1. Manually modify the /etc/chrony.conf file on the worker01 node, and verify that the MCO detects a mismatch in the chrony configuration file.
   1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

...output omitted...

* 1. Run the chroot /host command to use host binaries.

sh-4.4# **chroot /host**

* 1. Modify the content for the /etc/chrony.conf configuration file as follows. If the vi editor does not show the first line, then resize the window to show it.

pool **1.rhel.pool.ntp.org** iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Remove the debug pod for the worker01 node.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

|  |  |
| --- | --- |
|  | This command exit the chroot environment. |
|  | This command exit the debug pod. |

* 1. Verify that the MCO marks the worker01 node with the degraded state.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... False True **True** 1 ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

* 1. Verify the information for the custom MCP, and that the MCO detects a mismatch in the chrony configuration file.

[student@workstation nodes-mco]$ **oc describe mcp custom**

...output omitted...

Status:

Conditions:

...output omitted...

Last Transition Time: 2023-09-20T11:03:53Z

Message: **Node worker01 is reporting: "unexpected on-disk state validating against rendered-custom-009f7b72211eb0d37a8ba1d03a343e5d: content mismatch for file '/etc/chrony.conf'"**

Reason: 1 nodes are reporting degraded status on sync

Status: True

Type: NodeDegraded

Last Transition Time: 2023-09-20T11:03:53Z

Message:

Reason:

Status: True

Type: Degraded

...output omitted...

* 1. Get the name of the MCO daemon pod. The name for the daemon pod might be different on your system.

[student@workstation nodes-mco]$ **oc get pod -n openshift-machine-config-operator \**

**--field-selector spec.nodeName=worker01**

NAME READY STATUS RESTARTS AGE

***machine-config-daemon-jsrzm*** 2/2 Running 4 16d

* 1. Review the logs for the MCO daemon for more information about the configuration drift.

[student@workstation nodes-mco]$ **oc logs *machine-config-daemon-jsrzm* \**

**-n openshift-machine-config-operator**

...output omitted...

W0929 09:23:20.539413 2520 daemon.go:1763] current+desiredConfig is rendered-custom-11677dc3a3e84870c3a359ab1be4eafc but state is Degraded

I0929 09:23:20.858668 2520 rpm-ostree.go:394] Running captured: rpm-ostree kargs

E0929 09:23:20.905408 2520 on\_disk\_validation.go:207] **content mismatch for file "/etc/chrony.conf"** (-want +got):

bytes.Join({

"pool ",

**- "0",**

**+ "1",**

".rhel.pool.ntp.org iburst\ndriftfile /var/lib/chrony/drift\nmakest",

"ep 1.0 3\nrtcsync\nlogdir /var/log/chrony\n",

}, "")

E0929 09:23:20.905438 2520 daemon.go:589] Preflight config drift check failed: content mismatch for file "/etc/chrony.conf"

E0929 09:23:20.905457 2520 writer.go:200] Marking Degraded due to: content mismatch for file "/etc/chrony.conf"

* 1. Create a debug pod for the worker01 node.

[student@workstation nodes-mco]$ **oc debug node/worker01**

...output omitted...

sh-4.4# **chroot /host**

* 1. Fix the /etc/chrony.conf file.

pool **0.rhel.pool.ntp.org** iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

* 1. Exit the debug pod.

sh-4.4# **exit**

exit

sh-4.4# **exit**

exit

...output omitted...

* 1. Verify that the MCO marks the node with the ready state, which might take a few minutes.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... **True** False **False** 1 ...

master rendered-master-... True False False 3 ...

worker rendered-worker-... True False False 2 ...

1. Remove the node-role.kubernetes.io/custom label for the worker01 node. After the MCO updates the node with the new configuration, delete the 99-zcustom-chrony MC and the custom-ntp MCP.
   1. Remove the node-role.kubernetes.io/custom label from the worker01 node.

[student@workstation nodes-mco]$ **oc label node/worker01 \**

**node-role.kubernetes.io/custom-**

node/worker01 unlabeled

* 1. Verify that no node uses the node-role.kubernetes.io/custom label.

[student@workstation nodes-mco]$ **oc get nodes -l node-role.kubernetes.io/custom=**

No resources found

* 1. Verify that the MCO applies the changes to the worker01 node. Wait until the MCO updates the node. The MCO finishes updating the node when the desiredConfig and currentConfig annotations point to the same rendered MC, and the state annotation is Done.

[student@workstation nodes-mco]$ **oc describe node worker01 \**

**| grep machineconfiguration**

machineconfiguration.openshift.io/controlPlaneTopology: HighlyAvailable

**machineconfiguration.openshift.io/currentConfig: rendered-custom-1167...eafc**

**machineconfiguration.openshift.io/desiredConfig: rendered-worker-d4f4...6774**

machineconfiguration.openshift.io/desiredDrain: drain-rendered-worker-...

machineconfiguration.openshift.io/lastAppliedDrain: drain-rendered-worker-...

machineconfiguration.openshift.io/reason:

**machineconfiguration.openshift.io/state: Working**

* 1. Verify that the MCP updated status is set to True.

[student@workstation nodes-mco]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**custom** rendered-custom-... True False False **0** ...

master rendered-master-... True False False 3 ...

**worker** rendered-worker-... **True** False False **3** ...

* 1. Delete the 99-zcustom-chrony MC.

[student@workstation nodes-mco]$ **oc delete mc 99-zcustom-chrony**

machineconfig.machineconfiguration.openshift.io "99-zcustom-chrony" deleted

* 1. Delete the custom MCP.

### Warning

Remove the custom MCP only after the MCO applies the changes to the worker01 node.

If you remove the custom MCP before the MCO applies the changes to the node, then the MCP gets stuck in a degraded state, because the currentConfig MC of the node does not exist. Thus, none of the guided exercises from this point on will work.

To solve this issue, either recreate your entire cluster or apply the solution in <https://access.redhat.com/solutions/4970731>

[student@workstation nodes-mco]$ **oc delete mcp custom**

machineconfigpool.machineconfiguration.openshift.io "custom" deleted

* 1. Change to the student HOME directory.

[student@workstation nodes-mco]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish nodes-mco**

## **Node Configuration with Special Purpose Operators**

### **Objectives**

* Apply operating system settings to cluster nodes with higher-level operators instead of the low-level machine configuration operator.

### **Special Purpose Operators**

The Red Hat Enterprise Linux CoreOS (RHCOS) operating system that runs on each node in an OpenShift cluster is not administered with the traditional methods for Red Hat Enterprise Linux. To adapt to the paradigm shift of using RHCOS as an infrastructure operating system, learning the methods for managing RHCOS is a large part of gaining the required knowledge to manage OpenShift clusters.

Although the Machine Configuration Operator (MCO) can handle many of the preferred customizations for the operating system that runs on your nodes, other special purpose operators are available for higher-level configurations, such as to focus on specialized hardware tuning, cluster organization, and advanced optimizations.

OperatorHub is a library of shared operators that provides a curated catalog for identifying and installing available operators in a cluster. The CLI and web console approaches are available for installing these special purpose operators, including those operators that are not available in OperatorHub.

Each operator runs in a designated namespace, and uses CRDs and CRs to define the implementation. An operator's function might also extend to actions across other namespaces. Red Hat recommends running each special purpose operator within its own namespace and using a service account with the appropriate permissions. Although operators typically configure the namespace and service account as part of the installation process, some operators might require manual setup.

### **The Node Tuning Operator**

The Node Tuning Operator is a cluster operator that manages node-level optimizations by using the TuneD service and the tuned daemon. The operator manages the containerized tuned daemon for RHOCP as a Kubernetes daemon set. Creating custom TuneD profiles can enable granular customizations to cluster nodes. This approach helps the cluster nodes provide ideal environments for the running applications.

The Node Tuning Operator is installed by default in RHOCP clusters, and runs in the openshift-cluster-node-tuning-operator namespace. You can view the default TuneD profile for the cluster by running the following command:

[user@host ~]$ **oc get Tuned/default -o yaml \**

**-n openshift-cluster-node-tuning-operator**

You can view the applied TuneD profiles for each node with the following command:

[user@host ~]$ **oc get profile -n openshift-cluster-node-tuning-operator**

When customizing a cluster node, create a TuneD profile and specify the optimized parameters from a TuneD plug-in. Each of the TuneD plug-ins, such as cpu, sysctl, or vm, provides various customizations for the cluster nodes. Consult the TuneD documentation from Red Hat to explore the full list of TuneD profiles.

### Note

For more information about TuneD plug-ins, refer to <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index#available-tuned-plug-ins_customizing-tuned-profiles>

### **The Node Feature Discovery Operator**

Node Feature Discovery (NFD) is a Kubernetes add-on in OperatorHub for installation, to expose node-level information. The NFD operator detects hardware details, such as processor architecture, specialized PCI hardware such as GPUs or networking cards, and other system configurations on your cluster nodes. The NFD operator runs inside the openshift-nfd namespace for these operations.

### Note

Due to the minimal environment within a classroom cluster, the special purpose operators might not be available in OperatorHub in this course.

The operator profiles each of the cluster nodes and collects information about the hardware and system attributes. This operator collects information about connected PCI devices, CPU specifications, kernel information, the operating system version, and other hardware information. The operator adds labels to the cluster nodes with the gathered information, to aid deployment placement onto nodes that provide the required specialized hardware and environments.

Consider a cluster that added high-throughput networking cards to a subset of its nodes. After the NFD operator profiles and labels these nodes, applications that require the higher network throughput can target the labeled nodes with this hardware.

### **The Kernel Module Management Operator**

The Kernel Module Management (KMM) Operator, which is available for installation in OperatorHub, manages, builds, signs, and deploys out-of-tree kernel modules and device plug-ins on OpenShift Container Platform clusters. This operator is useful for adding extra, alternative, or custom modules to the nodes in your cluster that run workloads that use those features.

Many applications, especially those applications that adopt later technologies before a full integration into modern operating systems is available, can require customizing the kernel modules for the cluster nodes. Developers who use Artificial Intelligence (AI) and Machine Learning (ML) applications often author custom kernel modules to optimize the computational operations.

### Note

After deployment, kernel modules can run any type of operation. For this reason, carefully consider security implications and permissions when adding kernel modules.

You can install the KMM by searching in OperatorHub to add the operator to a cluster. The KMM uses the openshift-kmm namespace to manage kernel module deployments. A Module CRD specifies the kernel that can be added to one or more cluster nodes. A corresponding Module CR applies this CRD to install the required kernel version for each cluster node.

### **Vendor-provided Operators**

Many hardware vendors author special purpose operators for OpenShift to provide unique functions that correspond to a particular product that the company sells. Companies such as IBM, Nvidia, and Intel author and distribute these operators to help cluster administrators in using the specialized vendor hardware that is installed on cluster nodes. Each of these special purpose operators can vary in availability, and might or might not be available in OperatorHub. When adding hardware to cluster nodes, consider vendors who provide OpenShift operators that help with the management and function of cluster nodes.

### References

For more information about the Node Tuning Operator, refer to the Node Tuning Operator section in the Red Hat OpenShift Container Platform 4.14 Working with Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-node-tuning-operator>

For more information about the Node Feature Discovery Operator, refer to the Node Feature Discovery Operator section in the Red Hat OpenShift Container Platform 4.14 Specialized Hardware and Driver Enablement documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/specialized_hardware_and_driver_enablement/index#node-feature-discovery-operator>

For more information about the Kernel Module Management Operator, refer to the Kernel Module Management Operator section in the Red Hat OpenShift Container Platform 4.14 Specialized Hardware and Driver Enablement documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/specialized_hardware_and_driver_enablement/index#kernel-module-management-operator>

For more information about TuneD, refer to the Getting Started with TuneD section in the Red Hat Enterprise Linux 8 Monitoring and Managing System Status and Performance documentation at <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index>

## **Guided Exercise: Node Configuration with Special Purpose Operators**

Label a cluster node and apply a TuneD configuration to it, without affecting other cluster nodes.

**Outcomes**

Use the Node Tuning Operator to create a TuneD profile that configures a compute node setting for optimizing a database workload.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start nodes-operators**

**Instructions**

The software vendor for the database that is used in a company application recommends adjusting a specific setting to improve performance. The suggestion is to disable the transparent\_hugepages setting, which is enabled by default, on the compute node where the database runs, to improve memory management.

Your company requests that you configure an RHOCP cluster node for the database that requires this memory management adjustment. Create a TuneD profile through the Node Tuning Operator, which configures the worker01 node to run the database with the customization for the node memory management, which the software vendor recommends.

Verify the initial setting and updated value on the node by inspecting the value in the /sys/kernel/mm/transparent\_hugepage/enabled file.

1. Connect to the OpenShift cluster and verify the roles for the nodes in the cluster.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

1. Verify that the worker01 node has the stabledb label.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 47d v1.25.7+eab9cc9

master02 Ready control-plane,master 47d v1.25.7+eab9cc9

master03 Ready control-plane,master 47d v1.25.7+eab9cc9

worker01 Ready **stabledb**,worker 16h v1.25.7+eab9cc9

worker02 Ready worker 16h v1.25.7+eab9cc9

worker03 Ready worker 16h v1.25.7+eab9cc9

1. View the pods that are running in the openshift-cluster-node-tuning-operator namespace. View the existing TuneD profiles and configuration for the cluster.
   1. View the running pods in the namespace.

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

**openshift-cluster-node-tuning-operator**

NAME READY STATUS RESTARTS AGE

cluster-node-tuning-operator-... 1/1 Running 2 47d

tuned-45hlf 1/1 Running 1 18h

tuned-glr54 1/1 Running 1 18h

...output omitted...

* 1. View the current tuned profiles in use for each cluster node.

[student@workstation ~]$ **oc get profile -n openshift-cluster-node-tuning-operator**

NAME TUNED APPLIED DEGRADED AGE

master01 openshift-control-plane True False 47d

master02 openshift-control-plane True False 47d

master03 openshift-control-plane True False 47d

worker01 openshift-node True False 20h

worker02 openshift-node True False 20h

worker03 openshift-node True False 20h

* 1. View the TuneD configuration in the cluster.

[student@workstation ~]$ **oc get tuneds.tuned.openshift.io \**

**-n openshift-cluster-node-tuning-operator -o yaml**

apiVersion: v1

items:

- apiVersion: tuned.openshift.io/v1

**kind: Tuned**

metadata:

creationTimestamp: "2023-10-04T10:12:26Z"

generation: 1

name: default

namespace: openshift-cluster-node-tuning-operator

resourceVersion: "8714"

uid: a4b818f2-dab5-4091-9beb-c763d216a75a

spec:

profile:

- **data: |**

**[main]**

**summary=Optimize systems running OpenShift (provider specific parent profile)**

**include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift**

name: openshift

recommend:

- **match:**

**- label: node-role.kubernetes.io/master**

**- label: node-role.kubernetes.io/infra**

operand:

tunedConfig: {}

**priority: 30**

...output omitted...

1. Create a debug pod to connect to the worker01 node and inspect the current setting for the transparent\_hugepages setting.
   1. Open a new terminal window and create a debug pod to connect to the worker01 node.

[student@workstation ~]$ **oc debug node/worker01**

...output omitted...

sh-4.4#

* 1. View the current transparent\_hugepages setting by displaying the value in the setting enabled file. The [always] indicator denotes that the setting is always enabled.

sh-4.4# **cat /sys/kernel/mm/transparent\_hugepage/enabled**

[always] madvise never

* 1. Log out and remove the debug pod for the worker01 node.

sh-4.4# **exit**

...output omitted...

1. Navigate to the exercise directory and create a TuneD profile CR file named stabledbCR.yml that defines a TuneD profile stabledb-tuning to disable the transparent\_hugepage setting for nodes with the stabledb label. You can find an incomplete example for the CR in the ~/DO380/labs/nodes-operators/stabledbCR.yml file.
   1. Change to the working directory for the exercise.

[student@workstation ~]$ **cd ~/DO380/labs/nodes-operators**

* 1. Create the stabledbCR.yml file and define the resource kind as Tuned for the metadata name of stabledb-tuning.

apiVersion: tuned.openshift.io/v1

**kind: Tuned**

metadata:

**name: stabledb-tuning**

namespace: openshift-cluster-node-tuning-operator

* 1. Specify the profile name as stabledb-tuning. You can add an optional description in the summary key for the profile.

spec:

profile:

**- name: stabledb-tuning**

data: |

[main]

**summary=Worker optimization disabling transparent\_hugepages.**

* 1. Supply the arguments for the customization that disables the transparent\_hugepages value.

**[vm]**

**transparent\_hugepage=never**

* 1. Add the recommend section to the CR to target the node with the stabledb label.

recommend:

- match:

**- label: node-role.kubernetes.io/stabledb**

* 1. Finally, provide the priority that is equal to or less than the currently applied profile. Profiles with lower priority values take precedence over ones with a higher value.

**priority: 30**

* 1. Verify that the file resembles the following CR:

apiVersion: tuned.openshift.io/v1

kind: Tuned

metadata:

name: **stabledb-tuning**

namespace: openshift-cluster-node-tuning-operator

spec:

profile:

- name: **stabledb-tuning**

data: |

[main]

summary=Worker optimization disabling transparent\_hugepages.

[vm]

transparent\_hugepage=**never**

recommend:

- match:

- label: node-role.kubernetes.io/stabledb

priority: 30

profile: **stabledb-tuning**

1. Apply the TuneD CR and verify that the profile is applied to the worker01 node.
   1. Apply the TuneD CR.

[student@workstation nodes-operators]$ **oc apply -f stabledbCR.yml**

tuned.tuned.openshift.io/stabledb-tuning created

* 1. Verify the stabled-tuning profile for the worker01 node.

[student@workstation nodes-operators]$ **oc get profile \**

**-n openshift-cluster-node-tuning-operator**

NAME TUNED APPLIED DEGRADED AGE

master01 openshift-control-plane True False 47d

master02 openshift-control-plane True False 47d

master03 openshift-control-plane True False 47d

worker01 **stabledb-tuning** True False 20h

worker02 openshift-node True False 20h

worker03 openshift-node True False 20h

1. From the second terminal window, re-create the debug pod to connect to the worker01 node and verify the updated transparent\_hugepages setting. This setting should reflect the changes from the new TuneD profile and no longer be enabled.
   1. Open a new terminal window and create a debug pod to connect to the worker01 node.

[student@workstation nodes-operators]$ **oc debug node/worker01**

...output omitted...

sh-4.4#

* 1. View the current setting for transparent\_hugepages by displaying the value in the setting enabled file. The [never] value denotes that the setting is disabled.

sh-4.4# **cat /sys/kernel/mm/transparent\_hugepage/enabled**

always madvise [never]

* 1. Log out and remove the debug pod for the worker01 node.

sh-4.4# **exit**

...output omitted...

1. Change to the student HOME directory.

[student@workstation nodes-review]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish nodes-operators**

## **Lab: Cluster Partitioning**

Apply operating system settings to cluster nodes with the machine configuration operator, and with higher-level operators instead of the low-level machine configuration operator.

**Outcomes**

* Create machine configuration pools (MCPs) to apply custom configurations to some nodes in the cluster.
* Use the node tuning operator (NTO) to enable non-uniform memory access (NUMA) balancing.
* Use the machine configuration operator (MCO) to enable a real-time kernel.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Your company has a workload that requires many CPUs and much memory. Thus, the bandwidth between the CPUs and the memory is limiting the performance of the workload. In your cluster, the nodes with the numa role have NUMA hardware systems. These nodes perform best when the threads of their processes are accessing memory on the same NUMA node as where the threads are scheduled. Thus, your company requires you to enable the NUMA balancing in the kernel through the NTO for the nodes with that role. In this classroom environment, only one node has the numa role due to the classroom size. However, no nodes in the cluster have real NUMA hardware systems.

Your company also needs to deploy a workload for a 5G core user-plane function. This workload requires a real-time kernel to ensure low latency. Although using a PerformanceProfile custom resource (CR) is the recommended approach for configuring low-latency workloads which require real-time kernels, in this exercise you must use the MCO instead to practice by using its syntax. The NTO creates, among other resources, a similar machine configuration (MC) to the one that you create manually in the exercise. Although in production environments, this application type is deployed in multiple nodes and the kernel configuration must be applied to all of them, in this exercise you apply the configuration only to one node due to the classroom size.

Use the admin user with redhatocp as the password. For the resources that you must create, you can use the incomplete CR YAML files in the ~/DO380/labs/nodes-review directory.

1. List the labels for the nodes in the cluster. Verify that the worker01 node has the numa role. Create the numa-bal MCP to include in the pool the nodes with the numa role. This MCP must select MCs with the numa-bal and worker labels. You can find an incomplete example for the MCP CR in the ~/DO380/labs/nodes-review/numa-mcp.yml file.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. List the labels for the nodes. Verify that the worker01 node has the numa role.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 49d v1.25.7+eab9cc9

master02 Ready control-plane,master 49d v1.25.7+eab9cc9

master03 Ready control-plane,master 49d v1.25.7+eab9cc9

**worker01** Ready **numa**,worker 2d4h v1.25.7+eab9cc9

worker02 Ready worker 2d4h v1.25.7+eab9cc9

worker03 Ready worker 2d4h v1.25.7+eab9cc9

* 1. Change to the ~/DO380/labs/nodes-review directory.

[student@workstation ~]$ **cd ~/DO380/labs/nodes-review**

* 1. Create the MCP CR YAML file. You can find an incomplete example for the CR in the ~/DO380/labs/nodes-review/numa-mcp.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

name: **numa-bal**

spec:

machineConfigSelector:

matchExpressions:

- key: machineconfiguration.openshift.io/role

operator: In

values: **[worker,numa-bal]**

nodeSelector:

matchLabels:

**node-role.kubernetes.io/numa: ""**

* 1. Apply the configuration for the MCP CR.

[student@workstation nodes-review]$ **oc create -f numa-mcp.yml**

machineconfigpool.machineconfiguration.openshift.io/numa-bal created

* 1. Verify that the numa-bal MCP is correctly created. Wait until the MCO updates the node and marks the UPDATED field as True. You might have to repeat this command many times.

[student@workstation nodes-review]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

master rendered-master-... True False False 3 ...

**numa-bal** rendered-numa-ba... **True** False False **1** ...

worker rendered-worker-... True False False 2 ...

* 1. Verify that the worker01 node is assigned for the numa-bal MCP, and that the MCO applies the rendered MC to the node.

[student@workstation nodes-review]$ **oc get events \**

**-n openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=numa-bal**

... REASON OBJECT MESSAGE

... RenderedConfigGenerated machineconfigpool/numa-bal rendered-numa-bal-2f56...81d0 successfully generated ...

... SetDesiredConfig machineconfigpool/numa-bal Targeted node worker01 to MachineConfig rendered-numa-bal-2f56...81d0

... RenderedConfigGenerated machineconfigpool/numa-bal rendered-numa-bal-2f56...81d0 successfully generated ...

... **SetDesiredConfig** machineconfigpool/numa-bal **Targeted node worker01** to MachineConfig rendered-numa-bal-2f56...81d0

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the kernel-numabal TuneD profile CR to enable NUMA balancing. The TuneD profile CR applies to all the nodes in the numa-bal MCP, and includes the default openshift-node TuneD profile. You can find an incomplete example for the TuneD profile CR in the ~/DO380/labs/nodes-review/numa-tuned.yml file. Verify that the NUMA balancing kernel parameter is enabled in the node. You can verify the NUMA balancing by running the sysctl command on the node and checking the kernel.numa\_balancing parameter. Verify that the NUMA balancing feature is disabled on other nodes in the cluster.
   1. Create the TuneD profile CR YAML file. You can find an incomplete example for the CR in the ~/DO380/labs/nodes-review/numa-tuned.yml file.

apiVersion: tuned.openshift.io/v1

kind: Tuned

metadata:

name: **kernel-numabal**

namespace: openshift-cluster-node-tuning-operator

spec:

profile:

- data: |

[main]

summary=Custom NUMA balancing profile

include=**openshift-node**

...output omitted...

recommend:

- machineConfigLabels:

**machineconfiguration.openshift.io/role: "numa-bal"**

priority: 30

profile: **kernel-numabal**

* 1. Apply the configuration for the TuneD CR.

[student@workstation nodes-review]$ **oc create -f numa-tuned.yml**

tuned.tuned.openshift.io/kernel-numabal created

* 1. View the current TuneD profiles in use for each cluster node.

[student@workstation nodes-review]$ **oc get profile \**

**-n openshift-cluster-node-tuning-operator**

NAME TUNED APPLIED DEGRADED AGE

master01 openshift-control-plane True False 49d

master02 openshift-control-plane True False 49d

master03 openshift-control-plane True False 49d

**worker01** **kernel-numabal** True False 13m

worker02 openshift-node True False 2d4h

worker03 openshift-node True False 2d4h

* 1. Create a debug pod for the worker01 node.

[student@workstation nodes-review]$ **oc debug node/worker01**

...output omitted...

sh-4.4#

* 1. Run the sysctl command and verify that the kernel.numa\_balancing parameter is enabled.

sh-4.4# **sysctl -n kernel.numa\_balancing**

1

* 1. Remove the debug pod for the worker01 node.

sh-4.4# **exit**

...output omitted...

* 1. Create a debug pod for the worker02 node.

[student@workstation nodes-review]$ **oc debug node/worker02**

...output omitted...

sh-4.4#

* 1. Run the sysctl command and verify that the kernel.numa\_balancing parameter is disabled.

sh-4.4# **sysctl -n kernel.numa\_balancing**

0

* 1. Remove the debug pod for the worker02 node.

sh-4.4# **exit**

...output omitted...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Label the worker02 node with the kernel-rt role. Create the kernel-rt MCP to include the nodes with the kernel-rt role in the pool. You can find an incomplete example for the MCP CR in the ~/DO380/labs/nodes-review/kernel-mcp.yml file.
   1. Add the kernel-rt role to the worker02 node.

[student@workstation nodes-review]$ **oc label node/worker02 \**

**node-role.kubernetes.io/kernel-rt=**

* 1. List the labels for the nodes. Verify that the worker02 node has the kernel-rt role.

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

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 49d v1.25.7+eab9cc9

master02 Ready control-plane,master 49d v1.25.7+eab9cc9

master03 Ready control-plane,master 49d v1.25.7+eab9cc9

worker01 Ready numa,worker 2d4h v1.25.7+eab9cc9

**worker02** Ready **kernel-rt**,worker 2d4h v1.25.7+eab9cc9

worker03 Ready worker 2d4h v1.25.7+eab9cc9

* 1. Create the MCP CR YAML file. You can find an incomplete example for the CR in the ~/DO380/labs/nodes-review/kernel-mcp.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfigPool

metadata:

name: kernel-rt

spec:

machineConfigSelector:

matchExpressions:

- key: machineconfiguration.openshift.io/role

operator: In

values: **[worker,kernel-rt]**

nodeSelector:

matchLabels:

**node-role.kubernetes.io/kernel-rt: ""**

* 1. Apply the configuration for the MCP CR.

[student@workstation nodes-review]$ **oc create -f kernel-mcp.yml**

machineconfigpool.machineconfiguration.openshift.io/kernel-rt created

Verify that the kernel-rt MCP is correctly created. If the UPDATING field is marked as true for the kernel-rt MCP, then wait until the MCO finishes updating the node.

[student@workstation nodes-review]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**kernel-rt** rendered-kernel-... **True** False False 1 ...

master rendered-master-... True False False 3 ...

numa-bal rendered-numa-ba... True False False 1 ...

worker rendered-worker-... True False False 1 ...

* 1. Verify that the worker02 node is assigned for the kernel-rt MCP, and that the MCO applies the rendered MC to the node.

[student@workstation nodes-review]$ **oc get events \**

**-n openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=kernel-rt**

... REASON OBJECT MESSAGE

...output omitted...

... RenderedConfigGenerated machineconfigpool/kernel-rt rendered-kernel-rt-2f56...81d0 successfully generated ...

... **SetDesiredConfig** machineconfigpool/kernel-rt **Targeted node worker02** to MachineConfig rendered-kernel-rt-2f56...81d0

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Use the oc explain command to find the MC parameter that changes the default kernel to a real-time kernel. Create the 99-kernel-realtime MC to change the default kernel to a real-time kernel for the nodes in the kernel-rt pool. You can find an incomplete example for the MC CR in the ~/DO380/labs/nodes-review/kernel-mc.yml file. Verify that the kernel for the worker02 node is a real-time kernel. You can verify that the node uses a real-time kernel by using the oc get nodes -o wide command and verifying that the kernel version shows the rt string.
   1. Find the MC parameter that changes the default kernel to a real-time kernel.

[student@workstation nodes-review]$ **oc explain mc.spec**

KIND: MachineConfig

VERSION: machineconfiguration.openshift.io/v1

...output omitted...

FIELDS:

...output omitted...

**kernelType** <string>

Contains which kernel we want to be running like default (traditional),

realtime

osImageURL <string>

OSImageURL specifies the remote location that will be used to fetch the OS

* 1. Create the YAML file for the MC CR. You can find an incomplete example for the CR in the ~/DO380/labs/nodes-review/kernel-mc.yml file.

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

name: **99-kernel-realtime**

labels:

**machineconfiguration.openshift.io/role: "kernel-rt"**

spec:

kernelType: realtime

* 1. Apply the configuration for the MC CR.

[student@workstation nodes-review]$ **oc create -f kernel-mc.yml**

machineconfig.machineconfiguration.openshift.io/99-kernel-realtime created

* 1. Verify that the 99-kernel-realtime MC is correctly created.

[student@workstation nodes-review]$ **oc get mc**

NAME GENERATEDBYCONTROLLER IGNITIONVERSION AGE

...output omitted...

01-worker-kubelet 52fe26136643a946ff... 3.2.0 49d

99-infra-chrony-conf-override 3.2.0 49d

**99-kernel-realtime**

99-master-chrony-conf-override 3.2.0 49d

99-master-generated-registries 52fe26136643a946ff... 3.2.0 49d

...output omitted...

* 1. Verify that the MCP starts updating the resources.

[student@workstation nodes-review]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**kernel-rt** rendered-kernel-rt... False **True** False **1** ...

master rendered-master-... True False False 3 ...

numa-bal rendered-numa-bal... True False False 1 ...

worker rendered-worker-... True False False 1 ...

* 1. Verify that OpenShift applies the rendered MC to the worker02 node.

[student@workstation nodes-review]$ **oc get events \**

**-n openshift-machine-config-operator \**

**--sort-by='{.lastTimestamp}' \**

**--field-selector involvedObject.name=kernel-rt**

... REASON OBJECT MESSAGE

...output omitted...

... RenderedConfigGenerated machineconfigpool/kernel-rt rendered-kernel-rt-2f56...81d0 successfully generated ...

... SetDesiredConfig machineconfigpool/kernel-rt Targeted node worker02 to MachineConfig rendered-kernel-rt-2f56...81d0

... **RenderedConfigGenerated** machineconfigpool/kernel-rt **rendered-kernel-rt-f6ab...234c** successfully generated ...

... **SetDesiredConfig** machineconfigpool/kernel-rt **Targeted node worker02** to MachineConfig rendered-kernel-rt-f6ab...234c

* 1. Verify that the MCO updates the node. If the UPDATING field is marked as true for the kernel-rt MCP, then wait until the MCO finishes updating the node.

[student@workstation nodes-review]$ **oc get mcp**

NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT ...

**kernel-rt** rendered-kernel-rt... **True** False False 1 ...

master rendered-master-... True False False 3 ...

numa-bal rendered-numa-bal... True False False 1 ...

worker rendered-worker-... True False False 1 ...

* 1. Verify that the worker02 node is using a real-time kernel. You can do so by verifying that the kernel version contains the rt string.

[student@workstation nodes-review]$ **oc get nodes -o wide**

NAME STATUS ROLES ... KERNEL-VERSION

master01 Ready control-plane,master ... 5.14.0-284.41.1.el9\_2.x86\_64

master02 Ready control-plane,master ... 5.14.0-284.41.1.el9\_2.x86\_64

master03 Ready control-plane,master ... 5.14.0-284.41.1.el9\_2.x86\_64

worker01 Ready numa,worker ... 5.14.0-284.41.1.el9\_2.x86\_64

**worker02** Ready kernel-rt,worker ... **5.14.0-284.41.1.rt14.326.el9\_2.x86\_64**

worker03 Ready worker ... 5.14.0-284.41.1.el9\_2.x86\_64

* 1. Change to the student HOME directory.

[student@workstation nodes-review]$ **cd**

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

**Evaluation**

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

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

# **Chapter 4.  Pod Scheduling**

[Pod Scheduling Concepts](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04)

[Quiz: Pod Scheduling Concepts](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s02)

[Quiz: Pod Scheduling Scenarios](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s03)

[Node Selectors and Taints](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s04)

[Guided Exercise: Node Selectors and Taints](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s05)

[High Availability with Affinity Rules and Pod Disruption Budgets](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s06)

[Guided Exercise: High Availability with Affinity Rules and Pod Disruption Budgets](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s07)

[Lab: Pod Scheduling](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s08)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch04s09)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Configure workloads to run on a dedicated set of cluster nodes and prevent other workloads from using those cluster nodes. |
| **Sections** | * Pod Scheduling Concepts (and Matching Quiz) * Pod Scheduling Scenarios (Quiz) * Node Selectors and Taints (and Guided Exercise) * High Availability with Affinity Rules and Pod Disruption Budgets (and Guided Exercise) |
| **Lab** | * Pod Scheduling |

## **Pod Scheduling Concepts**

### **Objectives**

* Describe the main settings that affect pod placement on Kubernetes cluster nodes.

### **Pod Scheduling**

Pod scheduling is the process of determining in which node to place a pod in the OpenShift cluster.

The OpenShift built-in scheduler identifies the most suitable node for the pods when you create them. The default scheduler meets the needs of most OpenShift users.

However, in some situations, OpenShift administrators might want more control over which node is used for pod placement. You can use the OpenShift advanced scheduling features to configure pods to run on a particular node or alongside a specific pod.

OpenShift includes the following advanced scheduling features:

* Scheduler profiles, to control how OpenShift schedules pods on nodes.
* Pod affinity rules, to keep sets of pods close to each other, on the same nodes. For example, you can run a REST service and its database on the same node to minimize network latency.
* Pod anti-affinity rules, to keep sets of pods far away from each other, on different nodes. For example, you can run replica pods of the same deployment in different nodes, so that if a node fails, then you do not lose all the pods for the workload.
* Node affinity, to keep sets of pods running on the same group of nodes. For example, you can configure a pod to run on nodes with a specific CPU.
* Node selectors, to schedule pods to a specific set of nodes. For example, you can schedule a pod to a node that provides special hardware that the pod needs.
* Taints and tolerations, to avoid scheduling pods to a specific set of nodes. For example, you can block a pod to run on a node that is reserved for OpenShift cluster services or control plane services.

OpenShift also includes the pod disruption budget resource, which controls how many instances can be down at the same time during voluntary disruptions, such as when scaling down, updating applications, or draining a node for maintenance.

|  |
| --- |
|  |

Figure 4.1: OpenShift advanced scheduling features

The previous diagram shows some of the advanced scheduling features. For example, the pod with a node selector for the CPU=fast label can be placed only on the second node, which contains the CPU=fast label.

For a pod that has a preferred node affinity rule for the CPU=fast label, OpenShift first tries to schedule it on the node with the CPU=fast label. If OpenShift cannot place the pod on that node, then OpenShift schedules the pod on another node.

If you create a deployment with the app=custom label and a pod anti-affinity rule for the same label, then OpenShift places the replicas for the deployment on different nodes.

If you taint a node with the type=mission-critical label, then OpenShift can schedule a pod on that node only if the pod has the correct toleration that matches the taint.

It is common in production environments to use more than one advanced scheduling feature at a time. For example, your cluster includes a workload that has a node selector to get GPU-enabled nodes, and pod anti-affinity rules to avoid downtime in the event of missing a node. The same workload also has a pod disruption budget to specify accepted degradation during a cluster upgrade, so the workload could run slower but still be available to its users.

### **Default Pod Scheduler**

The default OpenShift pod scheduler determines the placement of pods onto nodes within the cluster. The scheduler reads data from the pod and identifies a suitable node based on configured profiles. After identifying the most suitable node, the scheduler creates a binding that associates the pod with a specific node, without modifying the pod.

The OpenShift default pod scheduler determines the placement of a pod onto a particular node according to the following procedure:

1. Node filtering: The OpenShift scheduler filters the nodes based on the configured constraints or requirements by means of functions called predicates. These predicates include available CPU capacity on the node to satisfy a pod's CPU resource request, free ports, or volume availability, among others.
2. Prioritize the filtered list of nodes: In this step, the scheduler assesses each node by using a set of priority or scoring functions, and assigns each node a score from 0 to 10. A score of 0 indicates a poor fit, and a score of 10 indicates an excellent fit for hosting the pod. Additionally, OpenShift administrators can assign a numeric weight to each scoring function in the scheduler's configuration. With this weight attribute, administrators can prioritize certain scoring functions.
3. Select the best node: OpenShift sorts the nodes based on their scores and selects the node with the highest score. If multiple nodes receive the same score, then OpenShift selects one node randomly.

If a pod does not specify its resource requests, then the scheduler could place it on a node that is already full, which could lead to poor performance or even killing the pod if the node is out of memory.

#### **Scheduler Profiles**

The OpenShift scheduler profile controls how OpenShift schedules pods onto nodes. You can set the scheduler profile by using the oc edit scheduler cluster command and modifying the spec.profile parameter.

The following scheduler profiles are available:

**LowNodeUtilization**

By using this profile, OpenShift distributes pods evenly across nodes to ensure a low resource usage per node. This profile provides the default scheduler behavior.

**HighNodeUtilization**

With this profile, OpenShift places as many pods as possible onto as few nodes as possible. This profile minimizes the node count but increases the resource usage per node.

**NoScoring**

This profile aims for quick scheduling cycles by disabling all the score plug-ins. Thus, by using this profile you sacrifice better scheduling decisions for faster ones.

### **Advanced Pod Scheduling**

In some situations, you might want more control over which node is used for pod placement. You can use the following OpenShift advanced scheduling features to configure pods to run on a particular node or alongside a specific pod.

#### **Node Selectors**

Because a node selector specifies target node labels on your pod, OpenShift schedules the pod only on nodes that match the target labels. Thus, with a node selector, you must define the target labels in the pod specification file and label the nodes with those labels.

You can use node selectors to place pods on specific nodes. You can also define cluster-wide node selectors to place pods on specific nodes anywhere in the cluster, and define project node selectors to place pods in a project on specific nodes.

For example, you can label nodes in your cluster by using their geographical location. Then, application developers can use node selectors to deploy pods on nodes in a geographical region for availability and latency.

#### **Affinity and Anti-affinity Rules**

**Affinity**

Affinity refers to a pod property that controls the pod preference for a node.

**Anti-affinity**

Anti-affinity is a pod property that restricts the placement of the pod on a node.

You can use affinity and anti-affinity rules to control the following scenarios:

* The node to place a pod in, which is called node affinity.
* The node to place a pod in, relative to other pods, which is called pod affinity.

**Node affinity**

With node affinity, you can specify in a pod the affinity in relation to a group of nodes. For example, you can configure a pod to run on nodes with a specific CPU or in a specific availability zone.

Node affinity, which is conceptually similar to node selectors, provides a more flexible way to specify constraints on node selection. Whereas node selectors specify the target nodes by using a set of label key-value pairs, node affinity enables a conditional approach with logical operators in the matching process.

With node affinity, you can define required and preferred rules. Because required rules work similarly to node selectors, the OpenShift scheduler schedules the pod only if the conditions are met. By using preferred rules, the OpenShift scheduler tries to schedule the pod in a node that meets the rules. If the scheduler does not find a node, then the scheduler schedules the pod on a node that does not meet the rules.

**Pod affinity and anti-affinity**

With pod affinity and pod anti-affinity, you can control the nodes for your pod based on the labels of other pods.

By using pod affinity rules, you can locate a pod on the same group of nodes as other pods based on its label selector. As an example, you can configure the scheduler to place pods of two services onto the same group of nodes, because the services communicate with each other regularly.

By using pod anti-affinity rules, you can prevent the scheduler from locating a pod on the same group of nodes as other pods based on its label selector. For example, you can configure the scheduler to prevent a pod of a particular service from being on the same nodes as pods of another service, because the two services interfere and reduce the performance of the services. Another typical scenario is to use node topology labels, such as availability zones in a cloud provider or racks in a data center, and to use anti-affinity rules to avoid putting pods of the same workload on the same failure domain.

Pod affinity, pod anti-affinity rules, and node affinity rules each have two types: required and preferred.

Required rules must be met before the scheduler places a pod on a node. If you use required rules and those rules are not met, then the scheduler cannot find an appropriate node to place the pod.

Preferred rules specify conditions that the scheduler tries to enforce, but the scheduler still schedules the pod if it cannot find a matching node.

You can configure pod and node affinity and anti-affinity rules in the pod specification YAML files by setting the spec.config.affinity parameter. For that parameter, you can set the nodeAffinity, podAffinity, and podAntiAffinity properties, and specify a required rule, or a preferred rule, or both. If you specify both required and preferred rules, then the node must first meet the required rules and after the scheduler attempts to meet the preferred rules.

#### **Taints and Tolerations**

You can configure taints on nodes so they schedule a pod only if the pod has a matching toleration. The node with a taint repels all the pods except those pods that have a toleration for that taint. For example, OpenShift automatically taints the control plane nodes, so that the pods that manage the control plane can be scheduled on them. However, the control plane nodes reject any other data plane pods that users deploy.

You can apply taints to a node in the node specification YAML file by setting the spec.taints parameter, and apply tolerations to a pod in the pod specification YAML file by setting the spec.tolerations parameter.

Taints and tolerations consist of a key, value, and effect.

* The key is any string, up to 253 characters.
* The value is any string, up to 63 characters.
* The effect is one of the following options:

**NoSchedule**

If the pod does not match the taint, then the node prevents scheduling it. Existing pods on the node continue running on the node.

**PreferNoSchedule**

If the pod does not match the taint, then the scheduler tries to avoid scheduling it in the node. Existing pods on the node continue running on the node.

**NoExecute**

If the pod does not match the taint, then the node prevents scheduling it. The scheduler removes any existing pods on the node that do not have a matching toleration.

Tolerations also include the operator parameter. The following values for the operator parameter are possible:

**Equal**

The toleration matches the taint if the key, value, and effect parameters match. This behavior is the default.

**Exists**

The toleration matches the taint if the key and effect parameters match. You must leave a blank value parameter.

#### **Pod Disruption Budget**

A pod disruption budget is a policy resource to control the disruption of pods during voluntary disruptions, such as scaling down, updating applications, or draining a node for maintenance. Pod disruption budgets do not apply on node failures.

By setting up pod disruption budgets, you can manage the availability and stability of your applications during disruptions, to reduce the risk of service degradation or downtime when maintaining or updating your cluster.

The configuration for a pod disruption budget consists of the following parts:

* A label selector to target a specific set of pods. By using the label selector, you define the pods that the pod disruption budget protects. Only pods that match the label selector are subject to the budget constraints. The pod selector in a pod disruption budget typically selects only pods from the same deployment.
* The availability level, which specifies the minimum number of pods that must be available simultaneously. You can define the following availability levels:

**minAvailable**

Defines the minimum number of pods that must always be available, even during a disruption. A pod is available when it has the Ready condition with the True value.

**maxUnavailable**

Defines the maximum number of pods that can be unavailable during a disruption.

### References

For more information about pod scheduling on OpenShift, refer to the Controlling Pod Placement onto Nodes (Scheduling) section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling>

For more information about pod disruption budgets, refer to the Understanding How to Use Pod Disruption Budgets to Specify the Number of Pods That Must Be Up section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-pod-distruption-about_nodes-pods-configuring>

## **Node Selectors and Taints**

### **Objectives**

* Configure a workload to run on dedicated nodes, and prevent other workloads from running on those nodes.

### **Ensure and Prevent Workloads on Nodes**

OpenShift advanced scheduling features include node selectors, which ensure that a pod is placed in certain nodes, and taints and tolerations, which prevent certain nodes from running some workloads.

Use node selectors to schedule pods to a dedicated set of nodes. For example, schedule a pod to a node that provides special hardware that the pod needs.

Use taints and tolerations to avoid scheduling pods to a dedicated set of nodes. For example, block a pod to run on a node that is reserved for OpenShift cluster services or for control plane services.

### **Node Selectors**

By using node selectors, you can specify target node labels on your pod, so the OpenShift scheduler tries to place the pod only on those nodes that match the target labels. If the OpenShift scheduler does not find a node that matches the target labels, then your pod will be unavailable.

You can use pod node selectors to place pods on dedicated nodes. You define pod node selectors as attributes on the pod template.

Use project node selectors to place pods in a project on dedicated nodes. You define project node selectors as annotations in the project CR.

Use cluster-wide node selectors to place pods on dedicated nodes anywhere in the cluster. You define cluster-wide node selectors as attributes on the OpenShift scheduler CR.

Specific syntax details for node selectors are explained later in this section.

|  |
| --- |
|  |

Figure 4.2: Node selectors

The previous diagram shows how pod node selectors work. In the figure, the worker01 node has the key1=value1 label, but the worker02 node does not have that label. For the first pod, OpenShift can schedule it in either node, because it does not have a node selector.

The second pod has a node selector that is defined as an attribute in the pod template. Then, OpenShift can schedule the pod only in nodes with a label that matches the node selector. Thus, the OpenShift scheduler can place the pod only in the worker01 node.

#### **Node Labels**

To ensure that the OpenShift scheduler finds a node that matches the target labels, first label your nodes. You can add labels directly to a node resource or indirectly by using a machine set resource.

If your cluster is installed in a way that supports machine set resources, then you must use machine sets to set labels on nodes. In that case, do not label nodes directly. If your cluster does not support machine set resources, because for example it is installed by using the user-provisioned infrastructure method, then you must label nodes directly.

### Note

If your cluster supports machine set resources, then OpenShift can delete and re-create the nodes at any time. Thus, if you directly label the node, then the node loses the label outside the machine set when OpenShift re-creates it.

You can add one or more labels to a node by using the following command:

[user@host ~]$ **oc label node *node1* *key1=value1* ... *keyN=valueN***

node/node1 labeled

You can remove labels from a node by using the following command:

[user@host ~]$ **oc label node *node1* *key1-* ... *keyN-***

node/node1 unlabeled

#### **Pod Node Selectors**

Use node selectors on a pod to specify target node labels where the OpenShift scheduler tries to place the pod.

You can add one or more node selectors to a pod by using the following syntax:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

...output omitted...

spec:

**nodeSelector:**

**key1: value1**

**key2: value2**

...output omitted...

In the previous example, the OpenShift scheduler places the pod only in nodes with both the key1=value1 and key2=value2 labels.

You can read the node selectors for a pod by reviewing the Node-Selectors parameter as follows:

[user@host ~]$ **oc describe pod *myapp-95664666d-5l22c***

...output omitted...

Node-Selectors: key1:value1

key2:value2

...output omitted...

#### **Cluster-wide Node Selectors**

By default, the OpenShift scheduler tries to place the pods onto any available nodes in your cluster. However, you can modify the OpenShift scheduler default behavior so it includes default node selectors to any new pods that are created in the cluster. By using cluster-wide node selectors, you specify the default node selectors that OpenShift adds to the pods. You require administrative privileges to specify cluster-wide node selectors.

To define cluster-wide node selectors, you can edit the OpenShift scheduler CR as follows:

[user@host ~]$ **oc edit scheduler cluster**

Then specify the defaultNodeSelector parameter.

apiVersion: config.openshift.io/v1

kind: Scheduler

metadata:

name: cluster

...output omitted...

spec:

**defaultNodeSelector: key1=value1,key2=value2**

...output omitted...

After you edit the OpenShift scheduler CR, wait for the pods in the openshift-kube-apiserver project to redeploy.

In the previous example, any new pods that you create in the cluster have the key1=value1 and key2=value2 node selectors, by default.

#### **Project Node Selectors**

You can specify node selectors for a project, so OpenShift includes those node selectors to any new pods that are created in the project. You require administrative privileges to specify project node selectors.

To define project node selectors, you can create a YAML file for an OpenShift project CR, and define the node selectors in the metadata.annotations section, as follows:

apiVersion: v1

kind: Namespace

metadata:

name: myproject

annotations:

**openshift.io/node-selector: key3=value3,key4=value4**

...output omitted...

In the previous example, any new pods that you create in the project have by default the key3=value3 and key4=value4 node selectors.

#### **Levels for Node Selectors**

You can define node selectors at the following levels: pod, project, and cluster-wide.

Project node selectors take precedence over cluster-wide node selectors. Thus, if you define both project and cluster-wide node selectors, then OpenShift applies only the project node selectors to the new pods but not the cluster-wide node selectors.

Pod node selectors are additive to cluster-wide and project node selectors. Thus, if you define both pod and project node selectors, then OpenShift creates the pod with both the pod node selectors and the project node selectors. If you define both pod and cluster-wide node selectors, then OpenShift creates the pod with both the pod node selectors and the cluster-wide node selectors.

### Important

Creating a pod node selector with the same key string but with a different value string from a project node selector or from a cluster-wide node selector causes a conflict, and OpenShift fails to create your pod.

### **Taints and Tolerations**

You can use taints on nodes to prevent certain nodes from running some workloads. The OpenShift scheduler schedules pods in those nodes only if the pods have a matching toleration.

|  |
| --- |
|  |

Figure 4.3: Node taints and pod tolerations

In the previous figure, the worker01 node has a taint, but the worker02 node does not have a taint. The OpenShift scheduler cannot place the first pod in the worker01 node; the node rejects the pod because the toleration does not match the taint. Thus, OpenShift can schedule the pod only in the worker02 node.

OpenShift can schedule the second pod in both nodes, because the toleration matches the taint.

#### **Node Taints**

Taints consist of a key, a value, and an effect.

* The key is any string, up to 253 characters.
* The value is any string, up to 63 characters.
* The effect is one of the following options:

**NoSchedule**

If the pod does not match the taint, then the node prevents scheduling it. Existing pods on the node continue running on the node.

**PreferNoSchedule**

If the pod does not match the taint, then the scheduler tries to avoid scheduling it in the node. Existing pods on the node continue running on the node.

**NoExecute**

If the pod does not match the taint, then the node prevents scheduling it. The scheduler removes any existing pods on the node without a matching toleration.

You can apply a taint to a node in the node specification YAML file by setting the spec.taints parameter or by using the following command:

[user@host ~]$ **oc adm taint nodes node1 key1=value1:NoSchedule**

node/node1 tainted

After you apply a taint to a node, you can verify the taint by using the following command:

[user@host ~]$ **oc describe nodes node1**

Name: worker01

Roles: worker

...output omitted...

**Taints: key1=value1:NoSchedule**

...output omitted...

To remove a taint from a node, use the following command:

[user@host ~]$ **oc adm taint nodes node01 key1=value1:NoSchedule-**

node/node01 untainted

#### **Pod Tolerations**

Tolerations consist of a key, a value, and an effect, similar to the node taints. Tolerations also include the operator parameter. The following values for the operator parameter are possible:

**Equal**

The toleration matches the taint if the key, the value, and the effect parameters match. This behavior is the default.

**Exists**

The toleration matches the taint if the key and the effect parameters match. You must leave the value parameter blank.

When you define a toleration with the NoExecute effect, you can also define the tolerationSeconds parameter. The tolerationSeconds parameter defines the time that a pod stays within a node before the node evicts it.

You can apply a toleration to a pod in the pod specification YAML file by setting the spec.tolerations parameter as follows:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

...output omitted...

spec:

**tolerations:**

**- key: "key1"**

**operator: "Equal"**

**value: "value1"**

**effect: "NoExecute"**

**tolerationSeconds: 3600**

...output omitted...

In the previous example, OpenShift can schedule the pod in nodes with the key1=value1:NoExecute taint. If OpenShift schedules the pod in a node with the taint, then the pod runs in the node for 3600 seconds before the node evicts it and OpenShift reschedules the pod to another node.

#### **Pod Scheduling and Node Conditions**

By default, OpenShift taints and evicts nodes by condition. For example, OpenShift automatically taints nodes under some conditions, such as memory or disk pressure.

The following taints are built into OpenShift:

**node.kubernetes.io/not-ready**

The node is not ready. This taint corresponds to the Ready=False node condition.

**node.kubernetes.io/unreachable**

The node is unreachable from the node controller. This taint corresponds to the Ready=Unknown node condition.

**node.kubernetes.io/memory-pressure**

The node has memory pressure issues. This taint corresponds to the MemoryPressure=True node condition.

**node.kubernetes.io/disk-pressure**

The node has disk pressure issues. This taint corresponds to the DiskPressure=True node condition.

**node.kubernetes.io/network-unavailable**

The node network is unavailable.

**node.kubernetes.io/unschedulable**

The node is unschedulable.

**node.cloudprovider.kubernetes.io/uninitialized**

This taint sets the node as unusable when you start the node controller in an external cloud provider. After the cloud controller manager initializes the node, the kubelet removes the taint.

**node.kubernetes.io/pid-pressure**

The node has process identifier (PID) pressure. This taint corresponds to the PIDPressure=True node condition.

If a node reports one of the conditions from the previous table, then OpenShift applies the taint to the node until the condition clears.

For example, you can verify how OpenShift automatically applies the node.kubernetes.io/unschedulable taint to an unschedulable pod by following the next example. First, verify the taints for a node by using the following command:

[user@host ~]$ **oc describe nodes node1 | grep Taints**

Taints: <none>

Then, mark the node as unschedulable.

[user@host ~]$ **oc adm cordon node1**

node/node1 cordoned

Finally, verify that OpenShift automatically applies the node.kubernetes.io/unschedulable taint with the NoSchedule effect to the node.

[user@host ~]$ **oc describe nodes node1 | grep Taints**

Taints: node.kubernetes.io/unschedulable:NoSchedule

OpenShift applies the taints to the nodes from the previous table with the NoSchedule effect, except for the node.kubernetes.io/not-ready and node.kubernetes.io/unreachable taints where OpenShift uses the NoExecute effect.

For the taints with the NoSchedule effect, the OpenShift scheduler schedules the pod to the node if the pod has a matching toleration, or if the node returns to a normal state, but running pods remain unaltered.

For the taints with the NoExecute effect, the OpenShift scheduler schedules the pod to the node if the pod has a matching toleration, or if the node returns to a normal state, and starts evicting and rescheduling running pods to different nodes. For pods that do not tolerate the taint and that are running on the node, OpenShift evicts them immediately. For pods that match the toleration, OpenShift evicts them only if the tolerationSeconds parameter is defined on the pod. The tolerationSeconds parameter defines the time in seconds that a pod remains in a node until the node evicts it.

The tolerationSeconds parameter is useful in certain scenarios, such as when an application must create local files when running for the first time. In this case, waiting for a few seconds for the pod to recover might be preferable to directly re-creating the pod on another node. If you define the tolerationSeconds parameter for an application, then the node waits the specified time before evicting the pods.

#### **Project Tolerations**

You can specify tolerations for a project, so OpenShift includes those tolerations to any new pods that are created in the project. You require administrative privileges to specify project tolerations. If you define project and pod tolerations, then OpenShift creates the pod with both of these tolerations.

To define project tolerations, you can create a YAML file for an OpenShift project CR and define the toleration in the metadata.annotations section as follows:

kind: Project

apiVersion: project.openshift.io/v1

metadata:

name: myproject

annotations:

**scheduler.alpha.kubernetes.io/defaultTolerations: >-**

**[{"operator":"Equal","effect":"NoSchedule","key":"key1","value":"value1"}]**

In the previous example, any new pods that you create in the project have by default the key1=value1:NoSchedule toleration.

#### **Tolerating All Taints**

You can configure a pod to tolerate all the node taints by using the operator: "Exists" toleration with no key or value parameters.

apiVersion: v1

kind: Pod

metadata:

name: my-pod

...output omitted...

spec:

**tolerations:**

**- operator: "Exists"**

### Note

OpenShift does not remove the pods with the operator="Exists" toleration from a node that has taints.

### References

For more information about node selectors, refer to the Placing Pods on Specific Nodes Using Node Selectors section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-scheduler-node-selectors>

For more information about taints and tolerations, refer to the Controlling Pod Placement Using Node Taints section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-scheduler-taints-tolerations>

## **Guided Exercise: Node Selectors and Taints**

Configure a workload to run on dedicated nodes, and prevent other workloads from running on those nodes.

**Outcomes**

* Create deployments with node selectors to specify the node pool for a workload.
* Use taints and tolerations to prevent a node from running workloads.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start scheduling-selector**

**Instructions**

Your company has a cluster with three compute nodes in three different racks, as in the following table:

| **Node name** | **Labels** |
| --- | --- |
| worker01 | * rack=1 * cpu=standard |
| worker02 | * rack=2 * cpu=standard |
| worker03 | * rack=3 * cpu=fast |

The nodes include a label with the name of the rack that they are part of. The worker01 and worker02 nodes have a standard CPU and include the cpu=standard label. The worker03 node has a fast CPU and includes the cpu=fast label.

This guided exercise includes the following scenarios:

In the first scenario, you create a deployment with two replicas to verify the default OpenShift scheduler behavior, which distributes the deployment pods between the three available compute nodes. Then, your company requires you to create another deployment, with two replicas, which needs a fast CPU. Thus, you must use a node selector to ensure that the OpenShift scheduler places the deployment in the fast CPU compute node. Finally, you drain the worker03 node from the cluster to simulate a failure in the rack, and verify the status for the deployments.

In the second scenario, you create a project node selector for the standard CPU nodes. Then, you create a deployment with two replicas and verify that OpenShift applies the project node selector to the deployment.

In this scenario, you also create another deployment with two replicas and with a node selector for the compute node in the first rack. This deployment explores how project and pod node selectors interact with each other. Thus, you verify that pods that the deployment creates have both the project and pod node selectors.

In the third scenario, the student taints the worker01 and worker03 nodes to reserve some workload capacity, because your cluster has limited capacity. The taint is type=mission-critical with the NoSchedule option, which ensures that previous node workloads continue working but that new workloads need a toleration to be placed in those nodes. In this scenario, you first try to create a deployment with a node selector for the fast CPU label, to meet your application's requirements. Because your deployment does not include a toleration for the taint, the deployment is not available. Then, you create another deployment, to include the node selector and the toleration, to verify that the deployment is available.

Create all the deployments as the developer user.

1. As the admin user, connect to the OpenShift cluster and verify the labels for the nodes in the cluster.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. List the labels for the compute nodes. Verify the following node labels:
     + The rack=1 and cpu=standard labels for the worker01 node
     + The rack=2 and cpu=standard labels for the worker02 node
     + The rack=3 and cpu=fast labels for the worker03 node

[student@workstation ~]$ **oc get no -L rack,cpu**

NAME STATUS ROLES AGE VERSION **RACK** **CPU**

master01 Ready control-plane,master 140d v1.25.7+eab9cc9

master02 Ready control-plane,master 140d v1.25.7+eab9cc9

master03 Ready control-plane,master 140d v1.25.7+eab9cc9

**worker01** Ready worker 37d v1.25.7+eab9cc9 **1** **standard**

**worker02** Ready worker 37d v1.25.7+eab9cc9 **2** **standard**

**worker03** Ready worker 37d v1.25.7+eab9cc9 **3** **fast**

1. Create a deployment called myapp with two replicas. Verify the default OpenShift scheduler behavior, which distributes the deployment pods between the three available compute nodes.
   1. Connect to the OpenShift cluster as the developer user with developer as the password, and verify that OpenShift uses the scheduling-selector project.

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

...output omitted...

Using project "scheduling-selector".

* 1. Change to the ~/DO380/labs/scheduling-selector directory.

[student@workstation ~]$ **cd ~/DO380/labs/scheduling-selector**

* 1. Create a deployment CR YAML file with two replicas. You can find an incomplete example for the deployment CR in the ~/DO380/labs/scheduling-selector/deployment.yml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: **myapp**

spec:

replicas: **2**

selector:

matchLabels:

app: myapp

...output omitted...

* 1. Apply the configuration for the deployment CR.

[student@workstation scheduling-selector]$ **oc create -f deployment.yml**

deployment.apps/myapp created

* 1. Verify that the deployment is correctly created with two replicas. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

myapp 2/2 2 2 43s

* 1. Verify that the OpenShift scheduler distributes the two pods between the available compute nodes. The pod names and the nodes that run on your pods might differ in your system.

[student@workstation scheduling-selector]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

myapp-847cfd74d9-84tqf 1/1 Running ... **worker02** ...

myapp-847cfd74d9-npjrn 1/1 Running ... **worker01** ...

1. Your company requires you to create a deployment, with two replicas, which needs a fast CPU. Create a deployment called myapp-ns-fastcpu with two replicas. The deployment must have a node selector for the fastCPU label.
   1. Create a deployment CR YAML file with two replicas and a node selector for the fastCPU label. You can find an incomplete example for the deployment in the ~/DO380/labs/scheduling-selector/deployment-ns-fastcpu.yml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-ns-fastcpu

spec:

replicas: 2

selector:

matchLabels:

app: myapp-ns-fastcpu

template:

metadata:

labels:

app: myapp-ns-fastcpu

spec:

...output omitted...

**nodeSelector:**

**cpu: fast**

* 1. Apply the configuration for the deployment CR.

[student@workstation scheduling-selector]$ **oc create \**

**-f deployment-ns-fastcpu.yml**

deployment.apps/myapp-ns-fastcpu created

* 1. Verify that the deployment is correctly created with two replicas. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

myapp 2/2 2 2 2m57s

**myapp-ns-fastcpu** 2/2 2 2 27s

* 1. Verify that the OpenShift scheduler creates two pods on the worker03 node, because that node has the cpu=fast label. Pod names might differ in your system.

[student@workstation scheduling-selector]$ **oc get pods -o wide \**

**-l app=myapp-ns-fastcpu**

NAME READY STATUS ... NODE ...

myapp-ns-fastcpu-5577dd75d8-m6s9f 1/1 Running ... **worker03** ...

myapp-ns-fastcpu-5577dd75d8-pcw7j 1/1 Running ... **worker03** ...

1. Drain the worker03 node from the cluster to simulate a failure in the third rack. Verify the status for the deployments. The myapp deployment continues to be available, because the OpenShift scheduler can place its pods in the worker01 and worker02 nodes. However, the node selector in the myapp-ns-fastcpu deployment does not enable the scheduler to place the pods in the available compute nodes. Thus, the deployment is not available.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

...output omitted...

* 1. Drain all the pods from the worker03 node. This action might take a few minutes.

[student@workstation scheduling-selector]$ **oc adm drain worker03 \**

**--ignore-daemonsets --delete-emptydir-data**

...output omitted...

node/worker03 drained

* 1. Verify the status for the deployments. Although the myapp deployment is available, the myapp-ns-fastcpu deployment is not available due to the node selector.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

myapp 2/2 2 2 10m

myapp-ns-fastcpu **0/2** 2 **0** 6m4s

* 1. Verify that the scheduler cannot use the worker01 and worker02 nodes for the myapp-ns-fastcpu deployment due to the node selector.

[student@workstation scheduling-selector]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

myapp-847cfd74d9-84tqf 1/1 Running ... **worker02** ...

myapp-847cfd74d9-npjrn 1/1 Running ... **worker01** ...

myapp-ns-fastcpu-5577dd75d8-b422c 0/1 Pending ... **<none>** ...

myapp-ns-fastcpu-5577dd75d8-vdtsx 0/1 Pending ... **<none>** ...

* 1. Review the project events. The OpenShift scheduler does not find a node for the myapp-ns-fastcpu deployment due to the node selector.

[student@workstation scheduling-selector]$ **oc get events \**

**--sort-by='{.lastTimestamp}'**

...output omitted...

4m11s Normal SuccessfulCreate replicaset/myapp-ns-fastcpu-5577dd75d8 Created pod: myapp-ns-fastcpu-5577dd75d8-vdtsx

4m11s Normal SuccessfulCreate replicaset/myapp-ns-fastcpu-5577dd75d8 Created pod: myapp-ns-fastcpu-5577dd75d8-b422c

**4m11s Warning FailedScheduling pod/myapp-ns-fastcpu-5577dd75d8-b422c 0/6 nodes are available: 1 node(s) were unschedulable, 2 node(s) didn't match Pod's node affinity/selector, 3 node(s) had untolerated taint {node-role.kubernetes.io/master: }. preemption: 0/6 nodes are available: 6 Preemption is not helpful for scheduling.**

* 1. Review the pod information. Pod names might differ in your system.

[student@workstation scheduling-selector]$ **oc describe pod \**

***myapp-ns-fastcpu-5577dd75d8-b422c***

...output omitted...

Events:

Type Reason Age From Message

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

Warning FailedScheduling 2m10s (x2 over 7m12s) default-scheduler 0/6 nodes are available: **1 node(s) were unschedulable, 2 node(s) didn't match Pod's node affinity/selector, 3 node(s) had untolerated taint {node-role.kubernetes.io/master: }**. preemption: 0/6 nodes are available: 6 Preemption is not helpful for scheduling.

* 1. Mark the worker03 node as schedulable.

[student@workstation scheduling-selector]$ **oc adm uncordon worker03**

node/worker03 uncordoned

1. Create the scheduling-ns project with a project node selector for the standard CPU nodes.
   1. As the admin user, create the project CR YAML file with a node selector for the standard CPU nodes. You can find an incomplete example for the project in the ~/DO380/labs/scheduling-selector/project-scheduling.yml file.

apiVersion: v1

kind: Namespace

metadata:

name: **scheduling-ns**

annotations:

**openshift.io/node-selector: cpu=standard**

* 1. Apply the configuration for the project CR.

[student@workstation scheduling-selector]$ **oc create \**

**-f project-scheduling.yml**

namespace/scheduling-ns created

* 1. Give edit permission to the developer user in the scheduling-ns project.

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

**edit developer -n scheduling-ns**

clusterrole.rbac.authorization.k8s.io/edit added: "developer"

* 1. Connect to the OpenShift cluster as the developer user with developer as the password.

[student@workstation scheduling-selector]$ **oc login -u developer -p developer**

...output omitted...

* 1. Change to the scheduling-ns project.

[student@workstation scheduling-selector]$ **oc project scheduling-ns**

...output omitted...

* 1. Verify that the project includes the node selector.

[student@workstation scheduling-selector]$ **oc describe project scheduling-ns**

Name: scheduling-ns

Created: 4 minutes ago

...output omitted...

Annotations: **openshift.io/node-selector=cpu=standard**

openshift.io/sa.scc.mcs=s0:c26,c25

openshift.io/sa.scc.supplemental-groups=1000700000/10000

openshift.io/sa.scc.uid-range=1000700000/10000

Display Name: <none>

Description: <none>

Status: Active

**Node Selector: cpu=standard**

Quota: <none>

Resource limits: <none>

...output omitted...

* 1. Create a deployment CR YAML file with two replicas. You can find an incomplete example for the deployment in the ~/DO380/labs/scheduling-selector/deployment-project-ns.yml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: **project-ns**

spec:

replicas: **2**

selector:

matchLabels:

app: project-ns

...output omitted...

* 1. Apply the configuration for the deployment CR.

[student@workstation scheduling-selector]$ **oc create -f deployment-project-ns.yml**

deployment.apps/project-ns created

* 1. Verify the status for the deployment. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

project-ns 2/2 2 2 131s

* 1. Verify that the scheduler uses the worker01 and worker02 nodes due to the project node selector.

[student@workstation scheduling-selector]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

project-ns-79c4798c49-k8c92 1/1 Running ... **worker01** ...

project-ns-79c4798c49-r2dn7 1/1 Running ... **worker02** ...

* 1. Verify that the pods in the deployment include the project node selector. Pod names might differ in your system.

[student@workstation scheduling-selector]$ **oc describe pod \**

***project-ns-79c4798c49-k8c92***

Name: project-ns-79c4798c49-k8c92

Namespace: scheduling-ns

...output omitted...

**Node-Selectors: cpu=standard**

Tolerations: node.kubernetes.io/not-ready:NoExecute op=Exists for 300s

node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

...output omitted...

1. Create a deployment with two replicas and a node selector for the first rack. Verify that the deployment includes both the project and the pod node selectors.
   1. Create a deployment CR YAML file with two replicas and a node selector for the fast CPU node. You can find an incomplete example for the deployment in the ~/DO380/labs/scheduling-selector/deployment-project-podsel.yml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: project-podsel

spec:

replicas: 2

selector:

matchLabels:

app: project-podsel

template:

metadata:

labels:

app: project-podsel

spec:

...output omitted...

**nodeSelector:**

**rack: "1"**

* 1. Apply the configuration for the deployment CR.

[student@workstation scheduling-selector]$ **oc create \**

**-f deployment-project-podsel.yml**

deployment.apps/project-podsel created

* 1. Verify the status for the deployment. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

project-ns 2/2 2 2 12m

project-podsel 2/2 2 2 72s

* 1. Verify that the OpenShift scheduler places all the pods for the project-podsel deployment in the worker01 node, because the pods have both labels.

[student@workstation scheduling-selector]$ **oc get pods -o wide \**

**-l app=project-podsel**

NAME READY STATUS ... NODE ...

project-podsel-649f68f65d-44xxk 1/1 Running ... worker01 ...

project-podsel-649f68f65d-qvvjz 1/1 Running ... worker01 ...

* 1. Verify that the pods in the deployment include both the project and the pod node selectors. Pod names might differ in your system.

[student@workstation scheduling-selector]$ **oc describe pod \**

***project-podsel-649f68f65d-44xxk***

Name: project-podsel-649f68f65d-44xxk

Namespace: scheduling-ns

...output omitted...

**Node-Selectors: cpu=standard**

**rack=1**

Tolerations: node.kubernetes.io/not-ready:NoExecute op=Exists for 300s

node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

...output omitted...

1. As an OpenShift administrator, apply the type=mission-critical taint to the worker01 and worker03 nodes to reserve workload capacity. Use the NoSchedule option to ensure that previous node workloads continue working.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

...output omitted...

* 1. Apply the type=mission-critical:NoSchedule taint to the worker01 and worker03 nodes.

[student@workstation scheduling-selector]$ **oc adm taint nodes worker01 \**

**type=mission-critical:NoSchedule**

node/worker01 tainted

[student@workstation scheduling-selector]$ **oc adm taint nodes worker03 \**

**type=mission-critical:NoSchedule**

node/worker03 tainted

* 1. Verify the taints for the worker01 and worker03 nodes.

[student@workstation scheduling-selector]$ **oc describe node worker01 | grep Taints**

Taints: type=mission-critical:NoSchedule

[student@workstation scheduling-selector]$ **oc describe node worker03 | grep Taints**

Taints: type=mission-critical:NoSchedule

1. Create a deployment with two replicas with a node selector for the cpu=fast label. Verify that the deployment is not available because the two nodes with the cpu=fast label have a taint, and the deployment does not have a toleration for the taint.
   1. Connect to the OpenShift cluster as the developer user with developer as the password.

[student@workstation scheduling-selector]$ **oc login -u developer -p developer**

...output omitted...

* 1. Create the scheduling-taint project.

[student@workstation scheduling-selector]$ **oc new-project scheduling-taint**

...output omitted...

* 1. Create a deployment CR YAML file with two replicas and a node selector for the cpu=fast label. You can find an incomplete example for the deployment in the /​home/student/DO380/labs/scheduling-selector/deployment-taint-fastcpu.yml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-taint-fastcpu

spec:

replicas: 2

selector:

matchLabels:

app: myapp-taint-fastcpu

template:

metadata:

labels:

app: myapp-taint-fastcpu

spec:

...output omitted...

**nodeSelector:**

**cpu: fast**

* 1. Apply the configuration for the deployment CR.

[student@workstation scheduling-selector]$ **oc create \**

**-f deployment-taint-fastcpu.yml**

deployment.apps/myapp-taint-fastcpu created

* 1. Verify the status for the deployment. The myapp-taint-fastcpu deployment is not available due to node taint without a toleration.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

myapp-taint-fastcpu **0/2** 2 **0** 104s

* 1. Verify that the OpenShift scheduler cannot create the myapp-taint-fastcpu pods in a compute node.

[student@workstation scheduling-selector]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

myapp-taint-fastcpu-5c646fbd6b-k5r9g 0/1 Pending ... **<none>** ...

myapp-taint-fastcpu-5c646fbd6b-tz2nr 0/1 Pending ... **<none>** ...

* 1. Review the project events. The OpenShift scheduler does not find a node for the myapp-taint-fastcpu deployment due to the node selector and the node taint.

[student@workstation scheduling-selector]$ **oc get events \**

**--sort-by='{.lastTimestamp}'**

...output omitted...

2m11s Warning FailedScheduling pod/myapp-taint-fastcpu-5c646fbd6b-tz2nr **0/6 nodes are available: 1 node(s) didn't match Pod's node affinity/selector, 2 node(s) had untolerated taint {type: mission-critical}, 3 node(s) had untolerated taint {node-role.kubernetes.io/master: }**. preemption: 0/6 nodes are available: 6 Preemption is not helpful for scheduling.

2m11s Normal SuccessfulCreate replicaset/myapp-taint-fastcpu-5c646fbd6b Created pod: myapp-taint-fastcpu-5c646fbd6b-tz2nr

2m11s Normal SuccessfulCreate replicaset/myapp-taint-fastcpu-5c646fbd6b Created pod: myapp-taint-fastcpu-5c646fbd6b-k5r9g

2m11s Normal ScalingReplicaSet deployment/myapp-taint-fastcpu Scaled up replica set myapp-taint-fastcpu-5c646fbd6b to 2

* 1. Remove the myapp-taint-fastcpu deployment.

[student@workstation scheduling-selector]$ **oc delete deployment \**

**myapp-taint-fastcpu**

deployment.apps "myapp-taint-fastcpu" deleted

1. Create a deployment with the node selector for the cpu=fast label and the toleration for the taint. Verify that the deployment is available.
   1. Modify the YAML file for the myapp-taint-fastcpu deployment CR by adding a toleration for the type:mission-critical:NoSchedule taint.

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-taint-fastcpu

spec:

replicas: 2

selector:

matchLabels:

app: myapp-taint-fastcpu

template:

metadata:

labels:

app: myapp-taint-fastcpu

spec:

...output omitted...

nodeSelector:

cpu: standard

**tolerations:**

**- key: "type"**

**value: "mission-critical"**

**operator: "Equal"**

**effect: "NoSchedule"**

* 1. Apply the configuration for the deployment CR.

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

**deployment-taint-fastcpu.yml**

deployment.apps/myapp-taint-fastcpu created

* 1. Verify the status for the deployment. The myapp-taint-fastcpu deployment is available because it includes a toleration for the node taint.

[student@workstation scheduling-selector]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

myapp-taint-fastcpu **2/2** 2 **2** 62s

* 1. Verify that the OpenShift scheduler places the two pods for the myapp-taint-fastcpu deployment in the worker03 node, because the deployment has a toleration for the taint, and also because this node includes the specified label in the node selector.

[student@workstation scheduling-selector]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

myapp-taint-fastcpu-86cc9d87cc-7x4n8 1/1 Running ... **worker03** ...

myapp-taint-fastcpu-86cc9d87cc-8p4c4 1/1 Running ... **worker03** ...

* 1. Change to the student HOME directory.

[student@workstation scheduling-selector]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish scheduling-selector**

## **High Availability with Affinity Rules and Pod Disruption Budgets**

### **Objectives**

* Configure workloads for resilience against node failures.

### **Scheduling Affinity**

You can instruct the Kubernetes scheduler how to distribute the pods across the compute nodes in the cluster. The affinity rules enable the Kubernetes scheduler to set a preference to run the pods in a given cluster node, or to avoid running pods on a node where another set of pods is running.

Cluster administrators can set topology labels in the nodes to indicate the failure domain that they belong to. Application developers can configure the deployment resources to use the pod anti-affinity settings with custom topology labels to distribute the workloads across the failure domains.

Sometimes, the applications require multiple replicas to be available if a cluster disruption occurs. This type of requirement comes from load testing, which might indicate a need for a given number of replicas to give a suitable response time for users. The pod disruption budget resources set the minimum availability constraints to require that a given number or percentage of pods remain running even when a node is drained.

#### **Pod Affinity and Anti-affinity**

Pod affinity is a group of rules that are set in the pod specification. The scheduler uses the rules to place pods in the same cluster node where another workload is present, or to avoid placing pods in the same node where the pods from other workloads are running.

The pod affinity terms define a set of selectors to match the labels of other pods that are running in the cluster nodes. The scheduler can evaluate the pod affinity terms on a best-effort basis if the affinity setting is preferred, or require the conditions to be present to schedule the pod.

Kubernetes implements two types of pod affinity rules: pod affinity and pod anti-affinity.

**Pod affinity**

The pod affinity setting enables scheduling of pods in the same node where the pods from another workload are running. The pod specification contains the pod affinity setting, which uses a selector to match the labels from other pods by using an operator.

Pod affinity helps reduce the network latency between specific pods. Pod affinity can help improve the performance of workloads that require significant communication between pods.

In the following image, the pods have a pod affinity setting, to direct the scheduler to place them in the same node. The deployment is configured to run two replicas.

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The scheduler places the first pod in the first node. No preference exists, because no pods match the app=custom label selector. |
|  | The scheduler looks for pods that match the app=custom label selector and sets a preference to run the second pod in the same node as the first pod. The second pod is then scheduled in the first node. |

**Pod anti-affinity**

The pod anti-affinity setting prevents pods from being scheduled in the node where the pods from another workload are running. You can use a selector to match the labels from other pods. The pod specification contains the pod affinity setting that uses a selector to match the labels from other pods.

Pod anti-affinity prevents some workloads from running on the same node. Pod anti-affinity can help distribute a workload across failure domains to improve reliability, or prevent interference between workloads.

In the following image, the pods have a pod anti-affinity setting and direct the scheduler not to place them in the same node. The deployment is configured to run two replicas.

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | The scheduler places the first pod in the first node. No preference exists, because no pods match the app=custom label selector. |
|  | The scheduler looks for pods that match the app=custom label selector and sets a preference to avoid running the second pod in the same node as the first pod. The second pod is then scheduled on the second node. |

Similar to node affinity, you can apply the required or preferred affinity terms to the pod affinity or pod anti-affinity settings.

### Note

The scheduler evaluates the affinity terms only when the pod is created.

**Required**

The pod affinity terms are enforced when the pod is created, and the scheduler does not move the pod to another node if the conditions change.

The pod affinity rules include a list of selectors to match the labels of pods that are running in a cluster node.

**Preferred**

The scheduler evaluates the pod affinity terms on a best-effort basis. Thus, the scheduler might choose a node that does not satisfy all the requirements if no node matches all the conditions. The scheduler does not move the pod if the conditions change.

The preferred scheduling term has a list of weighted pod selector terms. Each term can match a label from running pods. Each rule has a weight value that is defined in the range 1-100 inclusive.

apiVersion: v1

kind: Pod

metadata:

name: my-pod

labels:

app: custom

spec:

affinity:

**podAntiAffinity**:

**preferredDuringSchedulingIgnoredDuringExecution**:

- **weight**: 100

**podAffinityTerm**:

**labelSelector**:

**matchExpressions**:

- key: **app**

operator: In

values:

- custom

topologyKey: **rack**

...content omitted...

|  |  |
| --- | --- |
|  | The expressions are evaluated on a best-effort basis and the pod is scheduled in a node even if the constraints are not met. |
|  | The weight value is between 1-100 inclusive. |
|  | The expression that matches a node label. |
|  | The operator that matches the labels. |
|  | The topology key that the scheduler uses to distribute pods across the failure domains. |

#### **Topology Keys**

The Kubernetes scheduler can use node labels to indicate the failure domain that each node belongs to. The failure domain can be a rack that is connected to a different power source or networking equipment, or an indicator that the node is in another building. Kubernetes uses the following node labels as default topology keys to distribute the workloads across the cluster nodes:

**kubernetes.io/hostname**

This label is set to match the hostname value of each node in the cluster.

**topology.kubernetes.io/zone**

Represents the availability zone of the node. This label is set by the cloud provider and might not be present in on-premise deployments.

**topology.kubernetes.io/region**

Represents the region of the node. This label is set by the cloud provider and might not be present in on-premise deployments.

The topology key helps the scheduler to spread the pods across the failure domains. The cluster administrator must label all the nodes to indicate their corresponding failure domain. Application developers can use the node label as a custom topology key in the pod affinity terms, to direct the scheduler how to place the pods in the cluster nodes.

The following scenario describes a cluster distributed between two racks. Each rack is connected to a different power source. The nodes are distributed by rack according to the following table:

| **Topology Key** | **Control plane nodes** | **Compute nodes** |
| --- | --- | --- |
| rack=rack-1 | master02 , master03 | node01 |
| rack=rack-2 | master01 | node02 , node03 |

The administrator added the rack label to indicate the location and failure domain of each node. This label is intended as a custom topology key, so that the scheduler can spread the pods evenly across the compute nodes in different racks.

**Without a custom topology key**

The value of the kubernetes.io/hostname topology key is different on each node, and the scheduler identifies each node as a separate failure domain. The scheduler uses the default topology keys, and distributes the pods evenly across all the compute nodes.

|  |
| --- |
|  |
|  |

**With a custom topology key**

The rack topology key specifies the failure domain that each node belongs to. The scheduler spreads pods evenly across the failure domains.

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | Three pods are placed in the node01 compute node in the rack-1 failure domain. |
|  | The other three pods are placed in the nodes in the rack-2 failure domain. |

### **Pod Disruption Budgets**

The pod disruption budget is a policy object where you can specify the percentage or number of pods that must remain running when a voluntary disruption occurs in the cluster. The pod disruption budget resource is part of the policy/v1 API group.

Kubernetes clusters have two types of disruptions: involuntary and voluntary.

**Involuntary disruptions**

Involuntary disruptions occur when a node fails and is disconnected from the cluster. The involuntary disruptions could be related to a hardware problem, a network issue within a zone, or a power issue in a rack. The Kubernetes cluster detects that a node is not responsive, creates replacement pods in another node, and eventually deletes all the pods that are scheduled on the failed node.

**Voluntary disruptions**

Voluntary disruptions occur when nodes are cordoned or taken offline for maintenance. When a voluntary disruption occurs, all the pods on the affected node are evicted at the same time, and the scheduler creates replacement pods in another node. If all the pods of a workload are running in the same node and that node is drained, then the minimum availability constraints for the application are not fulfilled.

You can define the pod disruption budget resource with either the minAvailable or the maxUnavailable attribute. The values for these settings are often discovered after load testing on the application. Sometimes, a given number of pods must run all the time to prevent the application from queuing the user requests. Another example is that the application delivers an acceptable response time for the users even if only 60% of the pods are running.

**minAvailable**

The minimum number of pods to be available, even during a voluntary disruption. You can set this attribute to a percentage or integer value to specify the minimum number of replica pods to be available.

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: nginx-minavailable

spec:

**minAvailable**: 80%

**selector**:

**matchLabels**:

app: nginx

|  |  |
| --- | --- |
|  | The minimum percentage or number of pods to be available. |
|  | The label selector for the affected pods. |

**maxUnavailable**

The maximum number of pods that can be unavailable during a voluntary disruption. You can set this attribute to a percentage or integer value to specify the maximum number of replica pods that can be unavailable.

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: nginx-maxunavailable

spec:

**maxUnavailable**: 33%

**selector**:

**matchLabels**:

app: nginx

|  |  |
| --- | --- |
|  | The maximum percentage or number of pods that can be unavailable. |
|  | The label selector for the affected pods. |

You can list or describe the pod disruption budget resources to inspect their properties.

[user@host ~]$ **oc get pdb nginx-minavailable**

NAME MIN AVAILABLE MAX UNAVAILABLE ALLOWED DISRUPTIONS AGE

nginx-minavailable **80%** N/A 2 45s

|  |  |
| --- | --- |
|  | The minimum percentage or number of pods to be available. |

You can also describe the resource to view the label selector for the affected pods.

[user@host ~]$ **oc describe pdb nginx-maxunavailable**

Name: nginx-maxunavailable

Namespace: scheduling-pdb

Max unavailable: **33%**

Selector: **app=nginx**

Status:

Allowed disruptions: 4

Current: 10

Desired: 6

Total: 10

Events: <none>

|  |  |
| --- | --- |
|  | The maximum percentage or number of pods that can be unavailable. |
|  | The label selector for the affected pods. |

#### **Node Drain Without Pod Disruption Budget**

The deployment is configured to run six replica pods that are distributed between the compute nodes. The application has an appropriate response time if five or more pods are running.

|  |
| --- |
|  |

The first node is drained by using the oc adm drain command. The node is cordoned and marked as unschedulable.

|  |
| --- |
|  |

[user@host ~]$ **oc adm drain node/node01 ...**

node/node01 cordoned

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

… request.go:682] Waited for … due to client-side throttling ...output omitted...

...output omitted...

### Note

You can safely ignore the warnings about managed pods and client-side throttling.

All the pods that are running in the drained node are marked for eviction. The scheduler creates replacement pods in another node. Only four pods are ready and the application starts responding slowly.

|  |
| --- |
|  |

...output omitted...

evicting pod scheduling-pdb/nginx-56bdf7d8c6-kf98k

evicting pod scheduling-pdb/nginx-56bdf7d8c6-8kk2b

pod/nginx-56bdf7d8c6-kf98k evicted

pod/nginx-56bdf7d8c6-8kk2b evicted

...output omitted...

The compute node is finally marked as drained. The replacement pods are marked as ready. The application now runs all six pods and the response time is within user expectations.

|  |
| --- |
|  |

...output omitted...

node/node01 drained

The full output of the oc adm drain command is as follows:

[user@host ~]$ **oc adm drain node/node01 --ignore-daemonsets --delete-emptydir-data**

node/node01 cordoned

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

… request.go:682] Waited for … due to client-side throttling ...output omitted...

...output omitted...

evicting pod scheduling-pdb/nginx-56bdf7d8c6-kf98k

evicting pod scheduling-pdb/nginx-56bdf7d8c6-8kk2b

...output omitted...

pod/nginx-56bdf7d8c6-kf98k evicted

pod/nginx-56bdf7d8c6-8kk2b evicted

...output omitted...

node/node01 drained

|  |  |
| --- | --- |
|  | All the pods are marked for eviction when the node is drained. |
|  | All the pods are evicted from the node at the same time and the application availability constraint is not met. |

#### **Node Drain with Pod Disruption Budget**

With a pod disruption budget resource, you can specify the minimum availability constraints for the application. The node drain is blocked until the replacement pods for the application are scheduled in another node and become ready. The pod eviction is blocked until the pod disruption budget availability constraints are met. The pod eviction operation retries after five seconds.

The deployment is configured to run six replica pods that are distributed between the compute nodes. The application has an appropriate response time if five or more pods are running. The developer creates a pod disruption budget resource where the spec.minAvailable parameter is set to five, and specifies the selector to match the labels of the deployment pods.

|  |
| --- |
|  |

The first node is drained by using the oc adm drain command. The node is cordoned and marked as unschedulable.

|  |
| --- |
|  |

[user@host ~]$ **oc adm drain node/node01 ...**

**node/node01 cordoned**

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

… request.go:682] Waited for … due to client-side throttling ...output omitted...

...output omitted...

### Note

You can safely ignore the warnings about managed pods and client-side throttling.

All the pods that are running in the drained node are marked for eviction. The scheduler creates a replacement pod in another node and waits for the new pod to be ready. The second pod cannot be evicted, because the availability constraint states that five out of six pods must be running. The eviction operation for the second pod retries after five seconds.

|  |
| --- |
|  |

...output omitted...

evicting pod scheduling-pdb/nginx-647b4c98f5-nqvs6

evicting pod scheduling-pdb/nginx-647b4c98f5-98ptm

error when evicting pods/"nginx-647b4c98f5-98ptm" -n "scheduling-pdb" (will retry after 5s): **Cannot evict pod as it would violate the pod's disruption budget**.

...output omitted...

### Important

The drain process is blocked until all the pods are evicted from the node.

The first replacement pod is marked as ready, and the first pod is evicted from the drained node. The scheduler creates a second replacement pod in another node and waits for it to be ready.

|  |
| --- |
|  |

...output omitted...

pod/nginx-647b4c98f5-nqvs6 evicted

evicting pod scheduling-pdb/nginx-647b4c98f5-98ptm

...output omitted...

The second replacement pod is marked as ready, and the second pod is evicted from the drained node.

|  |
| --- |
|  |

...output omitted...

pod/nginx-647b4c98f5-98ptm evicted

...output omitted...

The compute node is finally marked as drained.

|  |
| --- |
|  |

...output omitted...

node/node01 drained

The full output of the oc adm drain command is as follows:

[user@host ~]$ **oc adm drain node/node01 --ignore-daemonsets --delete-emptydir-data**

node/node01 cordoned

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

… request.go:682] Waited for … due to client-side throttling ...output omitted...

...output omitted...

evicting pod scheduling-pdb/nginx-647b4c98f5-nqvs6

evicting pod scheduling-pdb/nginx-647b4c98f5-98ptm

error when evicting pods/"nginx-647b4c98f5-98ptm" -n "scheduling-pdb" (will retry after 5s): **Cannot evict pod as it would violate the pod's disruption budget**.

...output omitted...

error when evicting pods/"nginx-647b4c98f5-98ptm" -n "scheduling-pdb" (will retry after 5s): Cannot evict pod as it would violate the pod's disruption budget.

pod/nginx-647b4c98f5-nqvs6 evicted

evicting pod scheduling-pdb/nginx-647b4c98f5-98ptm

...output omitted...

pod/nginx-647b4c98f5-98ptm evicted

...output omitted...

node/node01 drained

|  |  |
| --- | --- |
|  | The pod is marked for eviction. |
|  | The pod eviction is blocked until the availability constraints from the pod disruption budget are met. |
|  | The pod is finally evicted from the drained node. |

### Note

You can safely ignore the warnings about managed pods and client-side throttling.

### References

For more information about pod scheduling on OpenShift, refer to the Controlling Pod Placement onto Nodes (Scheduling) section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling>

For more information about pod disruption budgets, refer to the Understanding How to Use Pod Disruption Budgets to Specify the Number of Pods That Must Be Up section in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-pod-distruption-about_nodes-pods-configuring>

For more information about pod disruption budgets, refer to the Pod Disruption Budgets section in the Red Hat OpenShift Container Platform 4.14 Post-installation Configuration documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/postinstallation_configuration/index#post-install-pod-disruption-budgets>

[Kubernetes - Assigning Pods to Nodes](https://v1-27.docs.kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity)

[Kubernetes - Disruptions](https://v1-27.docs.kubernetes.io/docs/concepts/workloads/pods/disruptions/#pod-disruption-budgets)

[Kubernetes - Specifying a Disruption Budget for Your Application](https://v1-27.docs.kubernetes.io/docs/tasks/run-application/configure-pdb/)

[Kubernetes Well-known Labels, Annotations and Taints](https://v1-27.docs.kubernetes.io/docs/reference/labels-annotations-taints/)

## **Guided Exercise: High Availability with Affinity Rules and Pod Disruption Budgets**

Configure a workload to spread its pods between nodes from different failure domains and set minimum availability requirements. Then, drain a node to simulate a cluster update, and prove that the application keeps minimum capacity and availability.

**Outcomes**

* Add pod anti-affinity settings to a deployment resource manifest to spread the pods across the failure domains.
* Create a pod disruption budget to set a minimum availability constraint that is fulfilled when the cluster has a voluntary disruption.
* Drain a compute node to simulate a voluntary disruption.

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

[student@workstation ~]$ **lab start scheduling-pdb**

**Instructions**

The company has a local OpenShift cluster that is distributed between two racks. Each rack is connected to a different power source. The nodes are distributed by rack according to the following table:

| **Rack** | **Control plane nodes** | **Compute nodes** |
| --- | --- | --- |
| rack-a | master01 , master02 | worker03 |
| rack-b | master03 | worker01 , worker02 |

The administrator needs to drain the compute nodes for maintenance. The administrator added the rack label to indicate the location and failure domain of each node. This label is intended as a custom topology key, so that the scheduler can spread the pods evenly across the compute nodes in different racks.

The application runs six pods and requires five of them to be available if a voluntary cluster disruption occurs, to achieve the intended response time.

The developer modifies the deployment resource to add the pod anti-affinity settings that use the custom topology, and also creates a pod disruption budget to indicate the minimum availability constraint of the application.

1. Verify that the nodes are labeled according to their rack location.
   1. Log in to the cluster as the admin user.

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

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

Login successful.

...output omitted...

* 1. Verify that all the nodes have the rack labels according to the previous table.

[student@workstation ~]$ **oc get nodes -L rack**

NAME STATUS ROLES AGE VERSION **RACK**

master01 Ready control-plane,master 8d v1.27.6+… **rack-a**

master02 Ready control-plane,master 8d v1.27.6+… **rack-a**

master03 Ready control-plane,master 8d v1.27.6+… **rack-b**

worker01 Ready worker 7d v1.27.6+… **rack-b**

worker02 Ready worker 7d v1.27.6+… **rack-b**

worker03 Ready worker 7d v1.27.6+… **rack-a**

|  |  |
| --- | --- |
|  | The worker01 and worker02 compute nodes are placed in the rack-b rack. |
|  | The worker03 compute node is placed in the rack-a rack. |

1. Create the deployment without pod affinity or a pod disruption budget.
   1. Log in as the developer user and verify that you are using the scheduling-pdb project.

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

Login successful.

You have one project on this server: "scheduling-pdb"

**Using project "scheduling-pdb"**.

* 1. Change to the ~/DO380/labs/scheduling-pdb directory.

[student@workstation ~]$ **cd ~/DO380/labs/scheduling-pdb**

* 1. Create the deployment by using the YAML resource manifest.

[student@workstation scheduling-pdb]$ **oc apply -f deployment.yaml**

deployment.apps/nginx created

* 1. Open a new terminal window, and then execute the following command to see the status of the pod disruption budget, deployment, and pods.

Wait until all pods are running, and verify that all the pods from the nginx deployment are marked as ready and available.

This process might take a few minutes.

[student@workstation scheduling-pdb]$ **watch oc get pdb,deployments,pods -o wide**

Every 2.0s: oc get pdb,deployments,pods ... workstation: Wed Jan 3 15:59:55 2024

NAME **READY** UP-TO-DATE **AVAILABLE** AGE ...

deployment.apps/nginx **6/6** 6 **6** 60s ...

NAME **READY** **STATUS** RESTARTS AGE IP **NODE** ...

pod/nginx-5676948d76-75l7z **1/1** **Running** 0 60s ... worker01 ...

pod/nginx-5676948d76-pkdr7 **1/1** **Running** 0 60s ... worker01 ...

pod/nginx-5676948d76-gcst5 **1/1** **Running** 0 60s ... worker01 ...

pod/nginx-5676948d76-njzv2 **1/1** **Running** 0 60s ... worker02 ...

pod/nginx-5676948d76-94zmk **1/1** **Running** 0 60s ... worker03 ...

pod/nginx-5676948d76-mcdbj **1/1** **Running** 0 60s ... worker03 ...

### Note

Keep this terminal window open to view the status of the resources from this exercise.

* 1. Return to the first terminal window and count the pods that are running on each compute node by using the count-pods.sh shell script.

The replica pods are distributed across the cluster nodes but are not distributed evenly between the rack-a and rack-b failure domains.

[student@workstation scheduling-pdb]$ **./count-pods.sh**

NODE PODS

worker01 3

worker02 1

worker03 2

|  |  |
| --- | --- |
|  | The worker01 and worker02 compute nodes are placed in the rack-b rack. |
|  | The worker03 compute node is placed in the rack-a rack. |

### Note

Although the exact number of pods that are running on each node might be different, the total replica count is six pods.

1. Simulate a voluntary disruption where the cluster administrator takes the worker01 node offline for maintenance.

### Important

The selected node for draining must have at least two pods running.

* 1. Log in as the admin user.

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

Login successful.

...output omitted...

* 1. Drain the worker01 node to simulate taking it offline for maintenance.

This command might take a few minutes to complete. Leave it running and continue with the next step. You review the output of this command in a later step.

[student@workstation scheduling-pdb]$ **oc adm drain node/worker01 \**

**--ignore-daemonsets --delete-emptydir-data**

...output omitted...

* 1. Switch to the second terminal window to view the eviction of the pods from the drained node. Wait until all pods are running in another node and are marked as ready. This process might take a few minutes.

All the application pods in the drained node are evicted at the same time and the minimum availability constraints are not met. Use the values in the age column to determine which pods were evicted from the drained node and were scheduled in a different node.

This situation happens because no pod disruption budget is associated with the deployment pods, and the deployment resource also does not have an affinity setting that uses the rack label as a custom topology key.

Every 2.0s: oc get pdb,deployments,pods ... workstation: Wed Jan 3 16:02:33 2024

NAME **READY** UP-TO-DATE **AVAILABLE** AGE ...

deployment.apps/nginx **3/6** 6 **3** 14m ...

NAME READY STATUS ... AGE IP NODE ...

pod/nginx-5676948d76-njzv2 1/1 Running ... 3m ... worker02 ...

pod/nginx-5676948d76-94zmk 1/1 Running ... 3m ... worker03 ...

pod/nginx-5676948d76-mcdbj 1/1 Running ... 3m ... worker03 ...

pod/nginx-5676948d76-hfbv9 **0/1** **Init:0/1** ... **1s** ... worker02 ...

pod/nginx-5676948d76-zxxlg **0/1** **Init:0/1** ... **1s** ... worker02 ...

pod/nginx-5676948d76-dh6dh **0/1** **Init:0/1** ... **1s** ... worker03 ...

|  |  |
| --- | --- |
|  | Only three replica pods are available. |
|  | Three replica pods are evicted from the drained compute node. |

### Note

Although the exact number of pods that are running on each node might be different, the total replica count is six pods.

* 1. Return to the first terminal window and inspect the output of the oc adm drain command.

Observe the pod eviction messages of the nginx pods. All the application pods in the drained node are evicted at the same time and the minimum availability constraint is not met.

[student@workstation scheduling-pdb]$ **oc adm drain node/worker01 \**

**--ignore-daemonsets --delete-emptydir-data**

node/worker01 cordoned

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

...output omitted...

I1221 21:29:52.102938 111157 request.go:696] ...output omitted...

...output omitted...

**evicting pod** scheduling-pdb/nginx-5676948d76-pkdr7

**evicting pod** scheduling-pdb/nginx-5676948d76-75l7z

**evicting pod** scheduling-pdb/nginx-5676948d76-gcst5

...output omitted...

pod/nginx-5676948d76-gcst5 **evicted**

pod/nginx-5676948d76-75l7z **evicted**

pod/nginx-5676948d76-pkdr7 **evicted**

...output omitted...

node/worker01 drained

|  |  |
| --- | --- |
|  | All the pods are marked for eviction when the node is drained. |
|  | All the pods are evicted from the node at the same time and the application availability constraint is not met. |

### Note

You can safely ignore the warnings about managed pods and client-side throttling.

* 1. Get the state of the nodes to verify that the drained node is marked as not schedulable.

[student@workstation scheduling-pdb]$ **oc get nodes**

NAME STATUS ROLES ...

master01 Ready control-plane,master ...

master02 Ready control-plane,master ...

master03 Ready control-plane,master ...

**worker01** Ready,**SchedulingDisabled** worker ...

worker02 Ready worker ...

worker03 Ready worker ...

|  |  |
| --- | --- |
|  | The compute node is drained for maintenance. |

* 1. Count the pods that are running on each compute node. The scheduler placed replacement pods for the evicted pods in the worker02 and worker03 compute nodes.

[student@workstation scheduling-pdb]$ **./count-pods.sh**

NODE PODS

**worker01** 0

worker02 3

worker03 3

|  |  |
| --- | --- |
|  | No pods are on this node, because it was just drained. |

* 1. Delete the nginx deployment.

[student@workstation scheduling-pdb]$ **oc delete deployment/nginx**

deployment.apps "nginx" deleted

* 1. Uncordon the worker01 node that you drained previously to remove the SchedulingDisabled status.

[student@workstation scheduling-pdb]$ **oc adm uncordon node/worker01**

node/worker01 uncordoned

* 1. List the cluster nodes and verify that all the compute nodes are marked as ready.

[student@workstation ~]$ **oc get nodes -L rack**

NAME STATUS ROLES AGE VERSION RACK

master01 Ready control-plane,master 8d v1.27.6+… rack-a

master02 Ready control-plane,master 8d v1.27.6+… rack-a

master03 Ready control-plane,master 8d v1.27.6+… rack-b

worker01 **Ready** worker 7d v1.27.6+… rack-b

worker02 Ready worker 7d v1.27.6+… rack-b

worker03 Ready worker 7d v1.27.6+… rack-a

1. Create the nginx deployment with pod anti-affinity to spread the pods evenly across the compute nodes.
   1. Log in as the developer user.

[student@workstation scheduling-pdb]$ **oc login -u developer -p developer**

Login successful.

...output omitted...

* 1. Edit the deployment-affinity.yaml file and set the affinity properties according to the following specification. Then, save and close the file.

...output omitted...

spec:

...output omitted...

template:

...output omitted...

spec:

...output omitted...

containers:

...output omitted...

affinity:

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

**topologyKey**: **rack**

labelSelector:

matchExpressions:

- key: app

operator: In

values:

- **nginx**

|  |  |
| --- | --- |
|  | The weighted pod affinity term is evaluated only during pod scheduling, on a best-effort basis. |
|  | The node label that indicates the failure domain for the nodes. |
|  | The label to select the pods that this affinity setting affects. |

### Note

The ~/DO380/solutions/scheduling-pdb/deployment-affinity.yaml file contains the correct configuration, and you can use it for comparison.

* 1. Create the application deployment resource by using the YAML manifest.

[student@workstation scheduling-pdb]$ **oc apply -f deployment-affinity.yaml**

deployment.apps/nginx created

* 1. Switch to the second terminal window. Wait until all pods are running and verify that all the pods from the nginx deployment are marked as ready and available.

This process might take a few minutes.

Every 2.0s: oc get pdb,deployments,pods ... workstation: Wed Jan 3 16:24:11 2024

NAME **READY** UP-TO-DATE **AVAILABLE** AGE ...

deployment.apps/nginx **6/6** 6 **6** 120s ...

NAME **READY** **STATUS** RESTARTS AGE IP **NODE** ...

pod/nginx-d5b9c7498-5hbkw **1/1** **Running** 0 99s ... worker01 ...

pod/nginx-d5b9c7498-nkx9j **1/1** **Running** 0 99s ... worker01 ...

pod/nginx-d5b9c7498-g6ztb **1/1** **Running** 0 99s ... worker02 ...

pod/nginx-d5b9c7498-bk7g6 **1/1** **Running** 0 99s ... worker03 ...

pod/nginx-d5b9c7498-djn8p **1/1** **Running** 0 99s ... worker03 ...

pod/nginx-d5b9c7498-pz8g5 **1/1** **Running** 0 99s ... worker03 ...

* 1. Return to the first terminal window and count the pods that are running on each compute node.

The pods are evenly distributed across the racks, because of the pod anti-affinity settings.

* + - Three pods are running in the rack-b rack nodes.
    - Three pods are running in the rack-a rack nodes.

[student@workstation scheduling-pdb]$ **./count-pods.sh**

NODE PODS

worker01 **2**

worker02 **1**

worker03 **3**

|  |  |
| --- | --- |
|  | The worker01 and worker02 compute nodes are in the rack-b rack. |
|  | The worker03 compute node is in the rack-a rack. |

1. Create the pod disruption budget with the intended constraints.
   1. Edit the pod-disruption-budget.yaml file and set the minimum available percentage and the label selector according to the following specification. Then, save and close the file.

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: nginx

labels:

app: nginx

spec:

**minAvailable**: **80%**

selector:

matchLabels:

**app**: **nginx**

### Note

The ~/DO380/solutions/scheduling-pdb/pod-disruption-budget.yaml file contains the correct configuration, and you can use it for comparison.

* 1. Create the pod disruption budget by using the YAML manifest.

[student@workstation scheduling-pdb]$ **oc apply -f pod-disruption-budget.yaml**

poddisruptionbudget.policy/nginx created

* 1. Verify that the nginx pod disruption budget was created, and that it has the intended minimum available attribute.

[student@workstation scheduling-pdb]$ **oc describe pdb nginx**

Name: nginx

Namespace: scheduling-pdb

**Min available: 80%**

**Selector: app=nginx**

Status:

**Allowed disruptions: 1**

Current: 6

Desired: 5

Total: 6

Events: <none>

|  |  |
| --- | --- |
|  | Only one pod can be evicted at a time from a drained node. |

1. Drain a compute node to simulate a voluntary disruption.

### Important

The selected node for draining must have at least two pods running.

* 1. Log in again as the admin user.

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

Login successful.

...output omitted...

* 1. Drain the worker03 node to simulate taking it offline for maintenance.

This command might take a few minutes to complete. Leave it running and continue with the next step. You review the output of this command in a later step.

[student@workstation scheduling-pdb]$ **oc adm drain node/worker03 \**

**--ignore-daemonsets --delete-emptydir-data**

...output omitted...

* 1. Switch to the second terminal window to view the eviction of the pods from the drained node. Wait until all pods are running in another node and are marked as ready. This process might take a few minutes.

One pod is evicted at a time from the drained node and the availability constraints are met. Use the values in the age column to determine which pods were evicted from the drained node and were scheduled in a different node.

Every 2.0s: oc get pdb,deployments,pods ... workstation: Wed Jan 3 16:28:12 2024

NAME MIN AVAIL… MAX UNAVAIL… ALLOWED DISRUPTIONS

poddisruptionbudget.policy/nginx **80%** N/A **1**

NAME **READY** UP-TO-DATE **AVAILABLE** AGE ...

deployment.apps/nginx **5/6** 6 **5** 30m ...

NAME READY STATUS RESTARTS AGE IP NODE ...

pod/nginx-d5b9c7498-5hbkw 1/1 Running 0 5m ... worker01 ...

pod/nginx-d5b9c7498-nkx9j 1/1 Running 0 5m ... worker01 ...

pod/nginx-d5b9c7498-g6ztb 1/1 Running 0 5m ... worker02 ...

pod/nginx-d5b9c7498-pz8g5 1/1 Running 0 5m ... **worker03** ...

pod/nginx-d5b9c7498-q6rcf **1/1** **Running** 0 **50s** ... worker01 ...

pod/nginx-d5b9c7498-pxx86 **0/1** **Init:0/1** 0 **10s** ... worker02 ...

**^C**

|  |  |
| --- | --- |
|  | The pod eviction follows the pod disruption budget. |
|  | The pods in the drained node continue to run until the scheduler evicts them. |
|  | The replacement pods are scheduled in another compute node. |
|  | Only one pod is evicted at a time from the drained node. |

Press **Ctrl**+**C** and close the second terminal window when done.

* 1. Return to the first terminal window and inspect the output of the oc adm drain command.

From the pod eviction messages of the nginx pods, observe that one pod is evicted at a time from the drained node and the availability constraints are met.

The pod eviction is blocked until the PDB availability constraints are met. The pod eviction operation is retried after five seconds.

[student@workstation scheduling-pdb]$ **oc adm drain node/worker03 \**

**--ignore-daemonsets --delete-emptydir-data**

node/worker03 cordoned

Warning: ignoring DaemonSet-managed Pods: ...output omitted...

...output omitted...

I0103 16:27:16.659505 29741 request.go:696] Waited for … due to client-side throttling, not priority and fairness, request: ...output omitted...

...output omitted...

**evicting pod** scheduling-pdb/nginx-d5b9c7498-bk7g6

**evicting pod** scheduling-pdb/nginx-d5b9c7498-djn8p

**evicting pod** scheduling-pdb/nginx-d5b9c7498-pz8g5

...output omitted...

pod/nginx-d5b9c7498-bk7g6 **evicted**

...output omitted...

error when evicting pods/"nginx-d5b9c7498-djn8p" -n "scheduling-pdb" (will retry after 5s): **Cannot evict pod as it would violate the pod's disruption budget**.

error when evicting pods/"nginx-d5b9c7498-pz8g5" -n "scheduling-pdb" (will retry after 5s): **Cannot evict pod as it would violate the pod's disruption budget**.

...output omitted...

pod/nginx-d5b9c7498-djn8p **evicted**

evicting pod scheduling-pdb/nginx-d5b9c7498-pz8g5

error when evicting pods/"nginx-d5b9c7498-pz8g5" -n "scheduling-pdb" (will retry after 5s): **Cannot evict pod as it would violate the pod's disruption budget**.

...output omitted...

pod/nginx-d5b9c7498-pz8g5 **evicted**

node/worker03 drained

|  |  |
| --- | --- |
|  | The pod is marked for eviction. |
|  | The pod eviction is blocked until the PDB availability constraints are met. |
|  | The pod is finally evicted from the drained node. |

### Note

You can safely ignore the warnings about managed pods and client-side throttling.

* 1. List the cluster nodes and verify that the worker01 node status is SchedulingDisabled.

[student@workstation scheduling-pdb]$ **oc get nodes**

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane,master 28d v1.27.6+...

master02 Ready control-plane,master 28d v1.27.6+...

master03 Ready control-plane,master 28d v1.27.6+...

worker01 Ready worker 8d v1.27.6+...

worker02 Ready worker 8d v1.27.6+...

**worker03** Ready,**SchedulingDisabled** worker 8d v1.27.6+...

|  |  |
| --- | --- |
|  | The compute node is marked as not schedulable. |

* 1. Count the pods that are running on each compute node.

[student@workstation scheduling-pdb]$ **./count-pods.sh**

NODE PODS

worker01 **3**

worker02 **3**

worker03 0

|  |  |
| --- | --- |
|  | The pods are evenly distributed on the remaining nodes. |
|  | No pods are on this node, because it was just drained. |

* 1. Switch to the student HOME directory.

[student@workstation scheduling-pdb]$ **cd**

[student@workstation ~]$

1. **Optional**: Clean up the resources that were used in this exercise.
   1. Delete the scheduling-pdb project.

[student@workstation ~]$ **oc delete project scheduling-pdb**

project.project.openshift.io "scheduling-pdb" deleted

* 1. Uncordon all the compute nodes.
  2. [student@workstation ~]$ **oc adm uncordon -l node-role.kubernetes.io/worker**

...output omitted...

* 1. Remove the rack label from all nodes.
  2. [student@workstation ~]$ **oc label node --all rack-**

...output omitted...

**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 scheduling-pdb**

## **Lab: Pod Scheduling**

Configure a workload to ensure that its pods run on dedicated nodes, and configure a node to prevent a pod's workloads from running on them.

Configure applications for resilience against node failures.

**Outcomes**

* Create deployments with node selectors to specify node pools for a workload.
* Configure pods with tolerations for nodes that use taints.
* Use pod disruption budgets to minimize downtime during cluster updates and other node maintenance processes.
* Use pod affinity rules to ensure that pods from the different workloads run on the same nodes.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Your company has a cluster with three compute nodes, as in the following diagram:

| **Node name** | **Labels** | **Taints** |
| --- | --- | --- |
| worker01 | disk=ssd | application=ml |
| worker02 | disk=ssd | (none) |
| worker03 | disk=nvme | (none) |

Servers might have different hardware. The worker01 and worker02 nodes have SATA SSD disks and include the disk=ssd label. The worker03 node has NVMe disks and includes the disk=nvme label. The NVMe disks are faster than the SATA SSD disks. Because the cluster has limited resources, the worker01 node includes the application=ml:NoSchedule taint to reserve some workload capacity for a machine learning application that is critical for your company.

Your company requires you to create a machine learning deployment that can run on the node with the taint. The name for the deployment must be review-toleration. This deployment uses the registry.ocp4.example.com:8443/ubi9/ubi:9.0.0-1468 container with eight replicas. Moreover, this deployment requires high availability in the cluster, with no more than 25% of the pods being unavailable. This requirement means that at least six pods must be always running on the cluster. Thus, if you try to drain a node that contains more than 25% of the pods, then OpenShift cannot drain it until the pods that exceed the 25% threshold are placed onto other nodes. The pod disruption budget name must be review-pdb.

Your company also requires you to create a deployment that needs a fast disk. Thus, you must create the deployment with a node selector for the node with the disk=nvme label. The deployment uses the registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 container with four replicas and with the review-ns name.

Finally, your company requires you to create a deployment that must communicate with the review-ns deployment frequently. Thus, you must create this deployment with a required pod affinity rule for the pods in the review-ns deployment. Use the kubernetes.io/hostname label as the topologyKey parameter. The deployment uses the registry.ocp4.example.com:8443/rhel9/mysql-80:1-237 container with four replicas and with the review-affinity name.

Create all the deployments as the developer user with developer as the password in the scheduling-review project. For the deployments, you can use the incomplete CR YAML files in the ~/DO380/labs/scheduling-review directory. If you need administrator permissions, then use the admin user with redhatocp as the password. You might use the count-pods.sh script, which counts the pods of a deployment in each node.

1. As the admin user, verify the labels and taints of your cluster nodes.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

...output omitted...

* 1. List the labels for the compute nodes. Verify that the worker01 and worker02 nodes include the disk=ssd label, and that the worker03 node includes the disk=nvme label.

[student@workstation ~]$ **oc get nodes -L disk**

NAME STATUS ROLES AGE VERSION **DISK**

master01 Ready control-plane,master 140d v1.25.7+eab9cc9

master02 Ready control-plane,master 140d v1.25.7+eab9cc9

master03 Ready control-plane,master 140d v1.25.7+eab9cc9

**worker01** Ready worker 37d v1.25.7+eab9cc9 **ssd**

**worker02** Ready worker 37d v1.25.7+eab9cc9 **ssd**

**worker03** Ready worker 37d v1.25.7+eab9cc9 **nvme**

* 1. Verify the taint for the worker01 node.
  2. [student@workstation ~]$ **oc describe node worker01 | grep Taints**

Taints: application=ml:NoSchedule

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the review-toleration deployment with a toleration for the worker01 node taint.
   1. Connect to the OpenShift cluster as the developer user with developer as the password, and verify that OpenShift uses the scheduling-review project.

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

Login successful.

...output omitted...

Using project **"scheduling-review"**.

* 1. Change to the ~/DO380/labs/scheduling-review directory.

[student@workstation ~]$ **cd ~/DO380/labs/scheduling-review**

* 1. Edit the ~/DO380/labs/scheduling-review/review-toleration.yml file for the review-toleration deployment, and add the toleration for the application=ml:NoSchedule taint.

apiVersion: apps/v1

kind: Deployment

metadata:

name: review-toleration

spec:

...output omitted...

spec:

...output omitted...

**tolerations:**

**- key: "application"**

**value: "ml"**

**operator: "Equal"**

**effect: "NoSchedule"**

* 1. Create the review-toleration deployment with the toleration.

[student@workstation scheduling-review]$ **oc create -f review-toleration.yml**

deployment.apps/review-toleration created

* 1. Verify that the deployment is correctly created. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-review]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

review-toleration 8/8 8 8 53s

* 1. Verify that the OpenShift scheduler distributes the pods between the available compute nodes. OpenShift places some pods in the worker01 node, because the deployment includes a toleration for the taint. The pod names and the nodes that run on your pods might differ on your system.

[student@workstation scheduling-review]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

review-toleration-76f8c74d7-87jjr 1/1 Running ... worker02 ...

review-toleration-76f8c74d7-gtwhx 1/1 Running ... worker03 ...

review-toleration-76f8c74d7-hfssj 1/1 Running ... **worker01** ...

review-toleration-76f8c74d7-lmdcn 1/1 Running ... worker03 ...

review-toleration-76f8c74d7-ph724 1/1 Running ... worker02 ...

review-toleration-76f8c74d7-tntwz 1/1 Running ... **worker01** ...

review-toleration-76f8c74d7-w4tfj 1/1 Running ... worker03 ...

review-toleration-76f8c74d7-zr7d5 1/1 Running ... worker02 ...

* 1. Use the count-pods.sh script to count the pods for the review-toleration deployment in each node. The output might differ on your system, because the OpenShift scheduler distributes the pods across the three compute nodes depending on their workload.

[student@workstation scheduling-review]$ **./count-pods.sh review-toleration**

NODE PODS

worker01 2

worker02 3

worker03 3

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the review-pdb pod disruption budget for the review-toleration deployment. This pod disruption budget ensures that no more than 25% of the pods can become unavailable.
   1. Edit the ~/DO380/labs/scheduling-review/review-pdb.yml file for the review-pdb pod disruption budget.

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: review-pdb

spec:

**maxUnavailable: 25%**

selector:

matchLabels:

**app: review-toleration**

* 1. Create the review-pdb pod disruption budget.

[student@workstation scheduling-review]$ **oc create -f review-pdb.yml**

poddisruptionbudget.policy/review-pdb created

* 1. Verify that the pod disruption budget is correctly created, with a maximum of two unavailable pods.

[student@workstation scheduling-review]$ **oc describe pdb review-pdb**

Name: review-pdb

Namespace: scheduling-review

Max unavailable: 25%

Selector: app=review-toleration

Status:

Allowed disruptions: 2

Current: 8

Desired: 6

Total: 8

Events: <none>

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Verify that the pod disruption budget works as expected, and that you get error messages when you try to drain a node that contains more than 25% of the review-toleration deployment pods. After verifying that the pod disruption budget works as expected, uncordon all nodes to mark them as schedulable.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

...output omitted...

* 1. To test that the pod disruption budget works as expected, first mark the worker02 and the worker03 nodes as unschedulable.

[student@workstation scheduling-review]$ **oc adm cordon worker02 worker03**

node/worker02 cordoned

node/worker03 cordoned

* 1. Perform a rollout restart of the review-toleration deployment to ensure that OpenShift creates all the pods in the worker01 node.

[student@workstation scheduling-review]$ **oc rollout restart \**

**deployment/review-toleration**

deployment.apps/review-toleration restarted

* 1. Verify that all the pods are scheduled to execute in the worker01 node. For any pods in a terminating status, wait until OpenShift removes those pods. You might have to repeat this command many times.

[student@workstation scheduling-review]$ **oc get pods -o wide**

NAME READY STATUS ... NODE ...

review-toleration-76f8c74d7-4494g 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-5jw2q 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-86mf8 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-kp6wp 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-ljxpd 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-nbzj6 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-t78bg 1/1 Running ... worker01 ...

review-toleration-76f8c74d7-tl29v 1/1 Running ... worker01 ...

* 1. Use the count-pods.sh script to count the pods for the review-toleration deployment in each node.

[student@workstation scheduling-review]$ **./count-pods.sh review-toleration**

NODE PODS

worker01 8

worker02 0

worker03 0

* 1. Mark the worker02 and worker03 nodes as schedulable.

[student@workstation scheduling-review]$ **oc adm uncordon worker02 worker03**

node/worker02 uncordoned

node/worker03 uncordoned

* 1. Drain all the pods from the worker01 node. You receive some error messages, because OpenShift can drain the node only after the pods from the review-toleration deployment that exceed the 25% threshold are placed onto other nodes. This action might take a few minutes.

[student@workstation scheduling-review]$ **oc adm drain worker01 \**

**--ignore-daemonsets --delete-emptydir-data**

...output omitted...

evicting pod scheduling-review/review-toleration-76f8c74d7-kp6wp

evicting pod scheduling-review/review-toleration-76f8c74d7-ljxpd

evicting pod scheduling-review/review-toleration-76f8c74d7-5jw2q

**error when evicting pods/"review-toleration-76f8c74d7-5jw2q" -n "scheduling-review" (will retry after 5s): Cannot evict pod as it would violate the pod's disruption budget.**

**error when evicting pods/"review-toleration-76f8c74d7-ljxpd" -n "scheduling-review" (will retry after 5s): Cannot evict pod as it would violate the pod's disruption budget.**

evicting pod scheduling-review/review-toleration-76f8c74d7-nbzj6

**error when evicting pods/"review-toleration-76f8c74d7-nbzj6" -n "scheduling-review" (will retry after 5s): Cannot evict pod as it would violate the pod's disruption budget.**

...output omitted...

node/worker01 drained

* 1. Mark the worker01 node as schedulable.

[student@workstation scheduling-review]$ **oc adm uncordon worker01**

node/worker01 uncordoned

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the review-ns deployment with a node selector for the node with the disk=nvme label, and verify that the pods are scheduled in the node with the disk=nvme label.
   1. Connect to the OpenShift cluster as the developer user with developer as the password.

[student@workstation scheduling-review]$ **oc login -u developer -p developer**

...output omitted...

* 1. Edit the ~/DO380/labs/scheduling-review/review-ns.yml file for the review-ns deployment, and add a node selector for the disk=nvme label.

apiVersion: apps/v1

kind: Deployment

metadata:

name: review-ns

spec:

...output omitted...

spec:

...output omitted...

**nodeSelector:**

**disk: nvme**

* 1. Create the review-ns deployment.

[student@workstation scheduling-review]$ **oc create -f review-ns.yml**

deployment.apps/review-ns created

* 1. Verify that the deployment is correctly created. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-review]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

**review-ns** 4/4 4 4 61s

review-toleration 8/8 8 8 46m

* 1. Verify that the OpenShift scheduler creates all the pods in the worker03 node, which has the disk=nvme label. The pod names might differ on your system.

[student@workstation scheduling-review]$ **oc get pods -o wide -l app=review-ns**

NAME READY STATUS ... NODE ...

review-ns-7475cbc8ff-d7dbq 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-hpl5b 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-p7lhb 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-wjxjz 1/1 Running ... worker03 ...

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create the review-affinity deployment with a required pod affinity rule for the pods in the review-ns deployment. Use the kubernetes.io/hostname node label as the topology key.
   1. Edit the ~/DO380/labs/scheduling-review/review-affinity.yml file for the review-affinity deployment, and add a required pod affinity rule for the pods in the review-ns deployment. Use the kubernetes.io/hostname node label as the topology key.

apiVersion: apps/v1

kind: Deployment

metadata:

name: review-affinity

spec:

...output omitted...

spec:

...output omitted...

**affinity:**

**podAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**- labelSelector:**

**matchExpressions:**

**- key: app**

**operator: In**

**values:**

**- review-ns**

**topologyKey: kubernetes.io/hostname**

* 1. Create the review-affinity deployment.

[student@workstation scheduling-review]$ **oc create -f review-affinity.yml**

deployment.apps/review-affinity created

* 1. Verify that the deployment is correctly created. Wait until all the pods are marked as ready and available. You might have to repeat this command many times.

[student@workstation scheduling-review]$ **oc get deployment**

NAME READY UP-TO-DATE AVAILABLE AGE

**review-affinity** 4/4 4 4 27s

review-ns 4/4 4 4 17m

review-toleration 8/8 8 8 64m

* 1. Verify that all the pods for the review-affinity deployment are placed in the worker03 node. The pod names might differ on your system.

[student@workstation scheduling-review]$ **oc get pods -o wide \**

**-l app=review-affinity**

NAME READY STATUS ... NODE ...

review-affinity-79d6b5bcfd-8qmh8 1/1 Running ... worker03 ...

review-affinity-79d6b5bcfd-cpkhz 1/1 Running ... worker03 ...

review-affinity-79d6b5bcfd-h8sq6 1/1 Running ... worker03 ...

review-affinity-79d6b5bcfd-mcv7k 1/1 Running ... worker03 ...

* 1. Verify that the OpenShift scheduler creates all the pods for the review-affinity deployment in the same compute node as the review-ns deployment. The pod names might differ on your system.

[student@workstation scheduling-review]$ **oc get pods -o wide -l app=review-ns**

NAME READY STATUS ... NODE ...

review-ns-7475cbc8ff-d7dbq 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-hpl5b 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-p7lhb 1/1 Running ... worker03 ...

review-ns-7475cbc8ff-wjxjz 1/1 Running ... worker03 ...

* 1. Change to the student HOME directory.

[student@workstation scheduling-review]$ **cd**

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

**Evaluation**

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

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

# **Chapter 5.  OpenShift GitOps**

[GitOps for Kubernetes](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05)

[Quiz: GitOps for Kubernetes](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s02)

[GitOps for Cluster Administration](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s03)

[Guided Exercise: GitOps for Cluster Administration](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s04)

[GitOps for Application Management](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s05)

[Guided Exercise: GitOps for Application Management](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s06)

[Lab: OpenShift GitOps](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s07)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch05s08)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Deploy OpenShift GitOps for managing clusters and applications. |
| **Sections** | * GitOps for Kubernetes (and Quiz) * GitOps for Cluster Administration (and Guided Exercise) * GitOps for Application Management (and Guided Exercise) |
| **Lab** | * OpenShift GitOps |

## **GitOps for Kubernetes**

### **Objectives**

* Define the fundamentals of GitOps and its use with Kubernetes clusters and applications.

### **Introduction to GitOps**

Red Hat OpenShift GitOps is an operator that helps implement GitOps practices to manage OpenShift clusters.

Kubernetes can import and export resource definitions as text files. By working with resource definitions in text files, administrators describe their workloads instead of using a sequence of operations to create them. This approach is called declarative resource management.

By storing resource definitions in text files, Kubernetes administrators can use any tool or process for text files. For example, by keeping cluster resource definitions in a version control system, administrators can track changes to resources.

GitOps or infrastructure as code are terms to describe practices that relate to these concepts.

By enabling cluster users to manage Kubernetes resources by using a Git repository, administrators can further restrict direct access to the cluster. If users can modify Kubernetes resources only through a Git repository, then administrators can use Git features to enforce controls such as reviews and approvals. For example, many organizations use pull request workflows to update repositories.

Additionally, cluster management presents significant challenges when multiple processes can update cluster resources. Cluster administrators can face problems if they cannot clearly visualize the intended state of the cluster. When multiple users and processes update the state of the cluster, the intended state for the cluster can become unclear. The term drift describes the difference between the intended and actual states of a cluster.

With GitOps, a single process changes the cluster based on a definition of the intended cluster state. Administrators can then reference a single cluster definition to track what is happening with the cluster. Also, administrators who invest resources in documenting their changes can have reliable information about each change to the cluster.

Additionally, with a single cluster definition, administrators can reproduce the deployments of a cluster on separate clusters. For example, a cluster administrator can create a test cluster based on their production cluster definition. Then, the administrator can test cluster updates and other changes, some of which might be risky or hard to revert in a production environment, with reasonable guarantees that results from the test cluster apply to the production cluster. Production clusters often have different loads and interact with external services, so test clusters cannot reproduce the exact conditions. However, tests can provide the level of confidence that your organization requires for such changes.

### **GitOps and Kubernetes**

Although the Kubernetes resource model is suited to GitOps, some Kubernetes idiosyncrasies require extra care.

Kubernetes resources contain information about both the definition and the status of the resource.

Consider the following excerpt from a Kubernetes deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

creationTimestamp: "2023-12-14T17:02:18Z"

generation: 1

name: hello-node

namespace: test

...output omitted...

spec:

replicas: 1

selector:

matchLabels:

app: hello-node

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

creationTimestamp: null

labels:

app: hello-node

spec:

containers:

- command:

- /agnhost

- serve-hostname

image: registry.k8s.io/e2e-test-images/agnhost:2.43

name: agnhost

restartPolicy: Always

schedulerName: default-scheduler

status:

availableReplicas: 1

conditions:

- lastTransitionTime: "2023-12-14T17:02:22Z"

lastUpdateTime: "2023-12-14T17:02:22Z"

message: Deployment has minimum availability.

reason: MinimumReplicasAvailable

status: "True"

type: Available

...output omitted...

observedGeneration: 1

readyReplicas: 1

replicas: 1

updatedReplicas: 1

The status key contains information about the number of available replicas, which can change during the lifetime of the deployment. Although the content of the metadata and spec keys is mostly the definition of the deployment, Kubernetes adds some fields, such as the creationTimestamp field, which are not necessary to re-create the deployment. Additionally, you might choose not to include in your resource definition some fields in the spec key with default values, such as the strategy key.

The kubectl command has subcommands to manipulate resource definitions, such as the apply, edit, and patch subcommands. These commands handle some complexities with updating resources.

Other Kubernetes features, such as mutating admission webhooks, can modify resources. These features can increase the complexity of updating resources.

Finally, although you can usually create resources in any order, with resources ultimately reaching the intended state, in some cases creating resources can bring more complexity. For example, you must create a namespace before creating the resources in the namespace. You can create custom resources only after the installation of the operator deploys the custom resource definitions.

Due to these factors, implementing GitOps involves additional sophistication. Simple GitOps implementations that are based on built-in features, such as the kubectl apply command, can present problems when resources increase in complexity.

System administrators can use other tools, such as Terraform and Ansible, to implement GitOps practices. Tools might differ in their approach and scope. Argo CD focuses on synchronizing Kubernetes resources from Git repositories to the cluster.

### **Introduction to OpenShift GitOps**

OpenShift GitOps is an operator that manages Argo CD in OpenShift clusters. Argo CD is a tool that can synchronize Kubernetes resources to definitions that are stored in Git repositories.

Cluster administrators can use OpenShift GitOps to create Argo CD instances with specific access rights. Administrators can create an Argo CD instance that can manage specific namespaces and grant specific users access to the instance. With these instances, teams can be autonomous in deploying applications to specific namespaces.

|  |
| --- |
|  |

Figure 5.1: OpenShift GitOps architecture for developers

OpenShift GitOps also includes a default administrative Argo CD instance that can manage Kubernetes resources that require administrative access, such as non-namespaced resources or authentication resources. OpenShift administrators can use Argo CD instances with these configurations to manage the cluster itself.

|  |
| --- |
|  |

Figure 5.2: OpenShift GitOps architecture for cluster administrators

Argo CD uses the concept of applications. An application is an Argo CD resource that references a Git repository with Kubernetes resource definitions. Users can create and manage Argo CD applications from the web console or with the Kubernetes API.

Applications define a synchronization policy. Argo CD can detect and synchronize changes automatically, or users can trigger synchronizations on demand.

The Argo CD web console displays the status of application resources graphically. This representation includes both the resources that are defined in the application, and resources that are related to the application. For example, for an application that contains an operator subscription, the Argo CD web console also shows the install plans, operator workloads, and other related resources.

In this representation, you can view whether the resources are synchronized with the application definition, and inspect their status.

With OpenShift GitOps, cluster administrators can restrict permissions so that only Argo CD instances can update cluster resources, and Git repositories fully describe the intended cluster state. Even if other processes can modify the cluster state, Argo CD can detect and address issues with drift. Additionally, cluster administrators might use Argo CD to manage multiple clusters from the same cluster definition. Besides testing, having a cluster definition can help with other tasks, such as sharing resource definitions between clusters.

### **Continuous Integration, Continuous Delivery, and Continuous Deployment**

The Argo CD name refers to the continuous delivery and continuous deployment practices. The CI​/​CD abbreviation refers to these practices and to continuous integration.

**Continuous integration**

Originally, continuous integration referred to integrating changes often to reduce conflicts from developing changes in isolation for long periods. Nowadays, continuous integration often refers to running tests automatically, and to practices such as requiring tests to pass before merging changes.

**Continuous delivery**

Continuous delivery is the practice of making each change a potential release of the project. With continuous delivery, the team can deploy the project at any time.

**Continuous deployment**

Continuous deployment is a variant of continuous delivery where every change triggers an automated deployment. Continuous deployment maximizes the release frequency of the project and shortens feedback loops.

CI​/​CD practices aim to reduce the lead time to deliver a new product or feature. When production issues are found, you can roll back small increments to known stable states.

OpenShift GitOps helps implement continuous delivery and deployment, but does not cover continuous integration. Red Hat provides other products that can help with implementing CI​/​CD:

**Red Hat OpenShift Pipelines**

The OpenShift Pipelines product is based on the open source Tekton project, and can run processes when changes occur to a Git repository. OpenShift Pipelines can implement the modern interpretation of CI, by validating changes. OpenShift Pipelines can also update container images with changes to source repositories.

With third-party applications, the application provider often provides updated images that you can deploy with Argo CD. However, with in-house applications, you can create updated images with OpenShift Pipelines.

**Jenkins**

Jenkins is a generic automation tool, to implement any type of CI​/​CD process.

For OpenShift automation, Red Hat recommends OpenShift Pipelines and OpenShift GitOps instead of Jenkins. The functions of the latter products require extra effort to implement with Jenkins. The OpenShift documentation covers migrating from Jenkins to OpenShift Pipelines.

By combining these products to implement CI​/​CD practices, organizations can create highly automated workflows that speed up development and that follow the necessary quality and security guidelines.

### **GitOps Workflows**

The following steps describe an example of such a workflow:

* A developer updates the source code of an application.
* The developer pushes the change to a Git repository branch that contains the source code of the application, and creates a pull request.
* OpenShift Pipelines detects the new pull request and runs automated checks:
  + The application source code is checked for mechanical errors and incorrect formatting. OpenShift Pipelines posts an update to the Git repository, to indicate whether the pull request passed the checks or conversely contains errors.
  + OpenShift Pipelines triggers an Argo CD job that deploys the application with the specified change to a testing environment. Any team members can access the deployment to validate the change. Argo CD can mark the change as failed if the application fails to deploy.
* After the changes are validated, other team members can review the pull request. If validation failed, then a team member can fix the issues and resubmit. With this process, reviewers review only changes that passed automated validation.
* For any issues that reviewers find in the change, they can request changes from the original developer to address issues.
* Approved changes are merged.
* OpenShift Pipelines detects the merged change and builds an updated container image from the source code of the application, and triggers an Argo CD synchronization of the production environment.
* Argo CD deploys the updated image.

To ensure adequate velocity, the organization should ensure that the automated processes can complete quickly, and limit manual approvals to what is strictly required. For example, minor changes might not require a review; typical changes might require a single approver; and significant changes might require approvals from multiple specific stakeholders. These processes aim to automate and speed up the delivery of changes, and introduce only necessary validation. Organizations can choose their tradeoffs between speed and quality, according to their needs.

### Warning

Following GitOps procedures can reduce security risks by introducing validation steps. However, version control systems such as Git can also introduce security risks.

For example, if you push sensitive data to a repository, then although your process might prevent pushing the data to production, unintended people might access the sensitive data. Furthermore, permanently removing information from a Git repository can require significant effort. Invalidating the data (for example, by creating an authentication token and disabling the leaked token) can be more efficient than permanently removing the sensitive data from the repository.

You can use elements, such as Git hooks, to improve the security of version control systems. Git hooks can prevent users from pushing sensitive information from their private workstation environment to a more accessible environment.

The following list describes some improvements to GitOps workflows that Argo CD can implement:

* Propagate changes through multiple static environments, such as testing, pre-production, and production environments.
* Limit synchronizations to specific days of the week or periods during the day.
* Add hooks for further processes, such as for sending notifications, pausing monitoring, or triggering backups before an Argo CD synchronization.

## **GitOps for Cluster Administration**

### **Objectives**

* Configure the default Argo CD instance from OpenShift GitOps for cluster administration.

### **The Red Hat OpenShift GitOps Operator**

Red Hat OpenShift GitOps is an operator in OperatorHub, from the Red Hat operator catalog. OperatorHub provides a latest channel, with the most recent stable version of the operator. OperatorHub also provides gitops-*version* channels, such as the gitops-1.10 channel, to install specific versions of the operator.

You can install the operator in a similar way to any other operator, for example by using the installation wizard from the web console. You can use the default operator installation options, although most deployments require customization after the operator installation.

The operator creates its workload in the openshift-gitops-operator namespace. This workload includes a deployment that monitors Argo CD custom resources and manages their workloads.

### **Argo CD Custom Resources**

The OpenShift GitOps operator installs the ArgoCD custom resource definition. Cluster administrators can use this custom resource definition to declaratively create multiple Argo CD instances in the cluster. The operator manages the workloads to run the Argo CD instances.

With this capability, administrators can introduce instances with different permissions. Because Argo CD applications can create and modify any type of OpenShift resource, limiting the permissions of Argo CD instances can reduce risks. With the OpenShift GitOps operator, you can create Argo CD instances by creating a custom resource that describes an instance (including RBAC configuration), and the operator manages the Argo CD instance. This process helps administrators apply the principle of least privilege, by creating Argo CD instances with granular permissions.

### Warning

Any user that can push resources to an application Git repository can perform operations with the permissions of the Argo CD instance, even without direct access to the Kubernetes API or Argo CD. For example, these operations might delete data or override access control, whether by accident or malicious intent. Although Argo CD has role-based access control for its resources, Argo CD trusts the content of the application Git repositories. Follow the same considerations for access control for the Git repositories as for granting cluster permissions.

You can also set repositories as private, and configure Argo CD to use credentials to access repositories.

OpenShift GitOps includes further resources. For example, Argo CD uses the Application custom resource to manage applications. When you create an application with the Argo CD web interface, Argo CD creates an Application custom resource. This course does not cover working with those resources.

### **The Default Argo CD Instance**

By default, the OpenShift GitOps operator creates a default Argo CD instance with the openshift-gitops name for both the instance and the namespace. This default instance has a set of permissions to support cluster administration. Red Hat recommends using this instance only for administering the cluster or for creating other instances for regular application deployment with minimal privileges.

The default instance has a limited configuration that might require further configuration to meet organizational requirements.

With the openshift-gitops-server route in the openshift-gitops namespace, the Argo CD web console is accessible outside the OpenShift cluster. By default, the web console is at the openshift-gitops-server-openshift-gitops.apps.*cluster\_domain* URL.

#### **Setting the Default Argo CD Instance as Unmanaged**

The operator manages the default Argo CD instance. If you modify the instance, then the operator can revert your changes.

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

creationTimestamp: "2024-01-02T11:27:35Z"

finalizers:

- argoproj.io/finalizer

generation: 1

name: openshift-gitops

namespace: openshift-gitops

**ownerReferences:**

- apiVersion: pipelines.openshift.io/v1alpha1

blockOwnerDeletion: true

controller: true

kind: GitopsService

name: cluster

uid: a7a0a073-0328-449e-8ffc-038854feb1c9

resourceVersion: "262080"

uid: 1d530e04-8068-4f6f-a066-c3ae36ba34f4

spec:

applicationSet:

...output omitted...

For example, if you remove the ownerReferences key from the instance metadata, then the operator does not restore the instance.

However, setting the default instance as unmanaged might prevent operator updates from applying updates to the default instance. You can also consider ignoring the default instance (or setting it as unmanaged and deleting it, to save resources), and always creating your own instance. Depending on the criticality of your environment, consider extra testing of operator updates in a non-production environment.

### **Configuring Argo CD RBAC**

Users can interact with Argo CD through the Kubernetes API, or through the Argo CD API, web interface, or command-line tool. Each Argo CD instance has a role-based access control system to manage permissions in the Argo CD interfaces.

Argo CD can use both local users that are defined in Argo CD and users from external authentication systems. The OpenShift GitOps operator includes features to automate integration with the OpenShift cluster authentication.

The default Argo CD instance includes a local admin user. This user has full permissions in the Argo CD interfaces. Argo CD updates the cluster by using a Kubernetes service account. Even unrestricted users in Argo CD are limited in how they can update the cluster, according to the permissions of the Argo CD service account.

The openshift-gitops-cluster secret in the openshift-gitops namespace contains the password for this user. Although this user can be suitable for small environments or experiments in test clusters, administrators usually integrate Argo CD with other authentication sources, such as the OpenShift cluster authentication.

The following excerpt from the default Argo CD instance shows the default authentication configuration:

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

name: openshift-gitops

namespace: openshift-gitops

...output omitted...

spec:

...output omitted...

**rbac:**

defaultPolicy: ""

policy: |

g, system:cluster-admins, role:admin

g, cluster-admins, role:admin

scopes: '[groups]'

...output omitted...

**sso:**

dex:

openShiftOAuth: true

...output omitted...

**rbac**

The rbac key contains the Argo CD role-based access control configuration. This configuration grants the admin role to the cluster-admins group.

You can create a cluster-admins group in the cluster and add users to have administrator privileges for the Argo CD instance. Alternatively, you can edit the policy to grant administrator privileges to existing groups.

### Warning

Argo CD resources, such as applications, are Kubernetes resources. Argo CD RBAC controls the use of resources through Argo CD, not through the Kubernetes API.

Use Kubernetes role-based access control to prevent access to Argo CD Kubernetes resources through the Kubernetes API.

**sso**

The sso key contains the authentication configuration. In the default Argo CD instance, the openShiftOAuth key is set to the true value. With this configuration, OpenShift users can log in to the Argo CD web console by clicking **LOG IN VIA OPENSHIFT**. By default, OpenShift users have limited privileges.

### **Configuring Trusted Certificates for Git Repository Access**

Argo CD validates HTTPS certificates when accessing Git repositories, and can access only trusted repositories.

The repo server component of Argo CD handles communicating with Git repositories. To change the trusted certificate authorities, modify the certificate bundle in the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path of the repo server.

The Argo CD custom resource contains fields to customize the repo server, similar to the volumes and volumeMounts keys of a pod template. The following example shows how to replace the certificate bundle with a bundle that is stored in a configuration map:

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

name: openshift-gitops

namespace: openshift-gitops

...output omitted...

spec:

...output omitted...

repo:

**volumeMounts:**

**- mountPath: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**

**name: *volume\_name***

**subPath: *bundle\_name***

**volumes:**

**- configMap:**

**name: *configuration\_map\_name***

**name: *volume\_name***

...output omitted...

#### **Configuring the Cluster Certificate Authority**

If your repositories use certificates that the cluster certificate authority signed (for example, you expose them by using OpenShift routes), then you can use certificate injection to help configure Argo CD to trust the repositories.

Create an empty configuration map in the openshift-gitops namespace. Add the config.openshift.io/inject-trusted-cabundle label to the configuration map with the true value. OpenShift adds the bundle with the cluster certificate authority to the configuration map, in the ca-bundle.crt key. Then, you can follow the previous example to configure the repo server to trust the bundle.

### **Customizing the Argo CD Web Interface Certificate**

The default Argo CD instance has its own self-signed HTTPS certificate. The Argo CD resource has a termination key to change the route termination. Use the reencrypt termination so that Argo CD uses the default router certificate.

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

...output omitted...

spec:

...output omitted...

server:

...output omitted...

route:

enabled: true

**tls:**

**termination: reencrypt**

...output omitted...

### **The Argo CD Web Interface**

By default, the web console for an Argo CD instance is at the *argo\_cd\_instance*-server-*namespace*.apps.*cluster\_domain* URL. Log in to Argo CD by accessing this URL.

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

Figure 5.3: The Argo CD login page

If the Argo CD instance is configured with the openShiftOAuth key, then click **LOG IN VIA OPENSHIFT** to log in by using the cluster authentication.

Alternatively, you can use the **Username** and **Password** fields, and the **SIGN IN** button to use Argo CD local users.

When you log in, the web console displays the **Applications** page. The web console contains a navigation panel:

You can view and create applications in the **Applications** page. You can use the **User Info** page to view details about the logged-in user. These details can be useful to troubleshoot authentication issues.

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

Figure 5.4: The Argo CD applications page

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

Figure 5.5: Application details

The application details page displays the Kubernetes resources that are associated with an Argo CD application. For each resource, you can view the synchronization status and resource details.

### Important

When Argo CD displays a large amount of resources, the web interface might not be easy to work with.

Because Argo CD displays both the resources in your application and related resources, more resources might be displayed than expected.

To reduce the number of displayed resources, you can collapse resources to hide dependent resources.

With large applications, you might need to use other means of examining resources, such as the kubectl and oc commands, or the Kubernetes API.

#### **Creating Applications**

You can create Argo CD applications by using the Kubernetes API, the argocd command-line application, or the web console.

By default, the Argo CD web console shows a form to customize the application. The console also provides a YAML resource editor, which is similar to the OpenShift web console.

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

Figure 5.6: Application creation form

Use the following fields to create your application:

**Application Name**

The application resource name.

**Project Name**

Argo CD uses projects to group applications and to apply further access controls. Projects are not covered in this course. You can create applications in the default project that Argo CD provides.

**Sync Policy**

The **Automatic** policy synchronizes periodically. With the **Manual** policy, administrators must trigger synchronizations.

**Retry**

As described later in this lecture, you can use these fields to add retries to the synchronization process. You might need to configure retries when working with complex applications.

**Repository URL**

The URL of the repository with the application definition. If you use the HTTPS protocol to access the repository, then the repo server must trust the HTTPS certificate.

**Path**

The path in the Git repository that contains the resource definitions. You can define multiple applications from different paths in the same repository. A repository might also contain other content.

**Cluster URL**

The URL of the Kubernetes API for the cluster to deploy the application to. Use the https://kubernetes.default.svc URL to deploy to the same cluster where Argo CD is deployed.

When you create an application, Argo CD retrieves the contents of the repository. In the exercises of this course, applications are YAML files that describe application resources. Argo CD supports other ways to define applications, including the following technologies:

* Kustomize
* Helm charts
* Jsonnet
* Other configuration management tools that additional plug-ins support

### **Deploying Complex Resources by Using Sync Waves and Retries**

Although you can usually create resources in any order and ultimately resources reach the intended state, in some cases creating resources can require further work. For example, you must create a namespace before creating the resources in the namespace. With custom resources, you usually need to wait until their operator installs the custom resource definitions.

You can use the argocd.argoproj.io/sync-wave annotation on resources to help with dependencies. The annotation value must be a positive or negative integer or zero. Resources without the annotation default to 0.

A group of resources with the same annotation is a sync wave. Argo CD synchronizes sync waves in order, starting with sync waves with lower numbers. Argo CD synchronizes a sync wave only when all lower sync waves are healthy.

Argo CD performs a dry run before synchronizations. When deploying custom resources, the synchronization continues only if the corresponding custom resource definitions exist when Argo CD performs the dry run. To prevent this problem, add the argocd.argoproj.io/sync-options annotation to these resources by adding the SkipDryRunOnMissingResource=true option.

For example, the following manifest defines an operator subscription and a custom resource:

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: example-operator

...output omitted...

---

kind: ExampleCustomResource

metadata:

name: example

annotations:

**argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true**

**argocd.argoproj.io/sync-wave: "1"**

...output omitted...

Because the subscription does not have an explicit sync wave, Argo CD considers the subscription to be part of the 0 sync wave. Because the custom resource belongs to the 1 sync wave, Argo CD creates the custom resource only when the subscription is healthy. The sync options also exclude the custom resource from the dry run.

However, the previous example might still fail. The next section describes additional issues with deploying complex applications.

#### **Eventual Consistency in Kubernetes**

Kubernetes is a resilient distributed system, to fulfill its requirement to be a framework to run distributed systems resiliently. Building resilient distributed systems adds significant complexity.

Kubernetes uses the eventual consistency model to help achieve reliability and also preserving scalability. In distributed systems, changes must propagate to all nodes. Because systems that use eventual consistency do not apply updates immediately, updates can take a perceivable amount of time to be effective across the cluster or even in a single node.

Eventual consistency avoids some scalability and reliability issues. However, non-immediate updates add complexity to many processes, because some operations might need to wait until other operations are effective. For changes for non-immediate updates, one option is to retry operations until they succeed after dependent operations take effect. Software that works with Kubernetes often uses retries to work correctly.

Sync waves are often insufficient when working with complex applications such as operators. If you face issues with deploying applications, then consider configuring retries in your Argo CD applications for more reliable synchronization.

However, retries can clutter logs, slow down feedback loops, and hide problems. To limit problems that derive from excessive retries, consider investing effort in finding an adequate retry configuration.

### **Administrating Clusters with OpenShift GitOps**

Although organizations often use Argo CD to deploy applications, cluster administrators can also use the default Argo CD instance to administer a cluster. By using Argo CD, administrators can customize authentication or deploy operators with all the benefits of GitOps.

When administering a cluster with Argo CD, you might need to patch existing resources, which is an uncommon scenario when deploying applications.

For example, cluster authentication uses the cluster resource of the OAuth type. You must edit this resource instead of creating other resources.

Argo CD can use the server-side apply process to edit existing resources. The server-side apply process is an alternative to the client-side apply process that kubectl apply and other processes use by default.

With client-side apply, commands such as kubectl apply submit the full definition of a resource to the API server. The API server overwrites the entire resource with the submitted definition.

With server-side apply, API clients submit partial resource definitions, and the API server updates only the submitted parts. Then, your application definitions can include only your changes and exclude parts of the resource that you do not want to manage.

For patching, use the ServerSideApply=true sync option. Also, add the Validate=false sync option to disable validation, because the partial resource definition might not be a valid complete resource definition.

The following manifest shows how to patch a resource:

apiVersion: operator.openshift.io/v1

kind: Console

metadata:

name: cluster

annotations:

**argocd.argoproj.io/sync-options: ServerSideApply=true,Validate=false**

spec:

customization:

customProductName: **Production**

This specification does not contain all the fields that a full console specification requires. The Validate=false sync option ensures that even if the partial definition lacks mandatory fields, Argo CD submits the change.

This change is less likely to interfere with other changes to the cluster such as cluster updates.

### References

For more information about the installation process, refer to the Installing Red Hat OpenShift GitOps chapter in the Red Hat OpenShift GitOps 1.10 Installing GitOps documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/installing_gitops/index#installing-openshift-gitops>

For more information about certificate injection, refer to the Certificate Injection Using Operators section in the Configuring a Custom PKI chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#certificate-injection-using-operators_configuring-a-custom-pki>

For more information about Argo CD role-based access control, refer to the Access Control and User Management chapter in the Red Hat OpenShift GitOps 1.10 Access Control and User Management documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/access_control_and_user_management/index#configuring-argo-cd-rbac>

For more information about cluster administration with OpenShift GitOps, refer to the Declarative Cluster Configuration chapter in the Red Hat OpenShift GitOps 1.10 Declarative Cluster Configuration documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/declarative_cluster_configuration/index#configuring-an-openshift-cluster-by-deploying-an-application-with-cluster-configurations>

[Kubernetes: Server-Side Apply](https://kubernetes.io/docs/reference/using-api/server-side-apply/)

## **Guided Exercise: GitOps for Cluster Administration**

Deploy an add-on operator and custom resources by using GitOps.

**Outcomes**

* Install Red Hat OpenShift GitOps.
* Configure the default Argo CD instance.
* Create an Argo CD application that deploys an operator and custom resources from the operator.
* Patch a cluster resource by using server-side apply.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start gitops-admin**

**Instructions**

Your cluster had downtime. During incident analysis, system administrators discovered changes to the cluster without proper documentation.

Your organization wants to implement an audit track for changes to the cluster. You decide to experiment with Red Hat OpenShift GitOps to achieve this objective.

Your experiment is to create a test cluster, install OpenShift GitOps, and perform some administrative changes to the test cluster. You test installing and using an operator, and patching existing resources.

1. As the admin user, locate and then navigate to the Red Hat OpenShift web console.
   1. Use the terminal to log in to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

...output omitted...

* 1. Identify the URL for the OpenShift web console.

[student@workstation ~]$ **oc whoami --show-console**

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

* 1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com. Either type the URL in a web browser, or right-click and select **Open Link** from the terminal.
  2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.

1. Install the OpenShift GitOps operator from OperatorHub.
   1. Navigate to **Operators** → **OperatorHub**.
   2. Click **Red Hat OpenShift GitOps**, and then click **Install**.
   3. Review the default configuration and click **Install**. The Operator Lifecycle Manager (OLM) can take a few minutes to install the operator. Click **View Operator** to navigate to the operator details.
   4. Open a separate tab and open the default Argo CD instance. You can use the application menu, which is the grid icon on the top navigation bar, by clicking **Cluster Argo CD**. You can also use the https://openshift-gitops-server-openshift-gitops.apps.ocp4.example.com URL.

The browser displays a warning because Argo CD uses a self-signed certificate. Argo CD might take a few minutes before starting to handle requests.

1. Disconnect the default instance from the operator so the operator does not revert changes to the instance.

If you use the web console for this operation, then click the **Argo CD** tab to display the default openshift-gitops Argo CD instance in the openshift-gitops namespace. Click the name of the instance, and then click the **YAML** tab to display the resource editor. Perform the modifications that are described later, and then click **Save**.

If you use the terminal for this operation, then run the following command:

[student@workstation ~]$ **oc edit -n openshift-gitops argocd openshift-gitops**

Remove the ownerReferences key from the metadata key in the resource, and save your changes. The resulting resource definition should resemble this extract:

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

creationTimestamp: "2023-11-29T18:47:29Z"

finalizers:

- argoproj.io/finalizer

generation: 1

name: openshift-gitops

namespace: openshift-gitops

resourceVersion: "398331"

uid: 00778335-f7d8-457d-92c2-121cd13f5d26

spec:

applicationSet:

...output omitted...

1. For the route of the default instance, change the termination type to the reencrypt type. Edit the Argo CD resource to match the following example:

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

...output omitted...

spec:

...output omitted...

server:

...output omitted...

route:

enabled: true

**tls:**

**termination: reencrypt**

...output omitted...

Reload the Argo CD browser tab. Instead of the certificate warning, the Argo CD login page is shown.

1. Grant administrator rights to the ocpadmins group.
   1. Edit the Argo CD resource to match the following example:

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

...output omitted...

spec:

...output omitted...

prometheus:

enabled: false

ingress:

enabled: false

route:

enabled: false

rbac:

policy: |

**g, ocpadmins, role:admin**

scopes: '[groups]'

redis:

...output omitted...

1. Configure the default instance to trust the cluster certificate authority. Argo CD accesses only trusted repositories.
   1. Create a cluster-root-ca-bundle configuration map in the openshift-gitops namespace.

[student@workstation ~]$ **oc create configmap -n openshift-gitops \**

**cluster-root-ca-bundle**

* 1. Add the config.openshift.io/inject-trusted-cabundle label to the configuration map with the true value. OpenShift injects the bundle with the cluster certificate authority into the configuration maps with this label. This bundle contains the signing certificate for the classroom GitLab instance.

[student@workstation ~]$ **oc label configmap -n openshift-gitops \**

**cluster-root-ca-bundle config.openshift.io/inject-trusted-cabundle=true**

configmap/cluster-root-ca-bundle labeled

* 1. Edit the Argo CD default instance to inject the bundle.

You can use the following command to edit the resource:

[student@workstation ~]$ **oc edit argocd -n openshift-gitops openshift-gitops**

Edit the resource to mount the ca-bundle.crt file in the configuration map to the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path of the repository server container.

...output omitted...

spec:

...output omitted...

repo:

resources:

limits:

cpu: "1"

memory: 1Gi

requests:

cpu: 250m

memory: 256Mi

**volumeMounts:**

**- mountPath: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**

**name: cluster-root-ca-bundle**

**subPath: ca-bundle.crt**

**volumes:**

**- configMap:**

**name: cluster-root-ca-bundle**

**name: cluster-root-ca-bundle**

resourceExclusions: |

...output omitted...

1. Create a public repository in the classroom GitLab.
   1. Open a web browser and navigate to https://git.ocp4.example.com. Log in as the developer user with d3v3lop3r as the password.
   2. Click **New project**, and then click **Create blank project**. Use gitops-admin as the project slug (repository name), select the **Public** visibility level, and use the default values for all other fields. Click **Create project**.
2. Populate the repository.
   1. Click **Clone**, and then copy the https://git.ocp4.example.com/developer/gitops-admin.git HTTPS URL.
   2. Change to the ~/DO380/labs/gitops-admin directory.

[student@workstation ~]$ **cd ~/DO380/labs/gitops-admin**

* 1. In a terminal, run the following command to clone the new repository.

[student@workstation gitops-admin]$ **git clone \**

**https://git.ocp4.example.com/developer/gitops-admin.git**

Cloning into 'gitops-admin'...

...output omitted...

* 1. Change to the cloned repository directory.

[student@workstation gitops-admin]$ **cd gitops-admin**

The default configuration for new repositories adds a README.md initial file.

* 1. Copy the provided operator.yaml file to the repository.

[student@workstation gitops-admin]$ **cp ../operator.yaml .**

* 1. Examine the file and edit the file to match the following text:

apiVersion: v1

kind: Namespace

metadata:

name: openshift-compliance

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

...output omitted...

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: compliance-operator

...output omitted...

---

apiVersion: compliance.openshift.io/v1alpha1

profiles:

- apiGroup: compliance.openshift.io/v1alpha1

name: rhcos4-moderate

kind: Profile

settingsRef:

apiGroup: compliance.openshift.io/v1alpha1

name: default

kind: ScanSetting

kind: ScanSettingBinding

metadata:

name: nist-moderate

namespace: openshift-compliance

annotations:

**argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true**

**argocd.argoproj.io/sync-wave: "1"**

The file contains manifests to install the compliance operator, with the namespace, operator group, and subscription.

The file also contains a scan setting binding. The compliance operator examines scan setting bindings and scans the cluster after they are defined.

Because scan setting bindings are custom resources that the compliance operator installs, you must use extra configuration so that Argo CD can create the resource correctly:

* + - The dry run must be skipped, because when the Argo CD application is created, the custom resource definition does not exist, so the Argo CD validation would fail.
    - The scan setting binding must be created after the custom resource definition exists. To delay the creation, you specify that the scan setting binding is created in a sync wave after the default sync wave. However, due to the Kubernetes use of eventual consistency, later you also configure retries for the application deployment to prevent further issues.
  1. Add the operator.yaml file to the Git index.

[student@workstation gitops-admin]$ **git add operator.yaml**

* 1. Commit the changes.

[student@workstation gitops-admin]$ **git commit -m "Add compliance manifests"**

[main 6785970] Add compliance manifests

...output omitted...

* 1. Push the changes to the repository. Use the developer user with d3v3lop3r as the password.

[student@workstation gitops-admin]$ **git push**

...output omitted...

1. Log in to the Argo CD web console as the admin user.
   1. Go back to the Argo CD browser tab.
   2. Click **LOG IN VIA OPENSHIFT**, and then click **Red Hat Identity Management**. Log in as the admin user with redhatocp as the password, and then allow the **user:info** permission.
2. Create an application with the repository and observe the results.
   1. Click **CREATE APPLICATION**.
   2. Create an application with the information in the following table:

| **Field** | **Value** |
| --- | --- |
| Application Name | gitops-admin |
| Project Name | default |
| Retry | Checked |
| Repository URL | https://git.ocp4.example.com/developer/gitops-admin.git |
| Path | . |
| Cluster URL | https://kubernetes.default.svc |

* 1. Then, click **CREATE**.

1. Synchronize the application.
   1. Click **gitops-admin** to view the application.
   2. Click **SYNC** to display the synchronization panel, and then click **SYNCHRONIZE**.

Argo CD starts synchronizing the application. After about one minute, the console shows the compliance operator as synchronized and healthy.

If you wait about four minutes, then the compliance operator scans and creates a compliance suite custom resource.

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

You can display the results of the scan from Argo CD by clicking the **nist-moderate** compliance suite. The compliance suite is in the done phase, with the non-compliant result.

You can also monitor the progress of the scan by running the watch oc get compliancesuite -A command.

### Note

The many resources that the application tracks can cause the Argo CD web console to become unstable. If the web console does not respond, then you can close the browser tab and reopen the Argo CD web console.

Collapsing the resources in the Argo CD web console can help with exploring the resources.

1. Customize the OpenShift console.
   1. Copy the provided console.yaml file to the repository.

[student@workstation gitops-admin]$ **cp ../console.yaml .**

* 1. Edit the file to match the following text:

apiVersion: operator.openshift.io/v1

kind: Console

metadata:

name: cluster

annotations:

**argocd.argoproj.io/sync-options: ServerSideApply=true,Validate=false**

spec:

customization:

customProductName: **Production**

Because the cluster console resource exists, this manifest defines only a patch to update the product name. For Argo CD to patch the resource, you must use an annotation. Because some patches can be valid resources, but other patches can be invalid resources, disable validation to ensure that patches are not validated as full resources.

* 1. Add the console.yaml file to the Git index.

[student@workstation gitops-admin]$ **git add console.yaml**

* 1. Commit the changes.
  2. [student@workstation gitops-admin]$ **git commit -m "Customize console"**

...output omitted...

* 1. Push the changes to the repository.

[student@workstation gitops-admin]$ **git push**

...output omitted...

* 1. Click **SYNC** to display the synchronization panel, and then click **SYNCHRONIZE**.
  2. Reload the OpenShift web console to observe the changes. After you reload the console, the browser tab title ends with the **Production** text.

1. Change to the student HOME directory.

[student@workstation gitops-admin]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish gitops-admin**

## **GitOps for Application Management**

### **Objectives**

* Deploy an Argo CD instance, from OpenShift GitOps, for application developers and application administrators.

### **Argo CD Instances**

By default, the Red Hat OpenShift GitOps operator creates an Argo CD instance in the openshift-gitops namespace. This Argo CD instance has permissions to manage cluster-wide resources. Thus, Red Hat recommends using this instance only for administration purposes, and to create other, less privileged instances for regular application deployment.

To deploy regular applications, you can deploy new Argo CD instances. By default, the new Argo CD instances have permissions to manage resources only in the namespace where they are deployed.

To create an Argo CD instance, you can create a YAML file as follows:

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

name: ***argocd-name***

namespace: ***argocd-ns***

spec:

...output omitted...

You can allow an Argo CD instance to manage resources in other namespaces than where it is deployed by adding the argocd.argoproj.io/managed-by label to the namespace. The following example allows the Argo CD instance in the argocd-ns namespace to manage the resources in the managed-ns namespace.

[user@host ~]$ **oc label namespace *managed-ns* \**

**argocd.argoproj.io/managed-by=*argocd-ns***

The following diagram shows an example of using an Argo CD instance to manage resources in other namespaces.

|  |
| --- |
|  |

In this diagram, a cluster administrator creates an Argo CD instance in the argocd-ns namespace. Thus, the Argo CD instance has permissions to manage resources only in that namespace.

The cluster administrator includes the argocd.argoproj.io/managed-by label in the managed-ns namespace. Thus, the Argo CD instance can manage resources in that namespace.

Developer users need only read permission in the managed-ns namespace for troubleshooting, according to the principle of least privilege, because they would be managing application resources in their projects by using Git.

### **Argo CD Authentication**

Argo CD can both use local users and integrate with other identity providers.

The OpenShift GitOps operator configures the default Argo CD instance to use Dex. Dex is an identity service that uses the OpenID connect protocol to connect Argo CD with the identity provider.

The default Argo CD instance in the openshift-gitops namespace has permission to manage cluster-wide resources. You can log in with write permission to this instance only by using the local admin user with the credentials in the openshift-gitops-cluster secret in the openshift-gitops namespace, or by using an OpenShift user from the cluster-admins group in OpenShift.

You can also configure the Argo CD instances to use Keycloak instead of Dex. For example, use the groups in Keycloak to determine the privileges in Argo CD. Keycloak acts as an identity broker between the Argo CD instance and OpenShift.

### Note

Configuring Keycloak as the SSO authentication provider for Argo CD is outside the scope of this course.

For more information, refer to <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/access_control_and_user_management/index#configuring-sso-for-argo-cd-using-keycloak>

#### **Dex**

The OpenShift GitOps operator configures Dex to delegate the authentication to the built-in OAuth server in OpenShift. Thus, Dex provides the users and groups that are defined in OpenShift.

You can manually configure Dex as the SSO authentication provider and use the OpenShift OAuth server by setting the .spec.sso parameter as in the following example:

...output omitted...

spec:

sso:

provider: dex

dex:

openShiftOAuth: true

...output omitted...

### **Argo CD Permissions**

By default, only the local Argo CD admin user and the OpenShift users in the cluster-admins group can log in to the default Argo CD instance with write permission. Other OpenShift users have only read permission. However, you can change the user and group level access by configuring the RBAC section in the Argo CD custom resource.

### Important

Argo CD RBAC controls the use of resources through Argo CD, not through the Kubernetes API. Users with access to the Kubernetes API can modify those resources and modify Argo CD RBAC. You must set up Kubernetes role-based access control to prevent access to Argo CD Kubernetes resources.

Argo CD comes with the following predefined roles:

* role:readonly for read access to all the resources.
* role:admin for read and write access to all the resources.

For example, the default Argo CD instance includes the following RBAC configuration:

...output omitted...

spec:

...output omitted...

rbac:

defaultPolicy: ""

policy: |

g, system:cluster-admins, role:admin

g, cluster-admins, role:admin

scopes: '[groups]'

...output omitted...

Thus, for OpenShift users in the system:cluster-admins and cluster-admins groups, the predefined role:admin role is assigned by default.

### Note

System groups, such as the system:cluster-admins group, are predefined groups that are built into the system to grant specific permissions to certain categories of users. Kubernetes creates these groups during the setup to control access to critical components and resources within the cluster. These groups are a part of cluster roles and cluster role bindings.

For more information, refer to <https://kubernetes.io/docs/reference/access-authn-authz/rbac/#referring-to-subjects>

To configure the Argo CD instance to assign the predefined role:readonly role to the users in the cluster-readers group, you can use the following RBAC configuration:

...output omitted...

spec:

...output omitted...

rbac:

defaultPolicy: ""

policy: |

g, system:cluster-admins, role:admin

g, cluster-admins, role:admin

**g, cluster-readers, role:readonly**

scopes: '[groups]'

...output omitted...

You can use the defaultPolicy field to define a policy for other users that do not conform to an existing policy.

#### **Fine-grained RBAC**

If you want a finer-grained RBAC for your Argo CD instance, then you can break down the permissions for all the resources in Argo CD. The resources in Argo CD are accounts, applications, applicationsets, certificates, clusters, exec, extensions, gpgkeys, logs, projects, and repositories.

For all the resources, the Argo CD actions include create, delete, get, and update. For the applications resource, Argo CD also includes the action/*group*/*kind*/*action-name*, override, and sync actions.

You can define specific fine-grained RBAC permissions with the following syntax in the spec.rbac.policy parameter:

p, ***role/user/group***, ***resource***, ***action***, ***target***

The *target* field differs between the application resources and the other resources:

* For the applications, applicationsets, logs, and exec resources, which belong to a project, the target field is *project*/*object*.
* For other resources, the target field is *object*.

As an example, the following excerpt creates the project-devs and project-admin roles, and assigns them to the users in the OpenShift groups with the same names.

...output omitted...

spec:

...output omitted...

rbac:

defaultPolicy: ''

policy: |

g, project-devs, role:project-devs

p, role:project-devs, applications, get, \*/\*, allow

p, role:project-devs, projects, get, \*, allow

p, role:project-devs, clusters, get, \*, allow

g, project-admins, role:project-admins

p, role:project-admins, applications, \*, \*/\*, allow

p, role:project-admins, projects, get, \*, allow

p, role:project-admins, clusters, get, \*, allow

scopes: '[groups]'

...output omitted...

Users in both the project-devs and project-admins groups can read the clusters and projects resources in Argo CD.

Users in the project-devs group can also read all the applications resources in any of the projects.

Users in the project-admins group can perform any action for the applications resources in any of the projects.

### Note

Argo CD includes other RBAC features that are outside the scope of this course.

For more information, refer to <https://argo-cd.readthedocs.io/en/stable/operator-manual/rbac/>

### **Monorepo and Polyrepo Environments**

If you give a user access to a Git repository, then the user has access to all the files in the repository. You cannot allow users to read or write only in certain paths within a repository. However, some products that build on top of Git, such as GitHub or GitLab, add features to improve protecting specific paths within a repository.

When setting up Git workflows for GitOps, you can either create only one repository for all your files, which is called a monorepo environment, or create repositories for different concerns, which is called a polyrepo environment.

In a monorepo environment, all the application sources, container files, Kubernetes manifests, and policy manifests are in a single Git repository.

A monorepo environment offers a centralized location for all the configuration changes. Thus, tracking, testing, and releasing processes are limited to a single Git repository. Moreover, having all your code in a single repository simplifies managing dependencies between your applications. However, users with access to the Git repository have access to all the code in the projects, which can lead to security issues.

From an Argo CD perspective, you can either treat the entire monorepo as a single application, to deploy changes across various services or components concurrently, or use paths to create several applications from a single repository. With the second option, you can use different paths for the application files and for the application configurations, such as network policies or quotas. This separation prevents application configuration changes from triggering unnecessary application rebuilds.

Another option when setting up Git workflows for GitOps is to separate the concerns by repository. A polyrepo environment enables different designs depending on your company needs.

For example, a polyrepo environment has several Git repositories for applications and configurations. Red Hat recommends using different repositories for application files and for application configurations, to prevent unnecessary application rebuilds. Independent lifecycles for application code and application configuration, along with specific approval processes, ensure smoother continuous integration. However, coordinating dependencies across repositories requires extra effort.

From an Argo CD perspective, you can create applications from the various repositories in a polyrepo environment. However, a polyrepo environment requires more complex configuration than a monorepo environment to manage the dependencies between repositories.

#### **Organizing Applications by Environment**

One GitOps practice separates the test and production environments by directory inside a Git repository, instead of creating different branches for them. This approach avoids the complexity of managing branches for various environments, where merging changes might not be straightforward because of configuration differences.

The goal is to avoid resource duplication across environments. Given that organizations often maintain many environments from development to production, replicating the entire set of deployment assets across these environments can lead to inconsistencies.

Tools such as Kustomize or Helm create reusable and modular configurations, to separate base manifests and environment-specific adjustments. With Kustomize declarative and scalable configuration management, you can create a set of base resource files to deploy the application, and then create a set of overlay files to adapt the application for the environment. With Helm, you can store in Git a Helm chart that contains all the necessary resources to deploy your application, and then use different Git repositories to store the application customization for each environment. Then, Argo CD uses the Helm chart and you can specify in Argo CD the values file to use for each environment.

### Note

Kustomize is outside the scope of this course.

Using Kustomize in an OpenShift cluster is explained in the DO280: Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster course.

For more information, refer to <https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/>

### **Managing Sensitive Data in GitOps**

To prevent security issues, you must ensure encryption or other protection of sensitive data or secrets. Sensitive data and secrets include passwords, APIs, encryption keys, SSH keys, tokens, or other digital credentials that allow users and applications access to other sensitive data, systems, and services.

Because Git users with access to the repository have access to all the files in the repository, they have access to any sensitive data that is stored in the repository. Thus, storing your sensitive data in Git repositories can introduce security concerns.

You can manually store your sensitive data in secret objects, outside a Git repository. However, if you do not correctly configure RBAC rules on your cluster, then anyone with API or etcd access can retrieve or modify a secret. Moreover, a user with permission to create a pod in a namespace can read any secret in that namespace. By default, cluster administrators can see the secrets of all users.

A mitigation is to use third-party secret management tools. Red Hat recommends using these secret management tools to mitigate security issues with features such as credential encryption, credential rotation, and short-lived tokens. Both commercial and open source solutions exist for this purpose.

For example, you can use the Secrets Store CSI Driver operator to manage multiple secrets, keys, and certificates. This driver enables Kubernetes pods to mount secrets that are stored in external secret management systems such as Azure Key Vault, AWS Secrets Manager, or HashiCorp Vault as volumes. This driver enhances security by centralizing secret management outside the cluster and allowing applications to access these secrets seamlessly through the Kubernetes API. The operator simplifies integrating and managing external secrets within Kubernetes deployments, to handle sensitive information more securely and efficiently.

### **Application Validation and Rollback**

When you deploy an application to production, even with thorough testing you cannot predict or prevent all potential issues. Also, deploying an application requires actions before, during, and after application synchronization, and if that process fails.

For any production issues, you can roll back to known stable states the small increments of your application that CI/CD practices release. Increasing the release size also results in larger rollbacks, and implies more risks.

Argo CD provides some of these features for deploying your application.

#### **Argo CD Resource Hooks**

Argo CD provides resource hooks, which are customizable triggers in deployment lifecycles. With Argo CD resource hooks, you can define and execute actions during the lifecycle of an application synchronization process.

Argo CD can run resource hooks before, during, and after an application synchronization process, and also if the synchronization process fails. For example, you can run a resource hook before the synchronization process to migrate a database before deploying a new application version, or you can run a resource hook after the synchronization process for a smoke test of your application by verifying the application health.

Argo CD defines the following resource hooks:

**PreSync**

Argo CD executes these hooks before applying the manifests.

**Sync**

Argo CD executes these hooks only after it successfully completes all the PreSync hooks. These hooks are applied at the same time as applying the manifests.

**Skip**

This hook indicates to Argo CD to skip applying the manifests. Use this hook for resources that Argo CD must create, but that other resources modify and that must be out of Argo CD synchronization.

**PostSync**

Argo CD executes these hooks only after it successfully completes all the Sync hooks, and the application and its resources are in a healthy state.

**SyncFail**

Argo CD executes these hooks whenever the synchronization operation fails.

Argo CD resource hooks are Kubernetes resources that include the argocd.argoproj.io/hook annotation. Typically, Argo CD resource hooks are implemented through Kubernetes jobs.

The following example shows the annotation for a PreSync hook that migrates the database before deploying the new application version:

apiVersion: batch/v1

kind: Job

metadata:

generateName: schema-migrate-database

annotations:

argocd.argoproj.io/hook: PreSync

...output omitted...

You can add the argocd.argoproj.io/hook-delete-policy annotation for your hooks, so Argo CD deletes them automatically according to one of the following policies:

**HookSucceeded**

Delete the hook resource after the hook succeeds.

**HookFailed**

Delete the hook resource after the hook fails.

**BeforeHookCreation**

Delete any existing hook resource before a new one is created.

The following excerpt shows the annotation for a PostSync hook that tests the application after the synchronization process and removes the hook if it succeeds:

apiVersion: batch/v1

kind: Job

metadata:

generateName: integration-test-app

annotations:

argocd.argoproj.io/hook: PostSync

argocd.argoproj.io/hook-delete-policy: HookSucceeded

...output omitted...

#### **Argo CD Rollback**

Argo CD enables you to roll back your application to a known stable state if the new version has errors.

You can manually roll back your application by clicking the **HISTORY AND ROLLBACK** button. You can then access previous application deployments and restore the application to an earlier version. For example, with this feature you can restore the application to a working version until you push a fix in your application repository.

Argo CD can also automate rollbacks in certain conditions, such as for failing resource hooks.

### Note

The Argo CD rollback feature is disabled if you enable automatic synchronization for your application, because Argo CD automatically synchronizes the application to the latest Git commit version.

To roll back your application, you can temporarily disable automatic synchronization in Argo CD, or revert the last commit in your Git repository.

#### **Argo Rollouts**

Argo Rollouts is a Kubernetes controller and a set of custom resource definitions that provide advanced deployment capabilities. These capabilities include update strategies, such as progressive rollouts, and automated rollbacks and promotions.

You can integrate Argo Rollouts with ingress controllers and service meshes, to gradually shift the traffic from your previous application version to a new one.

For example, you can use Argo Rollouts to deploy a new application version for only a small percentage of the production traffic, and then analyze if everything works as expected. Argo Rollouts can automatically roll back to the previous stable version if the deployment fails.

### Note

Argo Rollouts is a Technology Preview feature only, and is outside the scope of this course.

### References

For more information about configuring SSO on Argo CD by using Dex or Keycloak, refer to the Red Hat OpenShift GitOps 1.10 Access Control and User Management documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/access_control_and_user_management/index>

For more information about Argo Rollouts, refer to the Red Hat OpenShift GitOps 1.10 Argo Rollouts documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_gitops/1.10/html-single/argo_rollouts/index>

For more information about best practices for structuring Git workflows, refer to the The Path to GitOps book at <https://developers.redhat.com/e-books/path-gitops>

For more information about implementing GitOps with Argo CD in OpenShift, refer to the Getting GitOps: A Practical Platform with OpenShift, Argo CD, and Tekton book at <https://developers.redhat.com/e-books/getting-gitops-practical-platform-openshift-argo-cd-and-tekton>

## **Guided Exercise: GitOps for Application Management**

Deploy an application and its dependencies by using OpenShift GitOps.

**Outcomes**

* Create an Argo CD instance for developers with the appropriate permissions.
* Use Argo CD to deploy an application in the cluster.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start gitops-app**

**Instructions**

Your company requires you to create a separate instance of Argo CD in the gitops-app namespace, for a project administration team and a project developer team that must deploy an application.

For the Argo CD instance, the project-admins group has full permissions for Argo CD applications, but only read permission for projects and clusters. For the same instance, the project-devs group has only read permission for Argo CD applications, projects, and clusters. The project-admin user is part of the project-admins group, and the developer user is part of the project-devs group.

The following diagram summarizes the relationship between the components in the exercise:

|  |
| --- |
|  |

In this exercise, GitLab is configured with two repositories, one for project administration and one for project developer teams. The repository for the developer user is populated with the necessary files, including a MariaDB database and the Etherpad application. The project-admin user has one empty etherpad-admin repository.

Create the etherpad-devs namespace by using the OpenShift admin user. You must create the application in the etherpad-devs namespace. The Argo CD instance must manage the etherpad-devs namespace. The OpenShift admin user gives administrative access to the etherpad-devs namespace for users in the project-admins group.

The project-admin admin user creates a secret with the credentials for the MariaDB database to be deployed. This setup ensures that the database credentials are not stored in Git; storing in Git could lead to security issues.

At the beginning, the developer user has no permissions on the etherpad-devs namespace. Thus, the project-admin user must upload an RBAC file to the etherpad-admin repository and use Argo CD to give the developer user read permission in the project. Then, the developer user can view logs for troubleshooting if necessary.

Next, the project-admin user creates the MariaDB and Etherpad applications. The project-admin user also creates a PVC for the database backup.

The developer user creates two Argo CD hooks: one pre-synchronization hook that makes a database backup in a separate PVC, and one post-synchronization hook that verifies that the Etherpad application is running.

Finally, the developer user modifies the title for the Etherpad application in the repository and sees how the application is updated in Argo CD.

1. Create the Argo CD instance in the gitops-app namespace. The Argo CD must trust the classroom certificate.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

Login successful.

You have access to 70 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "default".

* 1. Change to the gitops-app project.

[student@workstation ~]$ **oc project gitops-app**

Now using project "gitops-app" on server "https://api.ocp4.example.com:6443".

* 1. Create a cluster-root-ca-bundle configuration map in the gitops-app namespace.

[student@workstation ~]$ **oc create configmap cluster-root-ca-bundle**

configmap/cluster-root-ca-bundle created

* 1. Add the config.openshift.io/inject-trusted-cabundle label to the configuration map with the true value. OpenShift injects the bundle with the cluster certificate authority to the configuration maps with this label. This bundle contains the signing certificate for the classroom GitLab instance.

[student@workstation ~]$ **oc label configmap cluster-root-ca-bundle \**

**config.openshift.io/inject-trusted-cabundle=true**

configmap/cluster-root-ca-bundle labeled

* 1. Create the Argo CD instance YAML file. Set the termination type of the route to the reencrypt type. Grant read permission to projects and clusters to the project-admins and project-devs groups. For applications, grant full permissions to the project-admins group, and read permission to the project-devs groups. The Argo CD instance must mount the ca-bundle.crt certificate in the configuration map to the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path of the repository server container. You can find an incomplete example for the Argo CD CR in the ~/DO380/labs/gitops-app/argocd-instance.yaml file.

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

name: argocd

namespace: gitops-app

spec:

server:

...output omitted...

route:

enabled: true

**tls:**

**termination: reencrypt**

...output omitted...

rbac:

defaultPolicy: ''

policy: |

g, system:cluster-admins, role:admin

**g, project-devs, role:project-devs**

**p, role:project-devs, applications, get, \*/\*, allow**

**p, role:project-devs, projects, get, \*, allow**

**p, role:project-devs, clusters, get, \*, allow**

**g, project-admins, role:project-admins**

**p, role:project-admins, applications, \*, \*/\*, allow**

**p, role:project-admins, projects, get, \*, allow**

**p, role:project-admins, clusters, get, \*, allow**

scopes: '[groups]'

repo:

resources:

limits:

cpu: 1000m

memory: 1024Mi

requests:

cpu: 250m

memory: 256Mi

**volumeMounts:**

**- mountPath: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**

**name: cluster-root-ca-bundle**

**subPath: ca-bundle.crt**

**volumes:**

**- configMap:**

**name: cluster-root-ca-bundle**

**name: cluster-root-ca-bundle**

...output omitted...

* 1. Create the Argo CD instance.

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

**~/DO380/labs/gitops-app/argocd-instance.yaml**

argocd.argoproj.io/argocd created

1. Create the etherpad-devs project and label it as managed by the Argo CD instance. Assign administrative permission to users in the project-admins group.
   1. Create the etherpad-devs project.

[student@workstation ~]$ **oc new-project etherpad-devs**

Now using project "etherpad-devs" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Label the etherpad-devs project as managed by the Argo CD instance in the gitops-app project.

[student@workstation ~]$ **oc label namespace etherpad-devs \**

**argocd.argoproj.io/managed-by=gitops-app**

namespace/etherpad-devs labeled

* 1. Assign administrative permission to the etherpad-devs project for users in the project-admins group.

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

**project-admins -n etherpad-devs**

clusterrole.rbac.authorization.k8s.io/admin added: "project-admins"

* 1. Log in as the developer user.

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

...output omitted...

* 1. Verify that the developer user has no access to the etherpad-devs project.

[student@workstation ~]$ **oc projects**

You are not a member of any projects. You can request a project to be created with the 'new-project' command.

* 1. Log in as the project-admin user.

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

...output omitted...

1. As the project-admin user in OpenShift, create the mariadb secret with the MariaDB database credentials. Create the database-backup persistent volume claim (PVC), for a later step to back up the MariaDB database.
   1. Create the mariadb secret YAML file that contains the MariaDB credentials. You can find an incomplete example for the secret in the ~/DO380/labs/gitops-app/secret.yaml file.

apiVersion: v1

kind: Secret

metadata:

name: **mariadb**

namespace: **etherpad-devs**

labels:

app: **mariadb**

stringData:

MARIADB\_DATABASE: etherpad\_lite\_db

MARIADB\_USER: appuser

MARIADB\_PASSWORD: securepassword

MARIADB\_ROOT\_PASSWORD: supersecurepassword

* 1. Create the mariadb secret.

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

**~/DO380/labs/gitops-app/secret.yaml**

secret/mariadb created

1. As the project-admin user, create an RBAC file in the etherpad-admin repository in GitLab to give the developer user read permission in the etherpad-devs project. The project-admin user uses this repository in Argo CD in a later step to give read permission to the developer user.
   1. Create the RBAC YAML file to give the developer user read permission in the etherpad-devs project. You can find an incomplete YAML example in the ~/DO380/labs/gitops-app/etherpad-admin/rbac.yaml file.

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: developer-view

namespace: **etherpad-devs**

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: view

subjects:

- kind: Group

apiGroup: rbac.authorization.k8s.io

name: **project-devs**

* 1. Open a web browser and navigate to https://git.ocp4.example.com. Log in as the project-admin user with r3dh4tgit as the password.
  2. Click the etherpad-admin project.
  3. Click the **Upload File** button and then click upload. Select the ~/DO380/labs/gitops-app/etherpad-admin/rbac.yaml file, and then click the **Open** button. Click **Upload file**.

1. Open the Argo CD instance as the project-admin user and create the RBAC rule for the developer user.
   1. Change to the terminal window and log in as the admin user.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp**

...output omitted...

* 1. Verify the route for the Argo CD instance.

[student@workstation ~]$ **oc get route -n gitops-app**

* 1. NAME HOST/PORT PATH ...

argocd-server **argocd-server-gitops-app.apps.ocp4.example.com** ...

* 1. Open a web browser tab and navigate to https://argocd-server-gitops-app.apps.ocp4.example.com.
  2. Click **LOG IN VIA OPENSHIFT** and log in as the project-admin user with redhat as the password, by using the **Red Hat Identity Management** identity provider, and allowing the **user:info** permission.
  3. Click **CREATE APPLICATION**.
  4. Create an Argo CD application with the information in the following table. Then, click **CREATE**.

| **Field** | **Value** |
| --- | --- |
| Application Name | rbac-rule |
| Project Name | default |
| Sync Policy | Automatic |
| Repository URL | https://git.ocp4.example.com/project-admin/etherpad-admin.git |
| Path | . |
| Cluster URL | https://kubernetes.default.svc |
| Namespace | etherpad-devs |

* 1. Change to the terminal window and log in as the developer user. The developer user has access to the etherpad-devs project.
  2. [student@workstation ~]$ **oc login -u developer -p developer**
  3. ...output omitted...

Using project "etherpad-devs".

1. Create the Etherpad application by using the Argo CD instance and verify that it works. The project-admin user creates the application by using the developer user repository. Thus, although the developer user cannot modify the application in Argo CD or in OpenShift, the developer user can change the files in the repository to update the application.
   1. Go back to the Argo CD browser tab.
   2. Click **NEW APP**.
   3. Create an application with the information in the following table:

| **Field** | **Value** |
| --- | --- |
| Application Name | etherpad-app |
| Project Name | default |
| Sync Policy | Automatic |
| Repository URL | https://git.ocp4.example.com/developer/etherpad-app.git |
| Path | . |
| Cluster URL | https://kubernetes.default.svc |
| Namespace | etherpad-devs |

* 1. Then, click **CREATE**.
  2. Click **etherpad-app** to view the application. Argo CD starts synchronizing the application. After about one minute, the console shows the Etherpad application as synchronized and healthy.
  3. Change to the terminal window and get the URL for the Etherpad application. The developer user has read access to the etherpad-devs project in OpenShift.
  4. [student@workstation ~]$ **oc get route**
  5. NAME HOST/PORT PATH ...

etherpad **etherpad-etherpad-devs.apps.ocp4.example.com** ...

* 1. Open a web browser tab and navigate to https://etherpad-etherpad-devs.apps.ocp4.example.com to verify that the application is up and running. Notice the DO380 - etherpad title in the tab.

1. As the developer user in Git, create two Argo CD hooks in the etherpad-app Git repository. The developer user Git credentials are already configured in the classroom environment. The pre-synchronization hook must back up the database in the database-backup PVC. The post-synchronization hook verifies that the Etherpad application is running. Modify the Etherpad application title.
   1. Change to the terminal window, and change to the ~/DO380/labs/gitops-app/etherpad-app directory. The Git repository is synchronized to that directory.

[student@workstation ~]$ **cd ~/DO380/labs/gitops-app/etherpad-app**

* 1. Copy the provided hook files to the repository.

[student@workstation etherpad-app]$ **cp ../hooks/{presync,postsync}.yaml .**

* 1. Edit the pre-synchronization hook file to match the following text. The pre-synchronization hook uses the mysqldump command to back up the database in the database-backup PVC.

apiVersion: batch/v1

kind: Job

metadata:

generateName: backup-mariadb

annotations:

argocd.argoproj.io/hook: **PreSync**

...output omitted...

* 1. Edit the post-synchronization hook file to match the following text. The post-synchronization hook uses the curl command to ensure that the Etherpad application is up and running.

apiVersion: batch/v1

kind: Job

metadata:

generateName: test-etherpad

annotations:

argocd.argoproj.io/hook: **PostSync**

spec:

template:

metadata:

name: test-etherpad

spec:

containers:

- name: test-etherpad

image: registry.ocp4.example.com:8443/redhattraining/mariadb:10.5

command: ["curl","-k","-s","https://etherpad-etherpad-devs.apps.ocp4.example.com"]

restartPolicy: Never

backoffLimit: 2

|  |  |
| --- | --- |
|  | This command should be written in a single line. |

* 1. Open the etherpad-deployment.yaml file and modify the Etherpad application title.

apiVersion: apps/v1

kind: Deployment

metadata:

name: etherpad

labels:

app.kubernetes.io/name: etherpad

spec:

...output omitted...

spec:

securityContext:

runAsNonRoot: true

seccompProfile:

type: RuntimeDefault

containers:

- env:

- name: TITLE

value: **My test Etherpad app**

...output omitted...

* 1. Add the hook and deployment files to the Git index.

[student@workstation etherpad-app]$ **git add presync.yaml postsync.yaml \**

**etherpad-deployment.yaml**

* 1. Commit the changes.

[student@workstation etherpad-app]$ **git commit -m \**

**"Add synchronization hooks and change title"**

...output omitted...

* 1. Push the changes to the repository.

[student@workstation etherpad-app]$ **git push**

...output omitted...

1. As the developer user in Argo CD, verify that the application is updated to the last repository version with the two hooks.
   1. Change to the Argo CD browser tab and click **Log out**.
   2. Click **LOG IN VIA OPENSHIFT** and log in as the developer user with developer as the password, by using the **Red Hat Identity Management** identity provider, and allowing the **user:info** permission.
   3. Click **etherpad-app** to view the application.
   4. Argo CD automatically detects the changes in the repository and starts synchronizing the application.

### Note

The Argo CD automatic synchronization interval is set to 3 minutes by default. Thus, if the last synchronization does not match the last commit, then you can click **REFRESH** so Argo CD detects the changes.

If the Argo CD application is in the Syncing stage, then wait until Argo CD finishes synchronizing it.

* 1. Verify that Argo CD creates the backup-mariadb and test-etherpad jobs according to the hooks. To inspect the logs for the hooks, click the job name and then click **LOGS**. The backup-mariadb pre-synchronization hook connects to and backs up the MariaDB database in the database-backup PVC. The test-etherpad post-synchronization hook uses the curl command to verify that the Etherpad application is up and running.
  2. Change to the terminal window and log in as the project-admin user.

[student@workstation etherpad-app]$ **oc login -u project-admin -p redhat**

...output omitted...

* 1. Create a pod to inspect the contents of the database-backup PVC. You can use the ~/DO380/labs/gitops-app/backup-inspector.yaml file for this purpose.

[student@workstation etherpad-app]$ **oc create -f ../backup-inspector.yaml**

pod/backup-inspector created

* 1. Wait until the backup-inspector pod is running and open a remote shell on it.

[student@workstation etherpad-app]$ **oc rsh backup-inspector**

* 1. Review the contents of the external-data directory. The pre-synchronization hook creates a database backup every time that the application is updated to a new version.
  2. sh-5.1$ **ls /external-data/**

database-backup-files-202312201021 lost+found

* 1. Close the remote shell.

sh-5.1$ **exit**

* 1. Remove the backup-inspector pod.
  2. [student@workstation etherpad-app]$ **oc delete pod backup-inspector**

pod "backup-inspector" deleted

* 1. Change to the Etherpad web browser tab and reload it. The tab title must change to My test Etherpad app.

1. Close the web browser and change to the student HOME directory in the terminal window.

[student@workstation etherpad-app]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish gitops-app**

## **Lab: OpenShift GitOps**

Define the fundamentals of GitOps and its use with Kubernetes clusters and applications.

Deploy a GitOps instance for cluster administration.

Deploy another GitOps instance for application developers and administrators.

**Outcomes**

* Create an Argo CD instance for developers with the appropriate permissions.
* Use GitOps practices to create a cron job.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

Some teams in your organization execute periodic tasks manually. Your organization wants to evaluate automating the tasks with OpenShift and GitOps practices.

In this exercise, you install the Red Hat OpenShift GitOps operator for this purpose. You create a custom Argo CD instance, a Git repository with a sample periodic task, and an Argo CD application.

Developers can create cron jobs only by pushing them to the Git repository. Developers must be able to troubleshoot the cron jobs.

1. Install the OpenShift GitOps operator from OperatorHub. Use the admin user with redhatocp as the password.
   1. Use the command line to log in to the OpenShift cluster as the admin user with redhatocp as the password.

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

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

...output omitted...

* 1. Identify the URL for the OpenShift web console.

[student@workstation ~]$ **oc whoami --show-console**

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

* 1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com. Either type the URL in a web browser, or right-click and select **Open Link** from the command line.
  2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.
  3. Navigate to **Operators** → **OperatorHub**.
  4. Click **Red Hat OpenShift GitOps**, and then click **Install**.
  5. Review the default configuration and click **Install**. The Operator Lifecycle Manager can take a few minutes to install the operator. Click **View Operator** to navigate to the operator details.

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

1. Create the Argo CD instance in the gitops-review namespace. The Argo CD must trust the classroom certificate. Grant read permission to applications, projects, and clusters to the users in the project-devs group.

You can find an incomplete example for the Argo CD CR in the ~/DO380/labs/gitops-review/argocd-instance.yaml file.

* 1. Change to the command line, and change to the gitops-review project.

[student@workstation ~]$ **oc project gitops-review**

Now using project "gitops-review" on server "https://api.ocp4.example.com:6443".

* 1. Create a cluster-root-ca-bundle configuration map in the gitops-review namespace.

[student@workstation ~]$ **oc create configmap cluster-root-ca-bundle**

configmap/cluster-root-ca-bundle created

* 1. Add the config.openshift.io/inject-trusted-cabundle label to the configuration map with the true value. OpenShift injects the bundle with the cluster certificate authority to the configuration maps with this label. This bundle contains the signing certificate for the classroom GitLab instance.

[student@workstation ~]$ **oc label configmap cluster-root-ca-bundle \**

**config.openshift.io/inject-trusted-cabundle=true**

configmap/cluster-root-ca-bundle labeled

* 1. Create the Argo CD instance YAML file. Set the termination type of the route to the reencrypt type. Grant read permission to applications, projects, and clusters to the project-devs groups. The Argo CD instance must mount the ca-bundle.crt certificate in the configuration map to the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path of the repository server container. You can find an incomplete example for the Argo CD CR in the ~/DO380/labs/gitops-review/argocd-instance.yaml file.

apiVersion: argoproj.io/v1beta1

kind: ArgoCD

metadata:

name: argocd

namespace: gitops-review

spec:

server:

...output omitted...

route:

enabled: true

**tls:**

**termination: reencrypt**

...output omitted...

rbac:

defaultPolicy: ''

policy: |

g, system:cluster-admins, role:admin

**g, project-devs, role:project-devs**

**p, role:project-devs, applications, get, \*/\*, allow**

**p, role:project-devs, projects, get, \*, allow**

**p, role:project-devs, clusters, get, \*, allow**

scopes: '[groups]'

repo:

resources:

limits:

cpu: 1000m

memory: 1024Mi

requests:

cpu: 250m

memory: 256Mi

**volumeMounts:**

**- mountPath: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**

**name: cluster-root-ca-bundle**

**subPath: ca-bundle.crt**

**volumes:**

**- configMap:**

**name: cluster-root-ca-bundle**

**name: cluster-root-ca-bundle**

...output omitted...

* 1. Create the Argo CD instance.

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

**~/DO380/labs/gitops-review/argocd-instance.yaml**

argocd.argoproj.io/argocd created

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

1. Create the project-devs group in OpenShift, and add the developer user to it. Add read access to the gitops-review project to the users in the project-devs group.
   1. Create the project-devs group.

[student@workstation ~]$ **oc adm groups new project-devs**

* 1. Add the developer user to the project-devs group.

[student@workstation ~]$ **oc adm groups add-users project-devs developer**

* 1. Add read access to the gitops-review project to users in the project-devs group.
  2. [student@workstation ~]$ **oc adm policy add-role-to-group view project-devs \**

**-n gitops-review**

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Log in to the Argo CD web console as the admin user by using their OpenShift credentials. Create an Argo CD application by using the cron-job repository from the developer user. Use the information in the following table:

| **Field** | **Value** |
| --- | --- |
| Application Name | cron-job |
| Project Name | default |
| SYNC POLICY | Automatic |
| Retry | Checked |
| Repository URL | https://git.ocp4.example.com/developer/cron-job.git |
| Path | . |
| Cluster URL | https://kubernetes.default.svc |

* 1. Verify the route for the Argo CD instance.
  2. [student@workstation ~]$ **oc get route -n gitops-review**
  3. NAME HOST/PORT PATH ...

argocd-server **argocd-server-gitops-review.apps.ocp4.example.com** ...

* 1. Open a web browser tab and navigate to https://argocd-server-gitops-review.apps.ocp4.example.com
  2. Click **LOG IN VIA OPENSHIFT** and log in as the admin user with redhatocp as the password, by using the **Red Hat Identity Management** identity provider, and allowing the **user:info** permission.
  3. Click **CREATE APPLICATION**.
  4. Fill the application information with the details in the table. Then, click **CREATE**.
  5. Wait until Argo CD shows the cron-job application as synchronized and healthy. Then, click **Log out**.

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

1. As the developer user, create a periodic-process cron job in the Git repository. The URL for the Git instance is https://git.ocp4.example.com. The password for the developer user in Git is d3v3lop3r. The developer user Git credentials are already configured in the classroom environment.

The cron job runs the echo hello command every minute with the registry.ocp4.example.com:8443/ubi9/ubi image.

You can use the following command to generate a template for the cron job:

[student@workstation ~]$ **oc create cronjob periodic-process \**

**-n gitops-review --schedule "\* \* \* \* \*" \**

**--image registry.ocp4.example.com:8443/ubi9/ubi \**

**--dry-run=client -o yaml -- echo hello**

* 1. In the web browser, open a tab navigate to the Git URL at https://git.ocp4.example.com. Log as the developer user with d3v3lop3r as the password. Click the cron-job project. Click **Clone**, and then copy the https://git.ocp4.example.com/admin/gitops-review.git HTTPS URL.
  2. Change to the command line. Change to the ~/DO380/labs/gitops-review directory.

[student@workstation ~]$ **cd ~/DO380/labs/gitops-review**

* 1. Run the following command to clone the cron-job repository.

[student@workstation gitops-review]$ **git clone \**

**https://git.ocp4.example.com/developer/cron-job.git**

Cloning into 'cron-job'...

...output omitted...

* 1. Change to the cloned repository directory.

[student@workstation gitops-review]$ **cd cron-job**

The default configuration for new repositories adds a README.md initial file.

* 1. Create a periodic-process.yaml file with the following content:

apiVersion: batch/v1

kind: CronJob

metadata:

name: periodic-process

namespace: gitops-review

spec:

jobTemplate:

metadata:

name: periodic-process

spec:

template:

spec:

containers:

- command:

- echo

- hello

image: registry.ocp4.example.com:8443/ubi9/ubi

name: periodic-process

restartPolicy: OnFailure

schedule: '\* \* \* \* \*'

You can create the file with the output of the previous command, and removing the parts that are not present in the previous content.

* 1. Add the periodic-process.yaml file to the repository.

[student@workstation cron-job]$ **git add periodic-process.yaml**

* 1. Commit the changes.
  2. [student@workstation cron-job]$ **git commit -m "add job"**

...output omitted...

* 1. Push the changes.
  2. [student@workstation cron-job]$ **git push**

...output omitted...

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

1. As the developer user in Argo CD, verify that the application is updated to the last repository version with the cron job. The password for the developer user in OpenShift is developer. Verify that the user can review the application logs in Argo CD.
   1. Change to the Argo CD browser tab.
   2. Click **LOG IN VIA OPENSHIFT** and log in as the developer user with developer as the password, by using the **Red Hat Identity Management** identity provider, and allowing the **user:info** permission.
   3. Click **cron-job** to view the application.
   4. Argo CD automatically detects the changes in the repository and starts synchronizing the application.

### Note

The Argo CD automatic synchronization interval is set to 3 minutes by default. Thus, if the last synchronization does not match the last commit, then you can click **REFRESH** so Argo CD detects the changes.

If the Argo CD application is in the Syncing stage, then wait until Argo CD finishes synchronizing it.

* 1. Verify that Argo CD creates the periodic-process cron job. The cron job runs the echo hello command every minute. Wait until the cron job creates a pod at least one time. To inspect the logs for the jobs, click the job name and then click **LOGS**. The logs show the word hello as the output.

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

1. As the developer user in OpenShift, verify that the developer user can retrieve the logs for the cron job pods.
   1. Change to the command line and log in as the developer user.

[student@workstation cron-job]$ **oc login -u developer -p developer**

...output omitted...

Using project "gitops-review".

* 1. List the pods for the gitops-review project.

[student@workstation cron-job]$ **oc get pods**

NAME READY STATUS RESTARTS AGE

argocd-application-controller-0 1/1 Running 0 111m

argocd-dex-server-85d586c89b-ndxtj 1/1 Running 0 111m

argocd-redis-7545d85b5-mt98r 1/1 Running 0 111m

argocd-repo-server-5c8bc5758d-9mtv9 1/1 Running 0 111m

argocd-server-68cbc7589c-md52w 1/1 Running 0 111m

periodic-process-28597732-wcrxq 0/1 Completed 0 2m41s

periodic-process-28597733-z99pk 0/1 Completed 0 101s

periodic-process-28597734-p9wc4 0/1 Completed 0 41s

...output omitted...

* 1. Check the logs for one of the periodic-process pods.

[student@workstation cron-job]$ **oc logs *periodic-process-28597734-p9wc4***

hello

* 1. Close the web browser and change to the /home/student directory in the command line.

[student@workstation cron-job]$ **cd**

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

**Evaluation**

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

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

# **Chapter 6.  OpenShift Monitoring**

[Cluster Monitoring](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06)

[Guided Exercise: Cluster Monitoring](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06s02)

[Alerts and Notifications](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06s03)

[Guided Exercise: Alerts and Notifications](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06s04)

[Lab: OpenShift Monitoring](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06s05)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch06s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Troubleshoot performance and availability issues with applications and clusters. |
| **Sections** | * Cluster Monitoring (and Guided Exercise) * Alerts and Notifications (and Guided Exercise) |
| **Lab** | * OpenShift Monitoring |

## **Cluster Monitoring**

### **Objectives**

* Describe the architecture of OpenShift Monitoring and query the information in its dashboards.

### **OpenShift Observability**

In the cloud computing era, systems are becoming ever more complex. All organizations aim for reliable, efficient, and secure applications and infrastructure. Observability plays a key role in achieving this goal. Observability is the ability to understand a system's or an application's state by collecting and analyzing its output and logs.

Red Hat OpenShift Observability collects logs, traces, events, and system metrics to provide real-time monitoring. Real-time monitoring helps to identify and troubleshoot issues for OpenShift applications and clusters.

Some key features of the Red Hat OpenShift Observability portfolio are as follows:

**OpenShift Logging**

Red Hat OpenShift Logging aggregates all the logs from the pods and nodes of an OpenShift cluster to a centralized location. Centralized logging improves searching, visualizing, and reporting of data.

**OpenShift Monitoring**

Red Hat OpenShift Monitoring provides monitoring for core platform components.

**Network Observability**

Network observability monitors and analyzes network traffic and helps to resolve connectivity issues.

**Distributed Tracing**

Distributed tracing collects observability data in distributed systems. Distributed tracing is based on the OpenTelemetry project.

For more details about network observability and distributed tracing, refer to the References section.

You can use these features with any stand-alone OpenShift cluster. Red Hat Advanced Cluster Management for Kubernetes (RHACM) provides multicluster observability features.

### **OpenShift Monitoring**

Red Hat OpenShift Container Platform comes with a monitoring stack. The cluster monitoring operator manages the monitoring components and ensures that they are always available and updated. The default monitoring stack collects metrics and generates alerts for the cluster, which includes core platform components and all projects. However, the default monitoring stack does not support custom metrics for user-defined projects.

You can enable monitoring for user-defined projects. You can collect custom metrics and generate alerts by using the monitoring stack for user-defined projects.

You can configure persistent storage for OpenShift monitoring. With this configuration, you can keep a record of the past cluster status, to investigate and correlate current and past issues within the cluster. The monitoring dashboard provides visuals for cluster metrics.

### **OpenShift Monitoring Stack**

The OpenShift monitoring stack is based on the Prometheus open source project. The stack includes the components that are shown in the following figure and are then explained in the following section:

|  |
| --- |
|  |

Figure 6.1: OpenShift monitoring architecture

#### **Components for Default Monitoring Stack**

The default monitoring stack is included with OpenShift Container Platform. All components of the default monitoring stack are installed in the openshift-monitoring project and provide monitoring features for core platform components.

Modifying any existing resource and creating additional ServiceMonitor, PodMonitor, or PrometheusRule resources in the openshift-monitoring project is not supported. The monitoring stack resets modified resources to ensure that its resources always remain in the expected state.

The monitoring stack deploys the following components in the environment for monitoring the infrastructure, receiving alerts, and consulting performance graphs.

**Cluster Monitoring Operator**

The cluster monitoring operator is the central component of the monitoring stack. The cluster monitoring operator controls the deployed monitoring components and ensures that they are always in sync with the latest version of the cluster monitoring operator.

**Prometheus Operator**

The Prometheus operator deploys and configures both Prometheus and Alertmanager. The operator also manages the generation of configuration targets (service monitors and pod monitors).

**Prometheus**

Prometheus is the monitoring server.

**Prometheus Adapter**

The Prometheus adapter exposes cluster resources for Horizontal Pod Autoscaling (HPA).

**Prometheus Alertmanager**

Alertmanager handles alerts from the Prometheus server. An alert is a rule that evaluates to true or false and is often based on cluster observations, such as cluster CPU utilisation. An alert fires when the alert rule meets the true condition. You can configure Prometheus Alertmanager to group and route the alerts to the receiver.

**Kube state metrics**

The kube-state-metrics converter agent exports Kubernetes objects to metrics that Prometheus can parse.

**OpenShift state metrics**

The openshift-state-metrics agent is based on the kube-state-metrics agent and adds monitoring for OpenShift-specific resources (such as image registry metrics).

**Node exporter**

The node-exporter agent exports low-level metrics for compute nodes.

**Thanos Querier**

Thanos Querier is a single, multitenant interface that enables aggregating and deduplicating cluster and user workload metrics.

**Telemeter Client**

Telemeter Client sends a data portion from Prometheus instances to Red Hat for remote health monitoring.

**The monitoring web console**

The OpenShift Container Platform web console provides the **Observe** section to access and manage monitoring features. In the Observe section, you can access monitoring dashboards, metrics, alerts, and metrics targets.

#### **Components for Monitoring User-defined Projects**

You can enable monitoring for user-defined projects. You can collect application-specific custom metrics and also create alerts by using custom metrics with this monitoring stack. The monitoring components for user-defined projects are installed in the openshift-user-workload-monitoring project.

The monitoring stack for user-defined projects deploys the following components:

* Prometheus Operator
* Prometheus
* Thanos Ruler
* Alertmanager

For more details about the OpenShift monitoring stack for user-defined projects, see the References section.

### **Prometheus**

Prometheus is an open source project for system monitoring and alerting.

Both Red Hat OpenShift Container Platform and Kubernetes integrate Prometheus to enable cluster metrics, monitoring, and alerting capabilities.

Prometheus gathers and stores streams of data from the cluster as time-series data. Time-series data consists of a sequence of samples, where each sample contains the following elements:

* A timestamp
* A numeric value (such as an integer, float, or Boolean)
* A set of labels in the form of key/value pairs

The key/value pairs isolate groups of related values for filtering.

For example, the machine\_cpu\_cores metric in Prometheus contains a sequence of measurement samples of the number of CPU cores for each machine.

### **OpenShift Monitoring Web Console**

You can access and manage the monitoring features by using the web console. The OpenShift Container Platform web console provides the **Observe** section, with the following subsections:

* Alerting
* Metrics
* Dashboards
* Targets

#### **Cluster Monitoring Alerting**

You access cluster alerts from the OpenShift web console at **Observe** → **Alerting**. For each alert, the **Alerting** page displays a brief description, the state, and the severity. You can view alert details by clicking the name of the alert. The **Alert details** page also displays a time-series graphic.

|  |
| --- |
|  |

Figure 6.2: Cluster monitoring alerting

For more details about forwarding alerts to other systems, see the next section.

#### **Cluster Monitoring Metrics**

OpenShift integrates Prometheus metrics at **Observe** → **Metrics**.

From the **Metrics** page, enter an expression, such as a metric name, and then click **Run Queries** to retrieve the most recent sample for the metric.

The following example displays the instance:node\_cpu\_utilisation:rate1m metric over time.

|  |
| --- |
|  |

Figure 6.3: Monitoring metrics

The metric contains data for each node instance in the cluster.

The OpenShift has three monitoring stack components to gather the metrics from the Kubernetes API: the kube-state-metrics, openshift-state-metrics, and node-exporter agents.

The dashboards in OpenShift cluster monitoring combine metrics from the three agents.

See the References section to learn about the complete list of exposed metrics.

Prometheus provides a query language, PromQL, to select and aggregate time-series data.

You can filter a metric to include only certain key/value pairs. For example, you can modify the previous query to show only metrics for the worker02 node by using the following expression:

instance:node\_cpu\_utilisation:rate1m{instance="worker02"}

Prometheus Query Language provides several operators to compute new time-series metrics. PromQL contains arithmetic operators, including addition, subtraction, multiplication, and division operators. PromQL contains comparison operators, including equality, greater-than, and less-than operators.

PromQL contains built-in functions, including the following ones, that you can include in PromQL expressions:

**sum()**

Adds the value of all sample entries at a given time.

**rate()**

Computes the per-second average of a time series for a given time range.

**count()**

Counts the number of sample entries at a given time.

**max()**

Selects the maximum value out of the sample entries.

The following examples of Prometheus Query Language expressions use one metric from the node-exporter agent, and another metric from the kube-state-metrics agent:

**node\_memory\_MemAvailable\_bytes/node\_memory\_MemTotal\_bytes\*100<50**

Shows nodes with less than 50% of available memory.

**kube\_persistentvolumeclaim\_status\_phase{phase="Pending"} == 1**

Shows persistent volume claims in the pending state.

The Red Hat add-on operators can define extra metrics and alerts. For example, the compliance operator exposes additional metrics to Prometheus. You can get a list of exposed metrics by using the following query:

{***name***=~"compliance.\*"}

#### **Cluster Monitoring Dashboards**

The OpenShift console integrates dashboards based on the gathered metrics at **Observe** → **Dashboards**. These dashboards refresh periodically to display current summary metrics and graphs.

In the graphs, which are interactive, you can further explore data features and characteristics that you observe.

The cluster monitoring dashboards serve as a good starting point for near real-time observability of cluster metrics and health.

After receiving an alert, an administrator might use the dashboards to investigate the problem. This investigation might include determining whether a specific node or project has a problem. Additionally, cluster monitoring dashboards can help identify whether a problem was temporary or appears to be persistent.

OpenShift cluster monitoring includes several default dashboards.

Some of the default monitoring default dashboards are as follows:

**Kubernetes / Compute Resources / Cluster**

This dashboard displays a high-level view of cluster resources. The **Kubernetes / Compute Resources / Cluster** dashboard page shows percentage values for CPU such as **CPU Utilisation**, **CPU Requests Commitment**, and **CPU Limits Commitment**. Similar values are also available for memory.

|  |
| --- |
|  |

Figure 6.4: Kubernetes / Compute Resources / Cluster dashboard

You can see metrics by clicking **Inspect** for each parameter. Clicking **Inspect** shows the **Metrics** page where you can see metrics and a related graph.

For example, clicking **Inspect** for **CPU Utilisation** shows a graph and values for the following metrics:

cluster:node\_cpu:ratio\_rate5m{cluster=""}

|  |
| --- |
|  |

Figure 6.5: Metrics CPU utilisation

The **Kubernetes / Compute Resources / Cluster** dashboard page also shows graphs for CPU, memory, and network, such as **CPU Usage**, **CPU Quota**, **Memory Usage**, and **Memory Quota**.

These graphs are common to some dashboard pages. The only difference is data filtration. For example, the **Kubernetes / Compute Resources / Namespace (Workloads)** dashboard filters resource usage, first by namespace and then by workload type, such as by deployment, daemon set, and stateful set.

**USE Method / Cluster**

USE stands for Utilisation Saturation and Errors. This dashboard displays several graphics to identify whether the cluster is overutilised, oversaturated, or experiencing many errors. Because the dashboard displays all nodes in the cluster, you might be able to identify a node that is not behaving in the same way as the other nodes in the cluster.

The following graphic indicates that the worker03 node is experiencing higher memory saturation than other nodes in the cluster.

|  |
| --- |
|  |

Figure 6.6: USE Method / Cluster - Memory Saturation

### References

For more information about OpenShift monitoring, refer to the Monitoring chapter in the Red Hat OpenShift Container Platform 4.14 Observability documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/monitoring/index#monitoring-overview>

For more information about enabling monitoring for user-defined projects, refer to the Enabling Monitoring for User-defined Projects chapter in the Red Hat OpenShift Container Platform 4.14 Observability documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/monitoring/index#enabling-monitoring-for-user-defined-projects>

[Red Hat OpenShift Observability](https://www.redhat.com/en/technologies/cloud-computing/openshift/observability)

[Check Out the new Network Observability Support in OpenShift 4.12](https://www.redhat.com/en/blog/check-out-the-new-network-observability-support-in-openshift-4.12)

[Observability Across OpenShift Cluster Boundaries with Distributed Data Collection](https://www.redhat.com/en/blog/observability-across-openshift-cluster-boundaries-with-distributed-data-collection)

[Prometheus Overview](https://prometheus.io/docs/introduction/overview)

[Prometheus Data Model](https://prometheus.io/docs/concepts/data_model)

[Node Exporter](https://github.com/prometheus/node_exporter)

[Exposed Metrics - kube-state-metrics](https://github.com/kubernetes/kube-state-metrics/tree/master/docs#exposed-metrics)

[Exposed Metrics - openshift-state-metrics](https://github.com/openshift/openshift-state-metrics/blob/master/docs/README.md#exposed-metrics)

[Querying Prometheus](https://prometheus.io/docs/prometheus/2.46/querying/basics/)

## **Guided Exercise: Cluster Monitoring**

Extract specific insights from monitoring dashboards and metrics queries to troubleshoot performance and availability issues with cluster nodes.

**Outcomes**

* Use the OpenShift monitoring default stack to identify a deployment that consumes excessive CPU and memory.
* Use the OpenShift monitoring default stack to find the cause of an availability issue.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start monitoring-cluster**

**Instructions**

Your company has an OpenShift cluster for development and testing environments with three compute nodes. The cluster has two node pools.

The first node pool is for the development environment and uses the env=dev label. In this classroom environment, the node pool for the development environment has one worker03 compute node. The development environment node pool has the dev-monitor and dev-finance namespaces. The development applications are deployed in the dev-monitor and dev-finance namespaces. The latest performance testing for each application suggests that with the given load, the applications use minimal resources and do not exceed 100% of the CPU and memory requests. Use the OpenShift monitoring default stack to identify whether a user workload consumes excessive CPU and memory. The load generator traffic-simulator.sh script is available at the ~/DO380/labs/monitoring-cluster location.

The second node pool is for the testing environment and uses the env=test label. In this classroom environment, the node pool for the testing environment has one worker02 compute node. The testing environment node pool has a test-monitor namespace. The frontend-test and budget-test test applications are deployed in the test-monitor namespace. The URLs for both applications is as follows:

**frontend-test**

* http://frontend-test-monitor.apps.ocp4.example.com

**budget-test**

* http://budget-test-monitor.apps.ocp4.example.com

Both applications are giving an Application Not Available message. Although the developers are trying to deploy more applications for testing, the pods are stuck at the Pending state. Use the OpenShift monitoring default stack to find the cause of this availability issue.

1. Run the load generator for applications that are deployed in the development environment. The load generator traffic-simulator.sh script is available at the ~/DO380/labs/monitoring-cluster location.
   1. Open a terminal window and navigate to the ~/DO380/labs/monitoring-cluster directory.

[student@workstation ~]$ **cd ~/DO380/labs/monitoring-cluster**

* 1. Execute the traffic-simulator.sh script to generate the load for applications that are deployed in the development environment.
  2. [student@workstation monitoring-cluster]$ **./traffic-simulator.sh**

...output omitted...

1. Use the cluster monitoring dashboards to display cluster resource usage information within the OpenShift web console.
   1. Log in as the admin user to the OpenShift web console. To do so, open a web browser window and navigate to https://console-openshift-console.apps.ocp4.example.com.

Then click **Red Hat Identity Management**.

* 1. Log in as the admin user with redhatocp as the password.
  2. Navigate to **Observe** → **Dashboards**.
  3. Select the **Kubernetes / Compute Resources / Cluster** dashboard.
  4. Scroll to the **CPU Quota** section. Click the **CPU Usage** column header, if necessary more than once, until it turns blue with a downward arrow. Rows are sorted by namespace in descending CPU order.

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

* 1. Figure 6.7: Namespaces with the highest CPU usage
  2. The dev-monitor and dev-finance namespaces are likely listed among the five namespaces that use the most CPU resources. Although the dev-monitor namespace requests only 0.65 CPU resources, it uses considerably more. The result is that the **CPU Requests %** column reports more than 100%.
  3. If you do not see the dev-monitor namespace with high resource consumption, then reload the page in your browser.
  4. Scroll to the **Memory Requests** section. Click the **Memory Usage** column header, if necessary more than once, until it turns blue with a downward arrow. Rows are sorted by namespaces in descending memory order.

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

* 1. Figure 6.8: Namespaces with the highest memory usage
  2. The dev-monitor namespace is likely listed among the five namespaces that use the most memory resources. As with CPU usage, the dev-monitor namespace uses considerably more memory than the 350 MiB that the pods in the namespace request. The result is that the **Memory Requests %** column reports more than 100%.

1. Use the cluster monitoring dashboards to identify the workload and pod that uses the most cluster resources in the dev-monitor namespace.
   1. Select the **Kubernetes / Compute Resources / Namespace (Workloads)** dashboard. Select the dev-monitor namespace and leave deployment as the workload type.
   2. Scroll to the **CPU Quota** section. The **CPU Quota** section shows that the frontend deployment requests a total of 0.5 CPU resources and currently uses 0.001 CPU resources. The exoplanets-app and exoplanets-db deployments request a total of 0.05 CPU resources, and currently use 0.007 and 0.003 CPU resources respectively. The python-app deployment requests a total of 0.1 CPU resources, but the one pod in the deployment consumes more than 2 CPU resources.

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

* 1. Figure 6.9: Deployment CPU usage
  2. Although the actual CPU usage in your cluster might differ, it is clear that the python-app deployment uses the most CPU resources.
  3. Scroll to the **Memory Usage** section. The **Memory Usage** graph shows that the python-app deployment has notable increases and decreases in memory usage. However, the ten pods in the frontend deployment consistently use a total of about 63 MiB of memory. One pod in the exoplanets-app deployment uses a total of about 15.02 MiB of memory, and one pod in the exoplanets-db deployment uses a total of about 129.2 MiB of memory.

|  |
| --- |
| A white background with black lines  Description automatically generated |

* 1. Figure 6.10: Deployment memory usage

1. You identify a deployment that appears to be behaving erratically and that uses more resources than intended. The next step might be to identify the owner of the python-app deployment to verify the application.

At minimum, the owner should add resource limits for both CPU and memory to the python-app deployment. Although a cluster administrator can add limits, setting incorrect limits might cause the pod to enter a CrashLoopBackOff state.

Similarly, a cluster administrator can implement a quota for the dev-monitor namespace to limit the total CPU and memory resources that the project uses. Because the python-app deployment does not specify limits for either CPU or memory, restricting CPU and memory resources with a quota would prevent the python-app pod from running.

1. Return to the terminal where the script is running. Press **Ctrl**+**C** to stop the script, and then close the terminal window.
2. Verify that the frontend-test and budget-test test applications are giving an Application Not Available message.
   1. Open a web browser window and navigate to the URL for the budget-test application http://budget-test-monitor.apps.ocp4.example.com. The web page shows the Application Not Available message.

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

* 1. Figure 6.11: Application not available
  2. Open a web browser tab and navigate to the URL for the frontend-test application http://frontend-test-monitor.apps.ocp4.example.com. The web page shows the Application Not Available message.

1. Use the cluster monitoring dashboards to monitor the workloads and pods in the test-monitor namespace.
   1. Switch to the OpenShift web console web browser tab and select the **Kubernetes / Compute Resources / Namespace (Workloads)** dashboard. Select the test-monitor namespace and leave deployment as the workload type.
   2. Scroll to the **CPU Quota** section. The **CPU Quota** section shows that the frontend-test deployment requests a total of 1 CPU resource and currently uses almost no CPU resources. The budget-test deployment requests a total of 1 CPU and currently uses almost no CPU resources.

|  |
| --- |
| A screenshot of a running pod  Description automatically generated |

* 1. Figure 6.12: Deployment CPU usage for the test environment
  2. Scroll to the **Memory Quota** section. The **Memory Quota** section shows that the budget-test and frontend-test deployments currently use no memory.

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

* 1. Figure 6.13: Deployment memory usage for the test environment
  2. Similarly, neither deployment currently shows any network usage.
  3. The applications are showing no CPU, memory, or network usage, and developers cannot deploy applications in the test environment. The testing environment has one worker02 compute node.

1. Use the cluster monitoring dashboards to monitor the state of compute node usage for the test environment.
   1. Select the **Node Cluster** dashboard. The **NotReadyNodesCount** section shows that one node is in the not ready state.
   2. Click **Inspect** in the **NotReadyNodesCount** section. The metric shows that the sum of nodes with an unknown or false status is 1. This metric value implies that one node is not ready.
   3. Use the following metric query to the filter to identify the node with an unknown or false status.

**kube\_node\_status\_condition{condition="Ready", status=~"unknown|false"} == 1**

The metric lists all entries, and the == 1 condition filters to show the entries with an unknown or false status.

Click **Run queries**.

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

Figure 6.14: Node status metric

The output shows that the worker02 node has unknown status.

1. You identify that the worker02 compute node is not ready. The test applications are giving an Application Not Available message, because the test environment node pool has only compute node, which is not ready.

The cluster administrator can add a healthy node to the test node pool for application availability and troubleshoot the worker02 compute node.

1. Close the web browser.

**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 monitoring-cluster**

## **Alerts and Notifications**

### **Objectives**

* View alerting rules.
* Configure and silence alerts.

### **OpenShift Alerts**

In OpenShift Container Platform, you can use the Alerts UI to manage alerts, silences, and alerting rules.

**Alerting rules**

Alerting rules contain a set of conditions that outline a particular state within a cluster. Alerts are triggered when those conditions are true. An alerting rule can be assigned a severity that defines how the alerts are routed.

**Alerts**

An alert is fired when the conditions that are defined in an alerting rule are true. Alerts notify that a set of conditions apply in an OpenShift Container Platform cluster.

**Alert receivers**

You can configure the alerting system to route alerts to a receiver and send notifications via email, send pager notifications, or forward them to another system by using a webhook.

**Silences**

A silence can be applied to an alert to prevent sending notifications when the conditions for an alert are true. You can mute an alert after the initial notification, while you work on resolving the underlying issue.

### **Viewing Alerting Rules**

Click **Observe** → **Alerting** and then click **Alerting rules** to list the alerting rules in the OpenShift cluster. You can apply custom filters to search for a specific alert rule. The alert state column indicates whether the alert is currently firing or is silenced.

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

Click the alerting rule name to view its details such as its severity, description, and log message. You can also view the PromQL expression, which checks for the alert.

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

Scroll down to view the graph that displays the time when the alert was detected.

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

### **Viewing Firing Alerts**

Click **Observe** → **Alerting** to view the alerts that are currently firing in the cluster. You can apply custom filters to find a specific alert.

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

Click the alert name to view its properties. The graph displays the time when the alert was detected.

|  |
| --- |
|  |

Scroll down to view the alert details and labels. You can create a receiver to match a label and forward the alert.

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

### **Silence Alerts**

You can silence an alert for a period of time to stop sending new notifications to a receiver. For example, you can silence an alert while you are troubleshooting a problem, to stop sending new alert emails, or silence certain alerts that are expected to be fired during a scheduled maintenance window. Click **Observe** → **Alerting** to display a list of the alerts that are currently firing. Then click the three dots icon **⋮** at the right of the alert, and click **Silence alert**.

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

The alert silence is configured to start immediately after it is created. You can clear the **Start immediately** field to set a specific start time.

You can select a predefined duration for the alert silence. Select - in the **For** field to set when the silence begins and finishes.

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

A comment is required when you create an alert silence. This comment is saved in the alert silence details.

|  |
| --- |
| A blue and white line  Description automatically generated |

#### **Viewing Alert Silences**

Click **Observe** → **Alerting** and then click **Silences** to view the alert silences in the cluster.

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

You can see the silence name, start time, end time, matching labels, state, and associated comment.

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

You can also view the alerts that match the silence.

|  |
| --- |
| A white background with black text  Description automatically generated |

#### **Viewing Silenced Alerts**

Click **Observe** → **Alerting** to return to the alerting main section. The PersistentVolumeUsageNearFull alert is not displayed in the list because it is currently silenced. Clear the **Alert State** filter by clicking **x** to display all the alerts.

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

Observe that the PersistentVolumeUsageNearFull alert is listed and marked as silenced.

|  |
| --- |
|  |

### **Sending Alert Notifications**

You can view the firing alerts in the Alerting UI. Alerts are not configured by default to be sent to any notification systems. You can configure OpenShift Container Platform to send alerts to the following receiver types:

* Email
* PagerDuty
* Slack
* Webhook

By routing alerts to receivers, you can send timely notifications to the appropriate teams when failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or to a critical response team. Alerts that provide non-critical warning notifications might instead be routed to a ticketing system for non-immediate review.

The alerting system uses Alertmanager, which is a component from the Prometheus monitoring stack. The Alertmanager configuration is saved in the alertmanager-main secret in the openshift-monitoring namespace.

#### **Configuring Alertmanager with the Web Console**

Click **Administration** → **Cluster Settings**, and then click **Configuration** and **Alertmanager** to open the Alertmanager configuration page. You can click the vertical ellipsis icon **⋮** at the right to create an alert receiver or to edit the Alertmanager configuration file.

|  |
| --- |
|  |

The Alertmanager configuration page lists the current alert routing parameters.

|  |
| --- |
|  |

Scroll down to view the configured alert receivers. Click **Create receiver** to add a receiver. You can click the three dots icon **⋮** at the right of each receiver to edit its settings or to delete it.

|  |
| --- |
| A screenshot of a login form  Description automatically generated |

You can view and edit the current Alertmanager configuration by clicking **YAML**.

|  |
| --- |
|  |

#### **Configuring the Alerting Email Receiver**

Click **Create Receiver** to create a receiver to send the alerts by email. Complete the values according to the following table, and then scroll down and click **Create**.

| **Field** | **Value** |
| --- | --- |
| **Receiver name** | email |
| **Receiver type** | Email |
| **To address** | ocp-admins@example.com |
| **SMTP configuration** | |
| **Save as default SMTP configuration** | **Checked** |
| **From address** | alerts@ocp4.example.com |
| **SMTP smarthost** | 192.168.50.254:25 |
| **SMTP hello** | localhost |
| **Auth username** | smtp\_training |
| **Auth password** | Red\_H4T@! |
| **Auth identity** | (empty) |
| **Auth secret** | (empty) |
| **Require TLS** | ***Unset*** |
| **Routing labels** | |
| **Value** | alertname=PersistentVolumeUsageNearFull |

The Alertmanager configuration changes are applied and the alerting system reloads in a few minutes.

#### **Configuring Alertmanager with the Command Line**

The Alertmanager configuration is saved in the alertmanager-main secret in the openshift-monitoring namespace.

[user@host ~]$ **oc get secret/alertmanager-main -n openshift-monitoring**

NAME TYPE DATA AGE

alertmanager-main Opaque 1 7d

You can extract the alertmanager-main secret to view the alertmanager.yaml configuration file.

[user@host ~]$ **oc extract secret/alertmanager-main -n openshift-monitoring \**

**--to ./ --confirm**

alertmanager.yaml

The default alertmanager.yaml configuration file contains many unnecessary quotation marks. Remove the quotation marks by using the sed command to improve readability.

[user@host ~]$ **sed -f script.sed alertmanager.yaml**

### Important

Although removing the extraneous quotation mark characters is not required, it improves readability. The quotation mark characters are not required in a YAML file, except to represent null as a string.

The previous sed command uses this script file to remove all the quotation marks and converts null to a string.

**#!/usr/bin/sed -f**

s/"//g

s/\<\(null\)\>/'\1'/g

|  |  |
| --- | --- |
|  | Remove all the quotation marks from the file. |
|  | Enclose null between quotation marks. |

The monitoring stack can send alerts by email through an SMTP server. The following example sends the PersistentVolumeUsageNearFull alerts to the ocp-admins@example.com email address.

global:

resolve\_timeout: 5m

**smtp\_from**: **alerts@ocp4.example.com**

**smtp\_smarthost**: **'192.168.50.254:25'**

**smtp\_hello**: **localhost**

**smtp\_auth\_username**: **smtp\_training**

**smtp\_auth\_password**: **Red\_H4T@!**

**smtp\_require\_tls**: **false**

...output omitted...

inhibit\_rules:

...output omitted...

receivers:

...output omitted...

- **name**: **email**

**email\_configs**:

- **to**: **ocp-admins@example.com**

route:

group\_by:

- namespace

**group\_interval**: **2m**

group\_wait: 30s

receiver: Default

**repeat\_interval**: **1m**

routes:

...output omitted...

- **receiver**: **email**

**match**:

**alertname**: **PersistentVolumeUsageNearFull**

**continue: true**

...output omitted...

|  |  |
| --- | --- |
|  | The global SMTP host. If you do not define smarthost in the email\_configs field for a receiver, then this field is the default host in use. |
|  | The global email sender address. If you do not define from in the email\_configs field for a receiver, then this field is the default address in use. |
|  | The hello parameter for the SMTP connection. |
|  | The global SMTP username for optional authentication. If you do not define auth\_username in the email\_configs field for a receiver, then this field is the default username in use. |
|  | The global SMTP password for optional authentication. This password is used if auth\_password is not defined in the email\_configs field for a receiver. If you do not define auth\_password in the email\_configs field for a receiver, then this field is the default password in use. |
|  | A global setting to specify whether TLS is required for SMTP. You can override this setting by using require\_tls in the email\_configs field for a receiver. |
|  | An arbitrary name for the receiver. A route specifies this receiver name for a match. |
|  | This setting indicates that the receiver sends alerts by email. |
|  | The to setting must be specified in the email\_configs field, and does not have an equivalent global SMTP setting. |
|  | Configure the group\_interval and repeat\_interval fields so the alert email notifications are sent more frequently. |
|  | The receiver to use if the match evaluates as true for the alert. |
|  | The expression to match a specific alert name. |
|  | By default, every alert that enters the routing tree stops after the first matching child. The continue: true parameter permitts the alert to continue through the routing tree matching subsequent routes. |

You can update the Alertmanager configuration by setting the data of the alertmanager-main secret in the openshift-monitoring namespace with the content of the alertmanager.yaml file.

[user@host ~]$ **oc set data secret/alertmanager-main -n openshift-monitoring \**

**--from-file alertmanager.yaml**

secret/alertmanager-main data updated

You can view the progression in the Alertmanager stateful set logs. A successful update generates the log message: Completed loading of configuration file.

[user@host ~]$ **oc logs -f statefulset.apps/alertmanager-main -c alertmanager \**

**-n openshift-monitoring**

Found 2 pods, using pod/alertmanager-main-***0***

...output omitted...

ts=2024-01-31T01:02:03.064Z caller=coordinator.go:113 level=info component=configuration msg="Loading configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml

**ts=2024-01-31T01:02:03.128Z caller=coordinator.go:126 level=info component=configuration msg="Completed loading of configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml**

An incorrect configuration generates a failed log message. If you see configuration errors in the logs, then modify the alertmanager.yaml file and reapply your changes to the alertmanager-main secret in the openshift-monitoring namespace.

...output omitted...

ts=2024-01-31T02:03:04.256Z caller=coordinator.go:113 level=info component=configuration msg="Loading configuration file" file=/etc/alertmanager/config\_out/alertmanager.env.yaml

**ts=2024-01-31T02:03:04.512Z caller=coordinator.go:118 level=error component=configuration msg="Loading configuration file failed" file=/etc/alertmanager/config\_out/alertmanager.env.yaml err="no global SMTP smarthost set"**

### References

For more information about managing alerts in OpenShift, refer to the Managing Alerts chapter in the Red Hat OpenShift Container Platform 4.14 Monitoring documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/monitoring/index#managing-alerts>

[Querying Prometheus](https://prometheus.io/docs/prometheus/2.46/querying/basics/)

[AlertManager Configuration](https://prometheus.io/docs/alerting/0.25/configuration/)

# **Chapter 7.  OpenShift Logging**

[Log Forwarding](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07)

[Guided Exercise: Log Forwarding](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07s02)

[Centralized Logging](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07s03)

[Guided Exercise: Centralized Logging](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07s04)

[Lab: OpenShift Logging](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07s05)

[Summary](https://rol.redhat.com/rol/app/courses/do380-4.14/pages/ch07s06)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Deploy OpenShift Logging and query log entries from workloads and cluster nodes. |
| **Sections** | * Log Forwarding (and Guided Exercise) * Centralized Logging (and Guided Exercise) |
| **Lab** | * OpenShift Logging |

## **Log Forwarding**

### **Objectives**

* Deploy OpenShift Logging for forwarding logs to an external aggregator.

### **OpenShift Logging**

OpenShift Logging collects and aggregates the log messages from all the pods and nodes in your cluster. Users and administrators can use the OpenShift web console to search and consult log entries.

Depending on the workload and the size of your cluster, processing many logs can require significant disk space and compute resources, which would then not be available for your application workloads. In such a scenario, you might need to deploy more compute nodes and increase your storage capacity to handle the extra load.

You can configure OpenShift Logging to forward the logs to a third-party log aggregator for long-term storage, or to an observability platform for further analysis, and minimize the resource requirement on your cluster.

The following examples of third-party logging solutions can receive logs from OpenShift Logging:

* Elasticsearch
* Grafana Loki
* Splunk
* Amazon CloudWatch
* Google Cloud Logging

### **OpenShift Logging Components**

OpenShift Logging is based on these components: a collector, a log store, and a visualization console. You can deploy them together as a complete logging solution, or you can deploy the collector alone and store the logs in an external solution.

|  |
| --- |
|  |

Figure 7.1: OpenShift Logging architecture

#### **Log Collector**

The collector is the main component of OpenShift Logging. OpenShift Logging uses Vector to collect logs from all running containers and cluster nodes, and replaces Fluentd, which was the collector in earlier versions of OpenShift Logging.

Vector collects various log types from your cluster that are then grouped into these categories:

**Infrastructure**

Infrastructure logs include container logs in the openshift-\*, kube\*, and default namespaces, and system logs from the cluster nodes.

**Audit**

Audit logs include both Kubernetes API and OpenShift API audit logs, as well as the Linux audit logs from the cluster nodes. These logs might contain sensitive security details, and OpenShift Logging does not store them by default.

**Application**

Application logs are all container logs from user projects.

To collect those logs, Vector runs as a daemon set in the cluster and therefore runs on all nodes.

In addition to collecting logs, Vector adds metadata to describe where the logs come from, and then forwards the logs to the log store, which is either internal or external to the cluster.

#### **Log Store**

The log store uses Grafana Loki to aggregate logs from the entire cluster into a central place and provides access control to logs.

Loki replaces Elasticsearch, which was the log store in earlier versions of the logging subsystem.

The internal log store is an optional component of OpenShift Logging.

#### **Visualization**

OpenShift Logging provides a native OpenShift Console plug-in to view and query logs in the internal log store.

The OpenShift Logging UI component replaces Kibana, which was the web interface in earlier versions of OpenShift Logging.

### **Install and Configure OpenShift Logging**

You can deploy OpenShift Logging by installing the OpenShift Logging operator with the Operator Lifecycle Manager (OLM) from OperatorHub on the web console or by using the oc command to create the OLM resources. The OpenShift Logging operator manages deploying and configuring the log collector and other resources to support the logging subsystem.

See the references section for instructions to install an operator by using OperatorHub or the oc command.

After installation, you configure the logging subsystem by using the ClusterLogging custom resource (CR), and configure the log collector by using the ClusterLogForwarder CR.

#### **Configure OpenShift Logging Components**

The ClusterLogging CR configures and manages deploying the OpenShift Logging components.

To configure OpenShift Logging, you must create the cluster logging resource in the openshift-logging namespace. The minimum configuration for OpenShift Logging is to enable the log collector, as follows:

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: instance

namespace: openshift-logging

spec:

managementState: Managed

collection:

type: vector

|  |  |
| --- | --- |
|  | The resource name must be instance. |
|  | The management state must be Managed for the components to receive updates from the OpenShift Logging operator. |
|  | The collector type to deploy. It can be either vector or the deprecated fluentd collector. |

You can use the cluster logging resource to enable or disable logging components, to customize pod placement, and to define pod resource limits. Configuring the log store and logging UI components are discussed in detail in the next section.

In the following example, the log collector component is configured to run on all nodes, including dedicated infrastructure nodes with the node-role.kubernetes.io/infra=reserved:NoExecute and node-role.kubernetes.io/infra=reserved:NoSchedule taints:

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: instance

namespace: openshift-logging

spec:

managementState: Managed

collection:

type: vector

tolerations:

- effect: NoSchedule

key: node-role.kubernetes.io/infra

value: reserved

- effect: NoExecute

key: node-role.kubernetes.io/infra

value: reserved

|  |  |
| --- | --- |
|  | List of tolerations to include in the collector daemon set. |

### Note

OpenShift Logging automatically adds the node-role.kubernetes.io/master:NoSchedule toleration to the collector daemon set so the collector can run on both control planes and compute nodes.

After the cluster logging resource is created, the OpenShift Logging operator starts scheduling pods for the components. The log collector pods are deployed only if the internal log store is enabled, or if log forwarding to an external log aggregator is configured.

### **Configure Log Collection and Forwarding**

The log collector is configured by default to forward infrastructure and application logs to the internal log store that you define in the cluster logging resource. If the internal log store is not deployed, then you must configure the log collector to forward logs to a third-party logging system instead.

Vector can forward logs to various log stores, in addition to the internal Loki deployment, such as Splunk, Amazon CloudWatch, or any logging solution that uses the syslog protocol.

See the references section for a list of external log stores that OpenShift Logging can use.

The ClusterLogForwarder custom resource configures the log collector, and defines which logs to collect and where to send them. The following example is a cluster log forwarder resource that configures Vector to forward audit and infrastructure logs to an external Splunk instance:

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: instance

namespace: openshift-logging

spec:

outputs:

- name: splunk-receiver

secret:

name: splunk-auth-token

type: splunk

url: https://mysplunkserver.example.com:8088/services/collector

pipelines:

- name: to-splunk

inputRefs:

- audit

- infrastructure

outputRefs:

- splunk-receiver

|  |  |
| --- | --- |
|  | The resource name must be instance and in the openshift-logging namespace. |
|  | Name of the log output. You use that name in the log pipeline to refer to that log destination. |
|  | Secret to use for connecting to the log store, if required. In this example, the secret contains the authentication token for the Splunk instance. |
|  | Type of the external log store. |
|  | URL of the external log store. |
|  | Log pipeline configuration that forwards audit and infrastructure logs to the Splunk instance. |

The cluster log forwarder resource is composed of inputs, outputs, and pipelines.

**inputs**

An input defines the log type to collect. OpenShift Logging provides a predefined input for each log category: infrastructure, audit, and application.

**outputs**

An output defines a destination for the logs. You can configure multiple log outputs to one or more external log stores.

If the internal log store is configured, then the default output becomes available to target the internal Loki instance.

**pipelines**

A pipeline defines a routing from one or more log inputs to one or more log outputs. You can create several log pipelines and use any combination of log inputs and outputs to forward logs according to your needs.

In the next configuration example, the collector forwards logs to multiple log aggregators:

* The audit logs to a remote syslog server for long-term storage and security compliance
* Both audit logs and infrastructure logs to the internal log store for cluster administrators
* The application logs to an external Splunk instance for developers

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: instance

namespace: openshift-logging

spec:

outputs:

- name: audit-rsyslog

syslog:

appName: ocp-prod

facility: local0

msgID: audit-msg

procID: audit-proc

severity: informational

rfc: RFC5424

type: syslog

url: tcp://audit-store.example.com:5514

- name: splunk-receiver

secret:

name: splunk-auth-token

type: splunk

url: https://mysplunkserver.example.com:8088/services/collector

pipelines:

- name: audit-to-syslog

inputRefs:

- audit

outputRefs:

- audit-rsyslog

- name: admin-logs

inputRefs:

- audit

- infrastructure

outputRefs:

- default

- name: app-logs

inputRefs:

- application

outputRefs:

- splunk-receiver

|  |  |
| --- | --- |
|  | Log output configuration for the remote syslog |
|  | Specific configuration for syslog to format the log messages |
|  | Log output configuration for the Splunk instance |
|  | Log pipeline that forwards audit logs to the remote syslog |
|  | Log pipeline that forwards audit and infrastructure logs to the internal log store |
|  | Log pipeline that forwards application logs to the Splunk instance |

#### **Collect Kubernetes Events**

The Event Router is an optional OpenShift Logging component that you can deploy to log Kubernetes events. The Event Router monitors the OpenShift event API and sends the events to the container stdout. The log collector captures the Event Router container logs and forwards them to the log store through the infrastructure category.

Because the OpenShift Logging operator does not manage this component, it must be manually deployed and updated. You can deploy the Event Router by using the OpenShift template from the documentation.

See the references section for more information about the Event Router and its deployment.

#### **Filtering Log**

OpenShift Logging provides filtering capabilities to limit the number of logs to forward to the log store. For example, you can limit the log collection to a specific set of OpenShift projects or application pods by creating a custom log input in the cluster log forwarder resource, as follows:

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: instance

namespace: openshift-logging

spec:

inputs:

- name: production-apps

application:

selector:

matchLabels:

environment: production

- name: qa-chain

application:

selector:

matchLabels:

environment: development

namespaces:

- qa-testing

- builders

...output omitted...

pipelines:

- name: to-splunk

inputRefs:

- qa-chain

- production-apps

outputRefs:

- splunk-receiver

|  |  |
| --- | --- |
|  | The production-apps log input collects application logs from pods with the environment: production label. |
|  | The qa-chain log input collects application logs from pods with the environment: development label in the qa-testing and builders projects. |
|  | The log pipeline forwards both qa-chain and production-apps log inputs to the Splunk instance. |

In addition to application logs, you can filter Kubernetes API audit events.

Each call to the Kubernetes and OpenShift APIs generates an audit event that is forwarded to the log store. Although these events can be valuable for security audits, they represent a high volume of data, which can increase your storage requirement and network bandwidth usage, and therefore increase the cost of your logging solution.

To mitigate this effect, you can define audit filters that remove unwanted or low-value events from the audit logs, and reduce the data that is sent to the log store.

Audit filters are defined in the cluster log forwarder resource and list a set of rules that match the events to remove. You can then apply these filters to the selected log pipeline.

In the following example, an audit filter is created to remove all audit events from infrastructure service accounts, and to remove audit events from updates to leases resources by Kubernetes system users:

spec:

filters:

- name: unwanted-events

type: kubeAPIAudit

kubeAPIAudit:

rules:

- level: None

namespaces:

- openshift-\*

- kube\*

userGroups:

- system:serviceaccounts:openshift-\*

- system:nodes

- level: None

resources:

- group: coordination.k8s.io

resources:

- leases

users:

- system:kube\*

- system:apiserver

verbs:

- update

...output omitted...

pipelines:

- name: audit-to-syslog

inputRefs:

- audit

filterRefs:

- unwanted-events

outputRefs:

- audit-rsyslog

|  |  |
| --- | --- |
|  | Name of the filter. You use that name in the log pipeline to refer to that filter configuration. |
|  | The first rule removes all audit events that were created from users in the system:serviceaccounts:openshift-\* and system:nodes groups, for resources in the openshift-\* and kube\* namespaces. |
|  | The second rule removes audit events that were generated from the system:kube\* and system:apiserver users, who update leases resources in the coordination.k8s.io API group. |
|  | The filterRefs field lists all filters to apply to the log pipeline. |

The KubeAPIAudit field uses the same syntax as the Kubernetes audit policy resources. See the references section for more information about the Kubernetes audit policy and its syntax.

### **Troubleshoot Log Forwarding**

You can verify the configuration of the cluster log forwarder by reviewing the status of the resource. The OpenShift Logging operator validates each log input, output, and pipeline for configuration errors:

[user@host ~]$ **oc -n openshift-logging describe clusterlogforwarder/instance**

...output omitted...

Status:

Conditions:

Message: **clusterlogforwarder is not ready**

Reason: **ValidationFailure**

Status: True

Type: Validation

Inputs:

Critical - Apps:

Last Transition Time: 2024-01-18T14:24:20Z

Status: True

Type: Ready

Outputs:

Audit - Syslog:

Last Transition Time: 2024-01-18T14:24:20Z

Status: True

Type: Ready

Infra - Syslog:

Last Transition Time: 2024-01-18T14:24:20Z

Status: True

Type: Ready

Pipelines:

Audit - Syslog:

Last Transition Time: 2024-01-18T14:24:20Z

Status: True

Type: Ready

**Critical - Apps - Syslog:**

Last Transition Time: 2024-01-18T14:24:20Z

Message: **invalid: unrecognized outputs: [app-syslog], no valid outputs**

Reason: Invalid

Status: False

Type: Ready

...output omitted...

In this example, one of the configured log pipelines refers to an unknown log output (app-syslog). After the configuration is fixed, the cluster log forwarder resource enters the ready state.

You can also verify the status of the logging configuration by reviewing Kubernetes events about the logging resources:

[user@host ~]$ **oc get event -n openshift-logging \**

**--field-selector=involvedObject.name=instance**

LAST SEEN TYPE REASON OBJECT MESSAGE

2m25s Normal ReconcilingLoggingCR clusterlogging/instance Reconciling logging resource

78s Normal ReconcilingLoggingCR clusterlogforwarder/instance Reconciling logging resource

4m11s Warning Invalid clusterlogforwarder/instance clusterlogforwarder is not ready

78s Normal Ready clusterlogforwarder/instance ClusterLogForwarder is valid

Each time that the log forwarder configuration is updated, the OpenShift Logging operator redeploys the collector pods.

If the cluster log forwarder configuration is correct, but logs are still not forwarded to the external log store, then you can review the collector logs for errors:

[user@host ~]$ **oc -n openshift-logging logs daemonset/collector**

...output omitted...

... ERROR vector::topology::builder: msg="Healthcheck failed." error=**Connect error: Connection refused** (os error 111) component\_kind="sink" component\_type="socket" component\_id=infra\_syslog **component\_name=infra\_syslog**

In this example, Vector cannot reach the external log store that is configured in the infra\_syslog log output. Verify that the remote server is accessible from the OpenShift cluster and that the log forwarder configuration is correct.

If you enable the cluster monitoring on the openshift-logging namespace during the operator installation or by adding the openshift.io/cluster-monitoring: true label to the namespace, then you can use the monitoring dashboard to inspect the collector status.

From the **Logging / Collection** monitoring dashboard, you can see how many logs are forwarded to each configured log store and the log rate for each application that is running in your cluster.

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

Figure 7.2: OpenShift Logging monitoring dashboard

### References

For more information about compatible log destinations, refer to the Log Output Types chapter in the Red Hat OpenShift Container Platform 4.14 Logging documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#logging-output-types>

For more information about installing the OpenShift Logging operator, refer to the Installing Logging chapter in the Red Hat OpenShift Container Platform 4.14 Logging documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#cluster-logging-deploying>

For more information about installing the Event Router, refer to the Collecting and Storing Kubernetes Events chapter in the Red Hat OpenShift Container Platform 4.14 Logging documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#cluster-logging-eventrouter>

[Kubernetes Audit Policy](https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#audit-policy)

[ClusterLogForwarder API Audit Filter](https://github.com/openshift/cluster-logging-operator/blob/master/docs/features/logforwarding/filters/api-audit-filter.adoc)

## **Guided Exercise: Log Forwarding**

Deploy OpenShift Logging for forwarding logs to an external syslog server.

**Outcomes**

* Deploy the OpenShift Logging operator.
* Configure the OpenShift Logging operator to forward logs to a remote syslog.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start logging-forward**

**Instructions**

Your company requires you to send OpenShift logs to the company's centralized syslog server for long-term storage.

You must forward the following log types:

* Kubernetes API audit logs
* Kubernetes events
* CoreOS journal logs
* CoreOS audit logs
* Infrastructure container logs
* Critical application container logs

### Note

For critical application pods, the logging: critical label is assigned. Such applications are available in the shop OpenShift project to verify the log forwarding configuration.

Install the OpenShift Logging operator and configure log forwarding to the syslog server. Deploy the OpenShift Event Router to capture Kubernetes events and forward them to the syslog server.

Verify that all required log types are forwarded and available on the syslog server. Forwarded OpenShift logs are stored in the /var/log/openshift path on the utility machine.

1. As the admin user, install the OpenShift Logging operator.
   1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com.
   2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.
   3. Click **Operators** → **OperatorHub**. In the **Filter by keyword** field, type openshift logging to locate the OpenShift Logging operator, and then click **Red Hat OpenShift Logging**.

|  |
| --- |
|  |

* 1. The web console displays information about the Red Hat OpenShift Logging operator. Click **Install** to proceed to the **Install Operator** page.

|  |
| --- |
|  |

* 1. Click **Install** to install the operator with the default options.
  2. Wait until the installation is complete and the web console displays the ready for use message.

|  |
| --- |
|  |

1. Configure OpenShift Logging to enable the Vector log collector.
   1. From the terminal, log in to your OpenShift cluster as the admin user.

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

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

Login successful.

...output omitted...

* 1. Change to the openshift-logging project.

[student@workstation ~]$ **oc project openshift-logging**

Now using project "openshift-logging" on server ...

* 1. Change to the ~/DO380/labs/logging-forward directory.

[student@workstation ~]$ **cd ~/DO380/labs/logging-forward**

[student@workstation logging-forward]$

* 1. Edit and complete the ClusterLogging resource definition in the ~/DO380/labs/logging-forward/clusterlogging.yml file.

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: **instance**

namespace: openshift-logging

spec:

managementState: Managed

collection:

**type: vector**

### Note

You can use the solution in the ~/DO380/solutions/logging-forward/clusterlogging.yml file.

* 1. Apply the cluster logging configuration.

[student@workstation logging-forward]$ **oc apply -f clusterlogging.yml**

clusterlogging.logging.openshift.io/instance created

1. Configure OpenShift Logging to forward logs to the syslog server.

The syslog server is configured to filter OpenShift audit, infrastructure, and application logs into separate files in the /var/log/openshift path by using the msgID syslog attribute with the audit, infra, and apps values, respectively.

Configure the cluster log forwarder and define three log pipelines to send each log type to the syslog server by using the matching msgID value.

Ensure that only application logs with the logging: critical label are forwarded.

The syslog server DNS name is utility.lab.example.com and the service listens to the TCP port 514.

* 1. Edit and complete the ClusterLogForwarder resource definition in the ~/DO380/labs/logging-forward/clusterlogforwarder.yml file.

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: **instance**

namespace: openshift-logging

spec:

inputs:

- name: critical-apps

application:

selector:

**matchLabels:**

**logging: critical**

outputs:

- name: audit-syslog

type: syslog

url: **tcp://utility.lab.example.com:514**

syslog:

msgID: **audit**

...output omitted...

- name: apps-syslog

type: syslog

url: **tcp://utility.lab.example.com:514**

syslog:

msgID: **apps**

...output omitted...

- name: infra-syslog

type: syslog

url: **tcp://utility.lab.example.com:514**

syslog:

msgID: **infra**

...output omitted...

pipelines:

- name: critical-apps-syslog

inputRefs:

- **critical-apps**

outputRefs:

- apps-syslog

- name: infra-syslog

inputRefs:

- **infrastructure**

outputRefs:

- infra-syslog

- name: audit-syslog

inputRefs:

- **audit**

outputRefs:

- audit-syslog

|  |  |
| --- | --- |
|  | Label selector to include application pod logs with the logging: critical label only |
|  | Syslog parameters for the audit log type |
|  | Syslog parameters for the application log type |
|  | Syslog parameters for the infrastructure log type |
|  | Log pipelines for each log type |

### Note

You can use the solution in the ~/DO380/solutions/logging-forward/clusterlogforwarder.yml file.

* 1. Apply the configuration for the cluster log forwarder.

[student@workstation logging-forward]$ **oc apply -f clusterlogforwarder.yml**

clusterlogforwarder.logging.openshift.io/instance created

* 1. Verify that the OpenShift Logging operator deploys the collector pod on each node.

[student@workstation logging-forward]$ **oc get daemonset -l component=collector**

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE ...

collector **6** 6 **6** 6 6 ...

1. Deploy the Event Router component to capture Kubernetes events.
   1. Use the OpenShift template from the ~/DO380/labs/logging-forward/eventrouter.yml file to deploy the Event Router component.

[student@workstation logging-forward]$ **oc process -f eventrouter.yml \**

**| oc apply -f -**

serviceaccount/eventrouter created

clusterrole.rbac.authorization.k8s.io/event-reader created

clusterrolebinding.rbac.authorization.k8s.io/event-reader-binding created

configmap/eventrouter created

deployment.apps/eventrouter created

* 1. Verify that the Event Router is running.

[student@workstation logging-forward]$ **oc get pod -l component=eventrouter**

NAME READY STATUS RESTARTS AGE

eventrouter-59dfb998cc-9v7gb 1/1 **Running** 0 8m17s

1. Verify that the syslog service received the logs on the utility machine.
   1. Open a new terminal and connect to the utility machine with SSH as the root user.

[student@workstation logging-forward]$ **ssh root@utility**

### Note

The remainder of this activity refers to this new terminal as the utility terminal, and the first terminal as the workstation terminal.

* 1. Check that the OpenShift log files are created in the /var/log/openshift path.

[root@utility ~]# **ls -l /var/log/openshift**

total 298564

-rw-------. 1 root root 863869 Jan 11 11:38 **apps.log**

-rw-------. 1 root root 70381151 Jan 11 11:38 **audit.log**

-rw-------. 1 root root 204538257 Jan 11 11:38 **infra.log**

1. Verify that the journal logs from the cluster nodes are forwarded to the syslog server.
   1. On the utility terminal, change to the /var/log/openshift directory.

[root@utility ~]# **cd /var/log/openshift**

[root@utility openshift]#

* 1. Monitor the infrastructure log and filter the output with the jq command to list events from the sshd daemon on the cluster nodes.

[root@utility openshift]# **tail -f infra.log | egrep -o "\{.\*}$" \**

**| jq '. | select(.systemd.u.SYSLOG\_IDENTIFIER=="sshd")**

**| .hostname + " " + .message'**

### Note

The select() function of the jq command filters the log messages that match the provided expression. In this specific case, only messages with the .systemd.u.SYSLOG\_IDENTIFIER field with the value sshd are listed.

* 1. On the workstation terminal, connect to the master01 cluster node with the ssh command as the core user.

[student@workstation logging-forward]$ **ssh core@master01.ocp4.example.com**

[core@master01 ~]$

* 1. On the utility terminal, verify that the SSH connection is logged. Press **Ctrl**+**C** to exit the tail command.

"master01 main: sshd: ssh-rsa algorithm is disabled"

"master01 Accepted publickey for core from 172.25.250.9 port 51158 ssh2: RSA SHA256:M8i...BT0"

"master01 pam\_unix(sshd:session): session opened for user core(uid=1000) by (uid=0)"

**^C**

[root@utility openshift]#

* 1. On the workstation terminal, close the SSH connection with the exit command or **Ctrl**+**D**.

[core@master01 ~]$ **exit**

logout

Connection to master01.ocp4.example.com closed.

[student@workstation logging-forward]$

1. Verify that the audit logs from the cluster nodes are forwarded to the syslog server.
   1. On the utility terminal, review the audit log. Use the jq command to list the USER\_LOGIN Linux audit events that are generated from the previous SSH connection to the master01 node.

[root@utility openshift]# **egrep -ho "\{.\*}$" audit.log\* \**

**| jq '. | select(."audit.linux".type == "USER\_LOGIN")**

**| ."@timestamp" + " " + .hostname + " " + .message'**

...output omitted...

"2024-01-12T11:08:25.618+00:00 master01 type=USER\_LOGIN msg=audit(1705057705.618:937): pid=106902 uid=0 auid=1000 ses=11 subj=system\_u:system\_r:sshd\_t:s0-s0:c0.c1023 msg='op=login id=1000 exe=\"/usr/sbin/sshd\" hostname=? addr=172.25.250.9 terminal=/dev/pts/0 res=success'\u001dUID=\"root\" AUID=\"core\" ID=\"core\""

### Note

To use a JSON key that contains special characters, such as ., @ or - with the jq command, you must surround the key name with double quotes.

1. Verify that the infrastructure container logs are forwarded to the syslog server.
   1. On the utility terminal, monitor the infrastructure log. Use the jq command to display container logs from the openshift-storage namespace. Press **Ctrl**+**C** to exit the tail command.

[root@utility openshift]# **tail -f infra.log | egrep -o "\{.\*}$" \**

**| jq '. | select(.kubernetes.namespace\_name == "openshift-storage")**

**| .kubernetes.pod\_name + " " + .message'**

...output omitted...

"noobaa-core-0 ... time=\"2024-01-12T11:45:54Z\" level=info msg=\"..."

"noobaa-operator-... time=\"2024-01-12T11:45:54Z\" level=info msg=\"..."

**^C**

[root@utility openshift]#

1. Verify that the Kubernetes audit logs are forwarded to the syslog server.
   1. On the utility terminal, monitor the audit log. Use the jq command to display audit events about the developer user.

[root@utility openshift]# **tail -f audit.log | egrep -o "\{.\*}$" \**

**| jq -c '.|select(.user.username == "developer")**

**| [.user.username, .annotations, .verb, .objectRef]'**

* 1. On the workstation terminal, log in to the OpenShift cluster as the developer user with developer as the password.

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

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

Login successful.

...output omitted...

* 1. Create the logger-app project.

[student@workstation logging-forward]$ **oc new-project logger-app**

Now using project "logger-app" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. On the utility terminal, verify that the Kubernetes audit events from the developer user are logged. Press **Ctrl**+**C** to exit the tail command.

["developer",{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"basic-users\" of ClusterRole \"basic-user\" to Group \"system:authenticated\""},"get",{"apiGroup":"user.openshift.io","apiVersion":"v1","name":"~","resource":"users"}]

...output omitted...

["developer",{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"self-provisioners\" of ClusterRole \"self-provisioner\" to Group \"system:authenticated:oauth\""},"create",{"apiGroup":"project.openshift.io","apiVersion":"v1","name":"logger-app","resource":"projectrequests"}]

**^C**

[root@utility openshift]#

1. Verify that the Kubernetes events are forwarded to the syslog server.
   1. On the utility terminal, monitor the infrastructure log. Use the jq command to display the Kubernetes events on the logger-app namespace.

[root@utility openshift]# **tail -f infra.log | egrep -o "\{.\*}$" \**

**| jq -c '.**

**| select(.kubernetes.event.involvedObject.namespace == "logger-app")**

**| [.kubernetes.event.involvedObject.kind,**

**.kubernetes.event.involvedObject.name, .message]'**

* 1. On the workstation terminal, deploy the logger application by using the definition in the ~/DO380/labs/logging-forward/logger.yml file.

[student@workstation logging-forward]$ **oc apply -f logger.yml**

deployment.apps/logger-app created

* 1. On the utility terminal, verify that the Kubernetes events from the logger-app namespace are logged. Press **Ctrl**+**C** to exit the tail command.

...output omitted...

["Pod","logger-app-...","Add eth0 [10.11.0.18/23] from ovn-kubernetes"]

["Pod","logger-app-...","Container image \"registry.ocp4.example.com:8443/redhattraining/access-logger:v0.1\" already present on machine"]

["Pod","logger-app-...","Created container access-logger"]

["Pod","logger-app-...","Started container access-logger"]

**^C**

1. Verify that only for critical applications, such as the shop-web application in the shop project, the container logs are forwarded to the syslog server.
   1. On the utility terminal, review the application log. Verify that container logs from the shop-web application are forwarded.

[root@utility openshift]# **grep -m1 shop-web apps.log | egrep -o "\{.\*}" | jq**

{

"@timestamp": "2024-01-11T16:23:14.051244304Z",

...output omitted...

"container\_name": "shop-web",

"labels": {

"app": "shop-web",

"logging": "critical",

"pod-template-hash": "5cf65ffc4"

},

"namespace\_name": "shop",

...output omitted...

"log\_type": "application",

"message": "2024/01/11 16:23:14 \"GET /products?id=371 HTTP/1.1\" 403 \"Mozilla/5.0 (X11; Linux x86\_64; rv:25.0) Gecko/20100101 Firefox/33.0\"",

}

* 1. Verify that container logs from the logger-app application that was deployed in the previous step are not forwarded.

[root@utility openshift]# **grep logger-app apps.log**

***no output expected***

* 1. Close the utility terminal and return to the workstation terminal.

1. Clean up the resources.
   1. Change to the student HOME directory.
   2. [student@workstation logging-forward]$ **cd**

[student@workstation ~]$

* 1. Log in to your OpenShift cluster as the admin user.

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

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

Login successful.

...output omitted...

* 1. Delete the logger-app project.

[student@workstation ~]$ **oc delete project logger-app**

project.project.openshift.io "logger-app" deleted

* 1. Delete the ClusterLogging and ClusterLogForwarder resources.

[student@workstation ~]$ **oc -n openshift-logging \**

**delete clusterlogging,clusterlogforwarder --all**

clusterlogging.logging.openshift.io "instance" deleted

clusterlogforwarder.logging.openshift.io "instance" deleted

* 1. Remove the Event Router deployment.

[student@workstation ~]$ **oc -n openshift-logging delete deployment eventrouter**

deployment.apps "eventrouter" deleted

**Finish**

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

[student@workstation ~]$ **lab finish logging-forward**

## **Centralized Logging**

### **Objectives**

* Deploy OpenShift Logging for short-term log retention and aggregation.

### **Log Retention**

By default, Kubernetes stores logs from pods in the local disk on the nodes. These logs are ephemeral, because the information is lost when the pod is deleted. You can keep the pods in the cluster instead of deleting them, to keep access to the logs. However, Kubernetes garbage collectors delete the pods when such disk space is needed, and access to the logs is lost. Kubernetes also removes the logs when a node is restarted, for example after applying an update.

Administrators and developers need access to the logs to debug the applications in the event of issues, to analyze application performance, or to review object state changes. A one-week retention period is typically enough for users and administrators to review these logs.

OpenShift Logging uses Grafana Loki as the internal log store to aggregate logs from the entire cluster into a central place and to provide access control to logs. Loki replaces the deprecated Elasticsearch log store from earlier versions of OpenShift Logging. OpenShift Logging can also use other external log stores, such as Splunk, Amazon CloudWatch, or Google Cloud Logging.

### **Loki Log Store**

Loki is a horizontally scalable, highly available, and multitenant log aggregation system. Loki indexes only a few fixed labels during ingestion, and then compresses and stores the log data in chunks in object stores. The Loki operator supports object stores such as AWS S3, Azure, and OpenShift Data Foundation. The Loki operator includes the LokiStack CR to manage the Loki deployment.

The steps for using Loki as the log store for OpenShift Logging are as follows:

1. Install the Loki operator.
2. Configure the Loki object storage.
3. Create a LokiStack CR with the object storage credentials.
4. Configure the ClusterLogging CR from OpenShift Logging to use Loki as the log store.

#### **Install the Loki Operator**

OpenShift Logging does not automatically deploy Loki. You must deploy Loki by installing the Loki operator from OperatorHub or by using the CLI. See the references section for instructions.

The Loki operator creates and manages the components of the log store.

#### **Configure Loki Object Storage**

Storing logs for your cluster might require significant disk space. Because the required disk space depends on the size of your cluster and application workloads, you might need to plan your storage solution with application developers.

### Note

This section focuses only on how to use ODF as the Loki object storage.

For more details about ODF, refer to the DO370: Enterprise Kubernetes Storage with Red Hat OpenShift Data Foundation training course.

For more information about how to configure other S3-compatible object stores, refer to <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#logging-loki-storage_installing-log-storage>

Similar to a persistent volume claim, you can use an object bucket claim to request an S3-compatible bucket. ODF creates a secret and a configuration map with the same name as the object bucket claim; between them, they provide the credentials and information to access the bucket.

The following excerpt is an example of an object bucket claim definition in the openshift-logging namespace:

apiVersion: objectbucket.io/v1alpha1

kind: ObjectBucketClaim

metadata:

name: ***loki-bucket-odf***

namespace: openshift-logging

spec:

generateBucketName: ***loki-bucket-odf***

storageClassName: openshift-storage.noobaa.io

ODF creates a configuration map with the bucket information in the openshift-logging namespace. You can retrieve one part of the credentials for the bucket from the configuration map as follows:

[user@host ~]$ **oc get configmap/*loki-bucket-odf* -o yaml**

apiVersion: v1

data:

BUCKET\_HOST: s3.openshift-storage.svc

BUCKET\_NAME: ***loki-bucket-odf-69d9...ab91***

BUCKET\_PORT: "443"

BUCKET\_REGION: ""

BUCKET\_SUBREGION: ""

kind: ConfigMap

...output omitted...

You can retrieve the other part of the bucket credentials from the secret as follows:

[user@host ~]$ **oc extract --to=- secret/loki-bucket-odf -n openshift-logging**

# AWS\_ACCESS\_KEY\_ID

***o0sOY8oaZtPfP18DRjSa***

# AWS\_SECRET\_ACCESS\_KEY

***z1QtDH37lUvLnkjpE3E4aS8yQI56CLPozGJOt31e***

After you retrieve the credentials, create a secret that contains those credentials in the openshift-logging namespace for Loki to use the credentials:

[user@host ~]$ **oc create secret generic logging-loki-odf \**

**-n openshift-logging \**

**--from-literal=access\_key\_id=${ACCESS\_KEY\_ID} \**

**--from-literal=access\_key\_secret=${SECRET\_ACCESS\_KEY} \**

**--from-literal=bucketnames=${BUCKET\_NAME} \**

**--from-literal=endpoint=https://${BUCKET\_HOST}:${BUCKET\_PORT}**

#### **Create a LokiStack Instance**

After installing the Loki operator and creating the object bucket, you create and configure the LokiStack instance with the bucket credentials.

The following example is a LokiStack resource that uses the logging-loki-odf secret that contains the bucket credentials:

apiVersion: loki.grafana.com/v1

kind: LokiStack

metadata:

name: ***logging-loki***

namespace: openshift-logging

spec:

limits:

global:

retention:

days: 20

stream:

- days: 4

priority: 1

selector: '{kubernetes\_namespace\_name=~"test.+"}'

size: 1x.demo

storage:

secret:

name: ***logging-loki-odf***

type: s3

tls:

caName: openshift-service-ca.crt

storageClassName: ***ocs-external-storagecluster-ceph-rbd***

tenants:

mode: openshift-logging

|  |  |
| --- | --- |
|  | Defines the limits that Loki applies to the log streams. See the references section for more information about the limit types for log streams. |
|  | This limit defines 20 days for log retention. |
|  | The retention policy that is based on log streams. Stream-based retention policies are useful if lowering the retention period for a log stream. For example, you might switch to debug mode temporarily in a project to troubleshoot a critical issue. However, debug logs can quickly fill your storage if the retention period is long. Then, you reduce the retention period for that project for a few days. The selector for the stream-based retention policy is specified in the LogQL query language, which is explained later in this section. |
|  | The size of the LokiStack instance. You can configure it by using sizes such as 1x.small and 1x.medium. |
|  | The name for the secret that contains the bucket credentials. |
|  | The corresponding storage type. |
|  | Loki must configure the Certificate Authority (CA) for the object storage endpoint if the object storage endpoint certificate is not trusted. For ODF, you can use the OpenShift service CA in the openshift-service-ca.crt configuration map. For other object stores, create a configuration map that contains the CA certificate. |
|  | The name of a storage class for temporary storage. You can list the storage classes for your cluster by using the oc get storageclasses command. |

### Note

Red Hat recommends setting the global retention at the object storage level for cost-efficient pruning of logs.

See the references section for more information about the lifecyle bucket configuration and the LokiStack CR.

After you create the LokiStack instance, the Loki operator starts scheduling pods for components. You can verify the installation by listing the pods in the openshift-logging project and verifying that the Loki pods are running.

#### **Configure the OpenShift Logging Log Store**

The OpenShift Logging ClusterLogging CR configures and manages deploying the OpenShift Logging components.

To configure OpenShift Logging to use Loki as the log store, you must create a cluster logging resource in the openshift-logging namespace and set lokistack as the log store type. Then, OpenShift Logging uses Loki to store the infrastructure and application logs.

The following excerpt shows a minimum configuration for OpenShift Logging to enable Loki as the log store:

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: instance

namespace: openshift-logging

spec:

managementState: Managed

logStore:

type: lokistack

lokistack:

name: logging-loki

collection:

type: vector

|  |  |
| --- | --- |
|  | The log store type. |
|  | The name for the internal LokiStack instance. |

After you create the ClusterLogging instance, the OpenShift Logging operator starts scheduling pods for components. You can verify the installation by listing the pods in the openshift-logging project and verifying that the collector pods are running.

### **Audit Logs**

By default, the OpenShift Logging operator includes logs for the infrastructure and the application. However, the Logging operator does not include audit logs, because these logs might contain sensitive security details.

To include audit logs in the log store, you must create a ClusterLogForwarder CR to configure the log collector to collect them.

The following excerpt shows a minimum configuration to configure the ClusterLogForwarder CR to forward the audit logs together with the infrastructure and application logs to the internal Loki instance:

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: instance

namespace: openshift-logging

spec:

pipelines:

- name: all-to-default

inputRefs:

- infrastructure

- application

- audit

outputRefs:

- default

|  |  |
| --- | --- |
|  | The type of logs to collect, which include infrastructure, audit, and application logs. |
|  | The destination for the logs, with the internal Loki instance being the default output. |

### **View, Search, and Query Logs**

OpenShift Logging provides a native OpenShift Console plug-in to view, search, and query logs in the internal log store.

To use the OpenShift web console for logging visualization, first enable the OpenShift Logging console plug-in. You can use the **Operators** → **Installed Operators** menu, by selecting the Red Hat OpenShift Logging operator, and ensuring that the console plug-in is enabled as in the following image:

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

Then, create a cluster logging resource in the openshift-logging namespace and set ocp-console as the visualization type. The following excerpt shows a minimum configuration for OpenShift Logging to enable the OpenShift web console for visualization:

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: instance

namespace: openshift-logging

spec:

...output omitted...

**visualization:**

**type: ocp-console**

After you enable the web console for visualization, you can view your logs in the **Observe** → **Logs** menu. The following image shows the **Logs** menu and the filters that you can apply to your logs by using the web console:

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

|  |  |
| --- | --- |
|  | Time range for the logs. |
|  | Filter the logs by content, namespace, pod, or container. |
|  | Log severity. Possible values are critical, error, warning, debug, info, trace, and unknown. |
|  | Log type. Possible values are application, infrastructure, or audit. |
|  | The log query in LogQL query language. You can show this field by clicking **Show Query**. |

#### **LogQL Queries**

Loki organizes logs into streams that are called chunks. A chunk is a sequence of log entries that use the same set of labels. Labels are key-value pairs that help identify and filter log entries. Labels can include information such as the application name, environment, severity, or any other relevant metadata. Loki uses LogQL as the query language for filtering and searching logs with labels and other criteria.

By using LogQL in the OpenShift web console, you can extract and filter specific information from the logs.

A LogQL query consists of the following parameters:

* The log stream selector, which accepts the following operators: = equals, != not equals, =~ regex pattern matches, and !~ regex pattern does not match.
* A filter expression, which accepts the following operators: |= equals, != not equals, |~ regex pattern matches, and !~ regex pattern does not match.

When using LogQL in OpenShift Logging, you must always use at least the log\_type log stream selector for the log type.

For example, the following LogQL query shows the infrastructure logs:

log\_type="infrastructure" | json

Adding the | json string to your query extracts all JSON properties as labels for ease of viewing. After you type the LogQL query, click **Run Query** to apply it to your logs.

By using LogQL you can filter your logs with more advanced expressions. For example, the following LogQL query filters the audit logs only from audit.log files in any directory:

log\_type="audit" | json | file=~".\*audit.log"

You can use multiple log stream selectors and filter expressions to increase filter granularity. The following example shows a LogQL query for the audit logs that contain the error string but not the warning string, and that return a 403 or 503 status.

log\_type="audit" | json |= "error" != "warning" |~ "status [45]03"

### Note

For more information about querying Loki by using the LogQL query language, refer to <https://grafana.com/docs/loki/latest/logql/>

Vector adds OpenShift metadata to pod logs to provide fields for querying and filtering. You can use these fields in your LogQL queries for more precise filtering.

Some fields that you can use in your queries include the following strings:

**hostname**

The hostname of the OpenShift node for the pod that generates the log.

**kubernetes\_container\_name**

The name of the container that generates the log.

**kubernetes\_namespace\_name**

The name of the project that contains the pod that generates the log.

**kubernetes\_pod\_name**

The name of the pod that generates the log.

**level**

The log level.

**message**

The log message.

### **Log Access Permissions**

By default, the OpenShift Logging operator grants only administrator users access to the logs. Thus, regular users cannot access the logs. As an administrator, you can configure access to the logs for users and groups by using RBAC rules.

The OpenShift Logging operator provides the following cluster roles to simplify RBAC rules:

**cluster-logging-application-view**

Grants read permission to application logs.

**cluster-logging-infrastructure-view**

Grants read permission to infrastructure logs.

**cluster-logging-audit-view**

Grants read permission to audit logs.

### References

For more information about logging on OpenShift, refer to the About Logging section in the Red Hat OpenShift Container Platform 4.14 Logging documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#cluster-logging>

For more information about installing the Loki operator, refer to the Installing Log Storage section in the Red Hat OpenShift Container Platform 4.14 Logging documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#installing-log-storage>

For more information about Loki limits for log streams, refer to the LimitsSpec section in the Loki Operator API documentation at <https://loki-operator.dev/docs/api.md/#loki-grafana-com-v1-LimitsSpec>

[Setting a Lifecycle Configuration on a Bucket](https://docs.aws.amazon.com/AmazonS3/latest/userguide/how-to-set-lifecycle-configuration-intro.html)

For more information about lifecyle bucket configuration, refer to the Lifecyle Bucket Configuration in Multicloud Object Gateway section in the Red Hat OpenShift Data Foundation 4.14 Managing Hybrid and Multicloud Resources documentation at <https://access.redhat.com/documentation/en-us/red_hat_openshift_data_foundation/4.14/html-single/managing_hybrid_and_multicloud_resources/index#con_lifecycle-bucket-configuration-in-multicloud-object-gateway_rhodf>

[Knowledgebase — How to Configure Lifecycle Policy in MCG](https://access.redhat.com/solutions/7058004)

[LokiStack](https://loki-operator.dev/docs/api.md/#loki-grafana-com-v1-LokiStack)

For more information about the Loki deployment size, refer to the Installing Log Storage section in the Red Hat OpenShift Container Platform 4.14 Log Storage documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#loki-deployment-sizing_installing-log-storage>

For more information about the secret type values, refer to the Loki Object Storage section in the Red Hat OpenShift Container Platform 4.14 Log Storage documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index#logging-loki-storage_installing-log-storage>

## **Guided Exercise: Centralized Logging**

Deploy OpenShift Logging for short-term log retention and aggregation.

**Outcomes**

* Use Loki as the log store for OpenShift Logging.
* Use Vector as the collector and the OpenShift web UI for log visualization.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start logging-central**

**Instructions**

Your company requires you to configure the deployed OpenShift Logging operator for short-term log retention and aggregation. You must configure OpenShift Logging by using Vector, Loki, and the OpenShift web console. Moreover, you must apply the cluster-logging-application-view cluster role to the ocpdevs group, so the developer user can retrieve application logs for the testing-logs project.

1. As the OpenShift admin user, install the Loki operator in the openshift-operators-redhat namespace.
   1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com.
   2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.
   3. Navigate to **Operators** → **OperatorHub**.
   4. Click **Loki Operator**, and then click **Install**.
   5. Select **Enable Operator recommended cluster monitoring on this Namespace** and click **Install**. The Operator Lifecycle Manager (OLM) can take a few minutes to install the operator. Click **View Operator** to navigate to the operator details.
2. Create an S3-compatible object storage bucket with OpenShift Data Foundation for the Loki operator. Retrieve the credentials for the bucket, and create a secret with those credentials.
   1. From the terminal, log in to your OpenShift cluster as the admin user.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

...output omitted...

* 1. Change to the ~/DO380/labs/logging-central directory.

[student@workstation ~]$ **cd ~/DO380/labs/logging-central**

* 1. Create an ObjectBucketClaim resource YAML file for a bucket called loki-bucket-odf in the openshift-logging namespace. The Loki operator uses this bucket. You can find an incomplete example for the resource in the ~/DO380/labs/logging-central/objectbucket.yaml file.

apiVersion: objectbucket.io/v1alpha1

kind: ObjectBucketClaim

metadata:

name: **loki-bucket-odf**

namespace: **openshift-logging**

spec:

generateBucketName: **loki-bucket-odf**

storageClassName: openshift-storage.noobaa.io

* 1. Create the ObjectBucketClaim resource.

[student@workstation logging-central]$ **oc create -f objectbucket.yaml**

objectbucketclaim.objectbucket.io/loki-bucket-odf created

* 1. Verify that the object bucket claim is created and in the Bound phase.

[student@workstation logging-central]$ **oc get obc -n openshift-logging**

NAME STORAGE-CLASS PHASE AGE

loki-bucket-odf openshift-storage.noobaa.io Bound 12m

* 1. Retrieve the S3 bucket information and credentials, and store them in environment variables. When an object bucket claim is created, OpenShift Data Foundation creates a secret and a configuration map with the same name as for the bucket information and credentials. The bucket credentials would differ on your system.

[student@workstation logging-central]$ **BUCKET\_HOST=$(oc get -n openshift-logging \**

**configmap loki-bucket-odf -o jsonpath='{.data.BUCKET\_HOST}'); \**

**BUCKET\_NAME=$(oc get -n openshift-logging configmap loki-bucket-odf \**

**-o jsonpath='{.data.BUCKET\_NAME}'); \**

**BUCKET\_PORT=$(oc get -n openshift-logging configmap loki-bucket-odf \**

**-o jsonpath='{.data.BUCKET\_PORT}'); \**

**ACCESS\_KEY\_ID=$(oc get -n openshift-logging secret loki-bucket-odf \**

**-o jsonpath='{.data.AWS\_ACCESS\_KEY\_ID}' | base64 -d); \**

**SECRET\_ACCESS\_KEY=$(oc get -n openshift-logging secret loki-bucket-odf \**

**-o jsonpath='{.data.AWS\_SECRET\_ACCESS\_KEY}' | base64 -d)**

* 1. Create a secret called logging-loki-odf in the openshift-logging namespace with the bucket credentials.

[student@workstation logging-central]$ **oc create secret generic logging-loki-odf \**

**-n openshift-logging \**

**--from-literal=access\_key\_id=${ACCESS\_KEY\_ID} \**

**--from-literal=access\_key\_secret=${SECRET\_ACCESS\_KEY} \**

**--from-literal=bucketnames=${BUCKET\_NAME} \**

**--from-literal=endpoint=https://${BUCKET\_HOST}:${BUCKET\_PORT}**

secret/logging-loki-odf created

1. Create a LokiStack instance by using the bucket as the storage.
   1. Create a LokiStack resource YAML file for an instance called logging-loki in the openshift-logging namespace. This instance uses the bucket as the storage. You can find an incomplete example for the resource in the ~/DO380/labs/logging-central/lokistack.yaml file.

apiVersion: loki.grafana.com/v1

kind: LokiStack

metadata:

name: **logging-loki**

namespace: openshift-logging

spec:

size: 1x.demo

storage:

secret:

name: **logging-loki-odf**

type: s3

tls:

caName: openshift-service-ca.crt

storageClassName: ocs-external-storagecluster-ceph-rbd

tenants:

mode: openshift-logging

* 1. Create the LokiStack resource.

[student@workstation logging-central]$ **oc create -f lokistack.yaml**

lokistack.loki.grafana.com/logging-loki created

* 1. Verify that the LokiStack pods are up and running.

[student@workstation logging-central]$ **oc get pods -n openshift-logging**

NAME READY STATUS RESTARTS AGE

cluster-logging-operator-554849f7dd-9tcz2 1/1 Running 0 20m

logging-loki-compactor-0 1/1 Running 0 98s

logging-loki-distributor-64c798c4c5-6wpxp 1/1 Running 0 99s

logging-loki-gateway-68fb59cdf5-6b8mt 2/2 Running 0 98s

logging-loki-gateway-68fb59cdf5-7qf4s 2/2 Running 0 98s

logging-loki-index-gateway-0 1/1 Running 0 98s

logging-loki-ingester-0 1/1 Running 0 99s

logging-loki-querier-577b55f8d5-f4cfb 1/1 Running 0 98s

logging-loki-query-frontend-775755684d-f94bd 1/1 Running 0 98s

1. Create a ClusterLogging instance by using loki as the log store, vector as the collector, and the ocp-console as the visualization console.
   1. Create a ClusterLogging resource YAML file by using loki as the log store, vector as the collector, and the ocp-console as the visualization console. You can find an incomplete example for the resource in the ~/DO380/labs/logging-central/clusterlogging.yaml file.

apiVersion: logging.openshift.io/v1

kind: ClusterLogging

metadata:

name: instance

namespace: openshift-logging

spec:

managementState: Managed

logStore:

type: **lokistack**

lokistack:

name: logging-loki

collection:

type: **vector**

visualization:

type: **ocp-console**

* 1. Create the ClusterLogging resource.

[student@workstation logging-central]$ **oc create -f clusterlogging.yaml**

clusterlogging.logging.openshift.io/instance created

* 1. Verify that the ClusterLogging pods are up and running.

[student@workstation logging-central]$ **oc get pods -n openshift-logging**

NAME READY STATUS RESTARTS AGE

cluster-logging-operator-554849f7dd-9tcz2 1/1 Running 0 24m

collector-7rb2n 1/1 Running 0 93s

collector-bkj8x 1/1 Running 0 93s

collector-cng5z 1/1 Running 0 93s

collector-hx2gd 1/1 Running 0 93s

collector-x92zq 1/1 Running 0 93s

collector-xlqw9 1/1 Running 0 93s

...output omitted...

logging-view-plugin-5b9b5b7bdc-tvkqk 1/1 Running 0 94s

1. Enable the console plug-in for the OpenShift Logging operator. Verify that you have access to the logs.
   1. Change to the web console browser. Click **Operators** → **Installed Operators**, and select **All Projects** from the drop-down menu.
   2. Click **Red Hat OpenShift Logging**, click **Console plugin**, select **Enable**, and click **Save**.
   3. Reload the web console, and navigate to **Observe** → **Logs**. If the **Observe** → **Logs** menu is not available, then wait until the web console shows the Web console update is available message and reload the web console. You have access to the logs for the application and infrastructure resources. By default, the ClusterLogging instance includes logs for the application and infrastructure, but not the audit logs. Observe the application logs, which are selected by default.
   4. From the drop-down menu, select the infrastructure logs and observe them.

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

* 1. From the drop-down menu, select the audit logs and observe the No datapoints found message. You receive this message because the ClusterLogging instance does not forward the audit logs.

1. Include the audit logs by creating a log forwarder for the application, infrastructure, and audit logs to the LokiStack resource.
   1. Change to the terminal window, and create an ClusterLogForwarder resource YAML file for a log forwarder called instance in the openshift-logging namespace. The log forwarder must forward the application, infrastructure, and audit logs. You can find an incomplete example for the resource in the ~/DO380/labs/logging-central/forwarder.yaml file.

apiVersion: logging.openshift.io/v1

kind: ClusterLogForwarder

metadata:

name: instance

namespace: openshift-logging

spec:

pipelines:

- name: all-to-default

inputRefs:

- **infrastructure**

- **application**

- **audit**

outputRefs:

- **default**

* 1. Create the ClusterLogForwarder resource.
  2. [student@workstation logging-central]$ **oc create -f forwarder.yaml**

clusterlogforwarder.logging.openshift.io/instance created

* 1. Change to the web console browser and reload it. You have access to the audit logs.

1. Verify that the infrastructure and audit logs are sent to Loki, by trying to open an SSH connection to one of the compute nodes.
   1. Change to the terminal window. Try to open an SSH connection to the worker01.ocp4.example.com node, which is rejected.

[student@workstation logging-central]$ **ssh worker01.ocp4.example.com**

Warning: Permanently added 'worker01.ocp4.example.com' (ED25519) to the list of known hosts.

student@worker01.ocp4.example.com: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).

* 1. Change to the web console browser, and click the refresh button in the upper-right corner to update the logs.
  2. Ensure that the audit logs are selected in the drop-down menu. Search for SSH connection messages by typing ssh in the Search by Content field and clicking **Run Query**.
  3. Unfold the information for the first audit log, which shows an attempt to log to the worker01 node with a failed result.
  4. From the drop-down menu, select the infrastructure logs. Click **Clear all filters** and click **Show Query**.
  5. Modify the query to filter logs only from the sshd service. The query should read as follows:

{ log\_type="infrastructure" } | json | systemd\_u\_SYSLOG\_IDENTIFIER="sshd"

* 1. Click **Run Query** and verify that you receive logs from the sshd service.

1. Create a project and a pod as the developer user and verify that this user has access to the pod logs.
   1. Change to the terminal window and log in to the OpenShift cluster as the developer user with developer as the password.

[student@workstation logging-central]$ **oc login -u developer -p developer**

Login successful.

...output omitted...

* 1. Create the testing-logs project.

[student@workstation logging-central]$ **oc new-project testing-logs**

Now using project "testing-logs" on server "https://api.ocp4.example.com:6443".

...output omitted...

* 1. Create a test-date pod.

[student@workstation logging-central]$ **oc run test-date --restart 'Never' \**

**--image registry.ocp4.example.com:8443/ubi9/ubi -- date**

pod/test-date created

* 1. Verify that the pod is in the Completed status.

[student@workstation logging-central]$ **oc get pods**

NAME READY STATUS RESTARTS AGE

test-date 0/1 Completed 0 56s

* 1. Review the logs for the pod by using the terminal.

[student@workstation logging-central]$ **oc logs test-date**

Tue Jan 16 11:15:15 UTC 2024

* 1. Delete the pod.

[student@workstation logging-central]$ **oc delete pod test-date**

pod "test-date" deleted

* 1. Try to retrieve the logs for the pod, and verify that the developer user no longer has access to the pod logs.

[student@workstation logging-central]$ **oc logs test-date**

Error from server (NotFound): pods "test-date" not found

1. Log in as the developer user and verify that the user has no access to the logs.
   1. Change to the browser window, open a private window, and navigate to https://console-openshift-console.apps.ocp4.example.com.
   2. Click **Red Hat Identity Management** and log in as the developer user with developer as the password. Click **Skip tour**.
   3. Navigate to **Observe**. Verify that the **testing-logs** project is selected. Change to the **Logs** tab. The developer user has no permissions to retrieve the pod logs.

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

1. Review the test-date pod logs by using the OpenShift logging operator. The admin user has access to the stored pod logs.
   1. Change to the web console browser where the admin user is logged in. From the drop-down menu where the infrastructure logs are currently selected, select the application logs. Click the refresh button in the upper-right corner to update the logs.
   2. Modify the query to filter the results for the testing-logs namespace. The query should read as follows:

{ log\_type="application", kubernetes\_namespace\_name="testing-logs" } | json

* 1. Click **Run Query**. Retrieve the information for the only entry, which shows the logs from the deleted pod.

1. Give the ocpdevs group permission to view the logs in the test-logs project, and verify that the developer user has access to the logs. Give the permission to the ocpdevs group by assigning it the cluster-logging-application-view cluster role.
   1. Change to the terminal window and log in to the OpenShift cluster as the admin user with redhatocp as the password.

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

Login successful.

...output omitted...

* 1. Review the required role to provide access to the application logs to the ocpdevs group. You can find an example in the ~/DO380/labs/logging-central/ocpdevs-role.yaml file.

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: view-application-logs

namespace: testing-logs

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-logging-application-view

subjects:

- kind: Group

name: ocpdevs

apiGroup: rbac.authorization.k8s.io

* 1. Apply the role to the ocpdevs group.

[student@workstation logging-central]$ **oc create -f ocpdevs-role.yaml**

rolebinding.rbac.authorization.k8s.io/view-application-logs created

* 1. Change to the web console in the private browser where the developer user is logged in, and refresh it.
  2. Verify that the developer user has access to the deleted pod logs.
  3. Close both the web browser windows and change to the student HOME directory in the terminal window.

[student@workstation logging-central]$ **cd**

**Finish**

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

[student@workstation ~]$ **lab finish logging-central**

Lab: OpenShift Logging

Configure OpenShift Logging for short-term and long-term log retention and aggregation.

**Outcomes**

* Configure OpenShift Logging to forward logs to an external aggregator for long-term storage.
* Configure OpenShift Logging with Loki for short-term log retention.
* Configure OpenShift Logging to collect logs from specific applications.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

**Instructions**

The security policy of your company requires you to send OpenShift audit logs to a dedicated syslog server for long-term storage.

The developer team asks you to provide them access to the production application logs and to the CI job logs that are running in the build-ci namespace. To limit the footprint of the log storage, the developers agree to keep only seven days of logs for the build-ci namespace.

Configure OpenShift Logging to forward only audit logs to the syslog server. The syslog server DNS name is utility.lab.example.com and the service listens on the TCP port 514. Forwarded audit logs are stored in the /var/log/openshift/audit.log file on the utility machine.

Configure Loki as the internal log store for both infrastructure logs for the cluster administrators and application logs for the developers.

An S3 bucket is available for you, in the lab environment, to configure as log storage for Loki. The bucket information and credentials are available in the ~/DO380/labs/logging-review/s3bucket.env file on the workstation machine.

Ensure that only application logs with the environment=production label or in the build-ci namespace are collected. Ensure that audit logs are not stored in the internal log store.

Configure the global log retention in Loki to 30 days. Configure the log retention for pods that are running in the build-ci namespace to seven days.

Provide access to the application logs to the ocpdevs group. You can use the developer user, which is in the ocpdevs group, to verify that the permissions are correct.

Use the following applications in the cluster to verify that the logging configuration is working as expected:

| **Label** | **Projects** |
| --- | --- |
| environment=production | * shop-prod * supportbot-prod |
| environment=staging | * shop-stg * supportbot-stg |
| environment=development | * shop-dev * supportbot-dev * build-ci |

1. Create a secret for Loki with the object storage credentials from the ~/DO380/labs/logging-review/s3bucket.env environment file.
   1. Connect to the OpenShift cluster as the admin user with redhatocp as the password.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

*...output omitted...*

* 1. Change to the ~/DO380/labs/logging-review directory.

[student@workstation ~]$ **cd ~/DO380/labs/logging-review**

* 1. Change to the openshift-logging project.
  2. [student@workstation logging-review]$ **oc project openshift-logging**

Now using project "openshift-logging" on server "https://api.ocp4.example.com:6443".

* 1. Use the ~/DO380/labs/logging-review/s3bucket.env environment file to create the logging-loki-odf secret in the openshift-logging namespace.
  2. [student@workstation logging-review]$ **oc create secret generic logging-loki-odf \**
  3. **-n openshift-logging --from-env-file=s3bucket.env**

secret/logging-loki-odf created

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Create and configure a logging-loki LokiStack instance to use the S3 bucket, and set the log retention for the build-ci namespace.

You can use the partial resource definition in the ~/DO380/labs/logging-review/lokistack.yml file.

* 1. Modify the partial resource definition in the ~/DO380/labs/logging-review/lokistack.yml file as follows:
  2. apiVersion: loki.grafana.com/v1
  3. kind: LokiStack
  4. metadata:
  5. name: **logging-loki**
  6. namespace: **openshift-logging**
  7. spec:
  8. limits:
  9. global:
  10. retention:
  11. days: 30
  12. streams:
  13. - selector: **'{kubernetes\_namespace\_name="build-ci"}'**
  14. priority: 1
  15. days: **7**
  16. size: 1x.demo
  17. storage:
  18. tls:
  19. caName: openshift-service-ca.crt
  20. secret:
  21. name: **logging-loki-odf**
  22. type: s3
  23. storageClassName: ocs-external-storagecluster-ceph-rbd
  24. tenants:

mode: **openshift-logging**

* 1. Create the LokiStack resource.
  2. [student@workstation logging-review]$ **oc create -f lokistack.yml**

lokistack.loki.grafana.com/logging-loki created

* 1. Verify that the LokiStack pods are up and running.
  2. [student@workstation logging-review]$ **oc get deployment,statefulset \**
  3. **-l app.kubernetes.io/name=lokistack**
  4. NAME READY UP-TO-DATE AVAILABLE AGE
  5. deployment.apps/logging-loki-distributor **1/1** 1 1 86s
  6. deployment.apps/logging-loki-gateway **2/2** 2 2 85s
  7. deployment.apps/logging-loki-querier **1/1** 1 1 85s
  8. deployment.apps/logging-loki-query-frontend **1/1** 1 1 85s
  9. NAME READY AGE
  10. statefulset.apps/logging-loki-compactor **1/1** 85s
  11. statefulset.apps/logging-loki-index-gateway **1/1** 85s

statefulset.apps/logging-loki-ingester **1/1** 86s

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

1. Configure OpenShift Logging to forward audit logs to the syslog server, and to forward the infrastructure and application logs to the internal log store.

Ensure that only application logs with the environment: production label or in the build-ci namespace are collected. Ensure that audit logs are not stored in the Loki instance.

You can use the partial resource definition in the ~/DO380/labs/logging-review/clusterlogforwarder.yml file.

* 1. Modify the partial resource definition in the ~/DO380/labs/logging-review/clusterlogforwarder.yml file as follows:
  2. apiVersion: logging.openshift.io/v1
  3. kind: ClusterLogForwarder
  4. metadata:
  5. name: **instance**
  6. namespace: **openshift-logging**
  7. spec:
  8. inputs:
  9. - name: production-apps
  10. application:
  11. **selector:**
  12. **matchLabels:**
  13. **environment: production**
  14. - name: ci
  15. application:
  16. **namespaces:**
  17. - **build-ci**
  18. outputs:
  19. - name: audit-syslog
  20. type: **syslog**
  21. url: **tcp://utility.lab.example.com:514**
  22. syslog:
  23. msgID: audit
  24. appName: ocp-lab
  25. facility: user
  26. procID: vector
  27. rfc: RFC5424
  28. severity: informational
  29. pipelines:
  30. - name: to-syslog
  31. inputRefs:
  32. - **audit**
  33. outputRefs:
  34. - **audit-syslog**
  35. - name: to-loki
  36. inputRefs:
  37. - **infrastructure**
  38. - **ci**
  39. - **production-apps**
  40. outputRefs:

- **default**

* 1. Apply the configuration for the cluster log forwarder.
  2. [student@workstation logging-review]$ **oc apply -f clusterlogforwarder.yml**

clusterlogforwarder.logging.openshift.io/instance created

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

1. Configure OpenShift Logging to deploy Vector and the web console plug-in, and use the Loki instance that you configure in a previous step as the log store.

You can use the partial resource definition in the ~/DO380/labs/logging-review/clusterlogging.yml file.

* 1. Modify the partial resource definition in the ~/DO380/labs/logging-review/clusterlogging.yml file as follows:
  2. apiVersion: logging.openshift.io/v1
  3. kind: ClusterLogging
  4. metadata:
  5. name: **instance**
  6. namespace: **openshift-logging**
  7. spec:
  8. managementState: Managed
  9. logStore:
  10. type: **lokistack**
  11. lokistack:
  12. name: **logging-loki**
  13. collection:
  14. type: **vector**
  15. visualization:

type: **ocp-console**

* 1. Apply the cluster logging configuration.
  2. [student@workstation logging-review]$ **oc apply -f clusterlogging.yml**

clusterlogging.logging.openshift.io/instance created

* 1. Verify that the OpenShift Logging operator deploys the collector pod on each node.
  2. [student@workstation logging-review]$ **oc get daemonset -l component=collector**
  3. NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE ...

collector **6** 6 **6** 6 6 ...

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

1. Connect to the utility server and verify that the syslog service receives the audit logs. Audit logs are stored in the /var/log/openshift/audit.log file on the utility machine.
   1. Connect to the utility machine with SSH as the root user.

[student@workstation logging-review]$ **ssh root@utility**

* 1. Check that the OpenShift audit log file exists in the /var/log/openshift path.
  2. [root@utility ~]# **ls -l /var/log/openshift**
  3. total 978112

-rw-------. 1 root root 462107806 Jan 30 10:10 audit.log

* 1. Review the content of the audit log.
  2. [root@utility ~]# **tail -1 /var/log/openshift/audit.log**

"2024-01-30T10:10:25.618+00:00 *...output omitted...*"

* 1. Disconnect from the utility machine.
  2. [root@utility ~]# **exit**
  3. logout
  4. Connection to utility closed.

[student@workstation logging-review]$

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Enable the web console plug-in for the OpenShift Logging operator and verify that infrastructure and application logs are available.
   1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.
   2. Click **Operators** → **Installed Operators**, and select All Projects from the drop-down menu.
   3. Click **Red Hat OpenShift Logging**, click **Console plugin**, select **Enable**, and click **Save**.
   4. Reload the web console, and navigate to **Observe** → **Logs**. If the **Observe** → **Logs** menu is not available, then wait until the web console shows the Web console update is available message and reload the web console.

Verify that the application logs are available.

* 1. From the drop-down menu, select infrastructure and verify that the infrastructure logs are available.

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

* 1. From the drop-down menu, select audit and confirm that no audit logs are stored in the log store.

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

1. Grant view access to the application logs to the ocpdevs group.
   1. Add the cluster-logging-application-view role to the ocpdevs group.
   2. [student@workstation logging-review]$ **oc adm policy add-cluster-role-to-group \**
   3. **cluster-logging-application-view ocpdevs**

clusterrole.rbac.authorization.k8s.io/cluster-logging-application-view added: "ocpdevs"

1. [Hide Solution](https://rol.redhat.com/rol/app/)
2. Verify that the developer user can access the application logs from the web console.

Verify that only application logs with the environment: production label or in the build-ci namespace are collected.

* 1. Open a new private browser window, and navigate to https://console-openshift-console.apps.ocp4.example.com.
  2. Click **Red Hat Identity Management** and log in as the developer user with developer as the password. Click **Skip tour**.
  3. Navigate to **Observe** and select build-ci from the project drop-down menu. Change to the **Logs** tab. Verify that the application logs are available.
  4. Change to the shop-prod project and verify that the application logs are available.
  5. Change to the shop-dev project, and verify that no application logs are available, because the application does not have the required label.
  6. Close both the web browser windows and change to the student HOME directory. in the terminal window.

[student@workstation logging-review]$ **cd**

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

**Evaluation**

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

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

## **Lab: Cluster Administration**

Use Red Hat OpenShift GitOps for cluster administration.

Configure an OIDC identity provider.

Configure a one-time backup with OADP and restore from it.

Configure OpenShift Logging for short-term log retention and aggregation.

Configure alert forwarding and inspect alerts.

**Outcomes**

* Configure Red Hat Single Sign-On (SSO) as an OIDC identity provider (IdP) for OpenShift by using Red Hat OpenShift GitOps.
* Deploy the OpenShift Logging operator and configure it to use Loki as the log store, Vector as the collector, and the OpenShift web UI for log visualization.
* Add permission for a user to read the logs from a project.
* Back up an application and restore the application to a different namespace.
* Configure alert forwarding.
* Use monitoring to identify an application problem.

As the student user on the workstation machine, use the lab command to prepare your environment for this exercise, and to ensure that all required resources are available.

[student@workstation ~]$ **lab start compreview-review1**

**Specifications**

* Use GitOps to configure the cluster to use Red Hat SSO for authentication. Red Hat SSO provides the filipmansur user, with the redhat\_sso password, in the ocp\_rhsso client. This user is part of the etherpad-devs group.

Use the following parameters to configure Red Hat SSO in OpenShift:

| **Attribute** | **Value** |
| --- | --- |
| Name | RHSSO\_OIDC |
| Client ID | ocp\_rhsso |
| Client secret | QGEP6zoLo6BUGbib5oCkGwtZ8EAlmMgW |

* + To use OpenShift GitOps, you can use the admin OpenShift user with redhatocp as the password. The admin user is part of the ocpadmins, so grant administrator rights in Argo CD to users the ocpadmins group. The operator adds a link to the default instance in the application menu of the OpenShift console.
  + The classroom has a GitLab instance that you can use to create any necessary Git repositories. GitLab is available at the https://git.ocp4.example.com URL. You can use the developer user with d3v3lop3r as the password. The lab scripts expect a compreview-review1 repository for cleanup.

The lab scripts configure the username, email, and authentication for Git in the workstation machine.

* + Argo CD accesses only trusted repositories. GitLab uses a certificate that is signed by the classroom CA. This CA is included in the certificates that are trusted by the cluster. You can use the config.openshift.io/inject-trusted-cabundle label to inject the cluster trusted certificates into a configuration map, and then configure Argo CD to trust the certificate. The injected certificate is in the ca-bundle.crt file in the configuration map, and Argo CD uses the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem for trusted certificates in the repository server container.
  + The ~/DO380/labs/compreview-review1/sso\_config.yml file contains an incomplete authentication configuration.
* The lab scripts deploy Etherpad in the compreview-review1 namespace. The Etherpad resources have the app.kubernetes.io/name label with etherpad as the value, and the supporting database resources have the app.kubernetes.io/name label with mariadb as the value.
  + Create an etherpad-backup backup schedule. The ~/DO380/labs/compreview-review1/schedule-db-backup.yml file contains an example to create the schedule.
  + You can define an alias to access the velero binary by using the following command:
  + [user@host ~]$ **alias velero='\**

**oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'**

* + Trigger an immediate backup from the schedule, and restore the backup to the etherpad-test-restore namespace.
* Configure the deployed OpenShift Logging operator for short-term log retention and aggregation. The Loki operator is deployed in the cluster.
  + An S3 bucket is available for you, in the lab environment, to configure as log storage for Loki. The bucket information and credentials are available in the ~/DO380/labs/compreview-review1/s3bucket.env file on the workstation machine.
  + Create a LokiStack instance with the 1x.demo size. Use logging-loki as the name for the LokiStack resource. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/lokistack.yaml file.
  + Use Loki as the log store, Vector as the collector, and the OpenShift web UI for log visualization. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/clusterlogging.yaml file.
  + Configure the Logging operator to include the audit logs, by using the ClusterLogForwarder resource. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/forwarder.yaml file.
  + Users in the etherpad-devs group have read and write permissions on the compreview-review1 namespace. Apply the necessary permissions so users in that group also have access to the application logs for that namespace. Red Hat SSO provides the filipmansur user from the etherpad-devs group. You can find an incomplete example for the role binding in the ~/DO380/labs/compreview-review1/logging/group-role.yaml file.
* Configure monitoring to send alerts by using a webhook.
  + The lab scripts deploy a webhook debugger service to the utility machine. This debugger starts a web server on port 8000 that prints the received payloads in the /home/student/persistent\_alert file in the utility machine.
  + OpenShift can send webhooks to this debugger at the http://utility.lab.example.com:8000 URL.
  + This exercise generates alerts with alertname labels that start with the Persistent text. You can use the alertname=~Persistent.\* regular expression filter so that the debugger prints only alerts for this exercise.
  + You can reduce the group and repeat intervals of the Alertmanager configuration to receive alerts sooner.
  + The debugger registers successful receipt of alerts for grading.
* Both the original Etherpad deployment and the restore deployment trigger critical monitoring alerts. Review the two critical alerts that the Etherpad deployments fire, and solve them.

1. Install the OpenShift GitOps operator.
   1. Open a web browser and navigate to https://console-openshift-console.apps.ocp4.example.com.
   2. Click **Red Hat Identity Management** and log in as the admin user with redhatocp as the password.
   3. Navigate to **Operators** → **OperatorHub**.
   4. Click **Red Hat OpenShift GitOps**, and then click **Install**.
   5. Review the default configuration and click **Install**. The OLM can take a few minutes to install the operator. Click **View Operator** to navigate to the operator details.
2. Configure the default Argo CD instance to trust the classroom certificate to access repositories. Argo CD accesses only trusted repositories. You can use the config.openshift.io/inject-trusted-cabundle label to inject the classroom certificate into a configuration map, and then configure Argo CD to trust the certificate. The injected certificate is in the ca-bundle.crt file in the configuration map, and Argo CD uses the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path for trusted certificates in the repository server container.

Grant administrator rights to users in the ocpadmins group.

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

...output omitted...

* 1. Create a cluster-root-ca-bundle configuration map in the openshift-gitops namespace.
  2. [student@workstation ~]$ **oc create configmap -n openshift-gitops \**

**cluster-root-ca-bundle**

* 1. Add the config.openshift.io/inject-trusted-cabundle label to the configuration map with the true value. OpenShift injects the cluster certificates into a configuration map with this label. This bundle contains the signing certificate for the classroom GitLab instance.
  2. [student@workstation ~]$ **oc label configmap -n openshift-gitops \**
  3. **cluster-root-ca-bundle config.openshift.io/inject-trusted-cabundle=true**

configmap/cluster-root-ca-bundle labeled

* 1. Edit the Argo CD default instance to inject the certificates.

You can use the following command to edit the resource:

[student@workstation ~]$ **oc edit argocd -n openshift-gitops openshift-gitops**

Edit the resource to mount the ca-bundle.crt file from the configuration map in the /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem path of the repository server container.

...output omitted...

spec:

...output omitted...

repo:

resources:

limits:

cpu: "1"

memory: 1Gi

requests:

cpu: 250m

memory: 256Mi

**volumeMounts:**

**- mountPath: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**

**name: cluster-root-ca-bundle**

**subPath: ca-bundle.crt**

**volumes:**

**- configMap:**

**name: cluster-root-ca-bundle**

**name: cluster-root-ca-bundle**

resourceExclusions: |

...output omitted...

* 1. Edit the Argo CD default instance to grant administrator rights to the ocpadmins group.
  2. ...output omitted...
  3. spec:
  4. ...output omitted...
  5. prometheus:
  6. enabled: false
  7. ingress:
  8. enabled: false
  9. route:
  10. enabled: false
  11. rbac:
  12. defaultPolicy: ""
  13. policy: |
  14. g, system:cluster-admins, role:admin
  15. g, cluster-admins, role:admin
  16. **g, ocpadmins, role:admin**
  17. scopes: '[groups]'
  18. redis:

...output omitted...

1. Create a compreview-review1 public repository for the authentication configuration in the classroom GitLab at https://git.ocp4.example.com. Use the developer GitLab user with d3v3lop3r as the password.
   1. Open a web browser and navigate to https://git.ocp4.example.com. Log in as the developer user with d3v3lop3r as the password.
   2. Click **New project**, and then click **Create blank project**. Use compreview-review1 as the project slug (repository name), select the **Public** visibility level, and use the default values for all other fields. Click **Create project**.
2. Populate the repository with the OAuth CR file so OpenShift synchronizes users from the Red Hat SSO OIDC client.
   1. Click **Clone**, and then copy the https://git.ocp4.example.com/developer/compreview-review1.git URL.
   2. In the terminal window, change to the ~/DO380/labs/compreview-review1 directory.

[student@workstation ~]$ **cd ~/DO380/labs/compreview-review1**

* 1. In a terminal, run the following command to clone the new repository.
  2. [student@workstation compreview-review1]$ **git clone \**
  3. **https://git.ocp4.example.com/developer/compreview-review1.git**
  4. Cloning into 'compreview-review1'...

...output omitted...

* 1. Change to the cloned repository directory.

[student@workstation compreview-review1]$ **cd compreview-review1**

* 1. Create the rhsso-oidc-client-secret OpenShift secret for the Red Hat SSO client secret by using the client secret from Red Hat SSO parameters in the table.
  2. [student@workstation compreview-review1]$ **oc create secret generic \**
  3. **rhsso-oidc-client-secret \**
  4. **--from-literal clientSecret=QGEP6zoLo6BUGbib5oCkGwtZ8EAlmMgW \**
  5. **-n openshift-config**

secret/rhsso-oidc-client-secret created

* 1. Create the OAuth CR YAML file. You can find an example for the CR in the ~/DO380/labs/compreview-review1/sso\_config.yml file. The YAML file includes an LDAP IdP that you must preserve, because it provides the admin and developer users. Do not remove the LDAP IdP, and add the OIDC IdP for Red Hat SSO.
  2. apiVersion: config.openshift.io/v1
  3. kind: OAuth
  4. metadata:
  5. name: cluster
  6. annotations:
  7. argocd.argoproj.io/sync-options: ServerSideApply=true,Validate=false
  8. spec:
  9. identityProviders:
  10. - ldap:
  11. ...output omitted...
  12. - openID:
  13. claims:
  14. email:
  15. - email
  16. name:
  17. - name
  18. preferredUsername:
  19. - preferred\_username
  20. groups:
  21. - groups
  22. clientID: **ocp\_rhsso**
  23. clientSecret:
  24. name: **rhsso-oidc-client-secret**
  25. extraScopes: []
  26. issuer: >-
  27. https://sso.ocp4.example.com:8080/auth/realms/internal\_devs
  28. mappingMethod: claim
  29. name: **RHSSO\_OIDC**

type: OpenID

* 1. Copy the modified sso\_config.yml file to the repository.

[student@workstation compreview-review1]$ **cp ../sso\_config.yml .**

* 1. Add the sso\_config.yml file to the Git index.

[student@workstation compreview-review1]$ **git add sso\_config.yml**

* 1. Commit the changes.
  2. [student@workstation compreview-review1]$ **git commit -m "Add SSO to OAuth"**
  3. [main f3c0ef1] Add SSO to OAuth

...output omitted...

* 1. Push the changes to the repository. The developer user is configured in the lab scripts as the default Git user.
  2. [student@workstation compreview-review1]$ **git push**

...output omitted...

* 1. Change to the ~/DO380/labs/compreview-review1 directory.

[student@workstation compreview-review1]$ **cd ..**

1. Log in to the default Argo CD instance with the admin user.
   1. Open a separate tab and open the default Argo CD instance. You can use the application menu to access this URL, by clicking **Cluster Argo CD** in the application menu, or use the https://openshift-gitops-server-openshift-gitops.apps.ocp4.example.com URL.

The browser displays a warning because Argo CD uses a self-signed certificate. Trust the certificate. Argo CD might take a few minutes before showing the login page.

* 1. Click **LOG IN VIA OPENSHIFT**, and then click **Red Hat Identity Management**. Log in as the admin user with redhatocp as the password, and then allow the **user:info** permission.

1. Create an Argo CD application with the repository and observe the results.
   1. On the Argo CD browser tab, click **CREATE APPLICATION**.
   2. Create an application with the information in the following table. Then, click **CREATE**.

| **Field** | **Value** |
| --- | --- |
| Application Name | sso-oauth |
| Project Name | default |
| Repository URL | https://git.ocp4.example.com/developer/compreview-review1.git |
| Path | . |
| Cluster URL | https://kubernetes.default.svc |

* 1. Click **sso-oauth** to view the application.
  2. Click **SYNC** to display the synchronization panel, and then click **SYNCHRONIZE**.

Argo CD starts synchronizing the application. After about one minute, the console shows the application as synchronized and healthy.

* 1. Change to the terminal window and verify the status for the OAuth pods. Wait for the OAuth pods to be redeployed. It can take a few minutes for OpenShift to redeploy the pods.
  2. [student@workstation compreview-review1]$ **watch oc get pods \**
  3. **-n openshift-authentication**
  4. Every 2.0s: oc get pods -n openshift-authentication workstation: Thu Feb 1 06:11:52 2024
  5. NAME READY STATUS RESTARTS AGE
  6. oauth-openshift-69d79b5598-85knt 1/1 Running 0 85s
  7. oauth-openshift-69d79b5598-q2nwk 1/1 Running 0 58s
  8. oauth-openshift-69d79b5598-sj2fj 1/1 Running 0 114s

**^C**

* 1. Verify that you can log in to the cluster as the filipmansur user with redhat\_sso as the password.
  2. [student@workstation compreview-review1]$ **oc login -u filipmansur -p redhat\_sso**
  3. Login successful.

...output omitted...

* 1. Log in to the OpenShift cluster as the admin user.
  2. [student@workstation compreview-review1]$ **oc login -u admin -p redhatocp**

...output omitted...

1. Create an etherpad-backup backup schedule of Etherpad. The ~/DO380/labs/compreview-review1/schedule-db-backup.yml file contains an example to create the backup definition. The backup must include resources with the app.kubernetes.io/name label with etherpad as the value, and the app label with mariadb as the value. The backup must also include the required resources to preserve the UID and GID of the namespace for OADP to restore, and to ensure that the backup hooks are executed during the backup.
   1. Edit the example to match the following text.
   2. apiVersion: velero.io/v1
   3. kind: Schedule
   4. metadata:
   5. name: **etherpad-backup**
   6. namespace: openshift-adp
   7. spec:
   8. schedule: "​0 7 \* \* 0​"
   9. paused: true
   10. template:
   11. ttl: 360h0m0s
   12. includedNamespaces:
   13. **- compreview-review1**
   14. **orLabelSelectors**:
   15. - **matchLabels**:
   16. **app.kubernetes.io/name**: **etherpad**
   17. - **matchLabels**:
   18. **app.kubernetes.io/name**: **mariadb**
   19. - **matchLabels**:
   20. **kubernetes.io/metadata.name**: **compreview-review1**
   21. includedResources:
   22. - deployment
   23. - route
   24. - service
   25. - pvc
   26. - persistentvolume
   27. - secret
   28. - service
   29. - **namespace**
   30. - **pods**
   31. hooks:

...output omitted...

* 1. Apply the configuration for the schedule resource.
  2. [student@workstation compreview-review1]$ **oc apply -f schedule-db-backup.yml**

schedule.velero.io/etherpad-backup created

* 1. Create an alias to access the velero binary from the Velero deployment in the openshift-adp namespace.
  2. [student@workstation compreview-review1]$ **alias velero='\**

**oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'**

* 1. Verify the status of the schedule with the velero command.
  2. [student@workstation compreview-review1]$ **velero get schedule**
  3. NAME STATUS ... PAUSED

etherpad-backup New ... true

1. Trigger an immediate backup from the scheduled backup, and restore it to the etherpad-test-restore project.

The backup and restore should complete without errors or warnings.

* 1. Use the velero command to start a backup by using the schedule definition from the previous step. Note the name of the backup that the command creates, to use in the next step.
  2. [student@workstation compreview-review1]$ **velero backup create \**
  3. **--from-schedule etherpad-backup**
  4. INFO[0000] No Schedule.template.metadata.labels set - using Schedule.labels for backup object backup=openshift-adp/etherpad-backup-20240131113134 labels="map[]"
  5. Creating backup from schedule, all other filters are ignored.
  6. Backup request "etherpad-backup-20240131113134" submitted successfully.

Run **velero backup describe etherpad-backup-20240131113134** or **velero backup logs etherpad-backup-20240131113134** for more details.

### Note

The S3 object storage that is configured in the lab environment uses a custom certificate that is signed with the OpenShift service CA. You must add the CA certificate to the velero backup describe --details and velero backup logs commands as follows:

[user@host]$ **velero backup logs \**

**--cacert=/run/secrets/kubernetes.io/serviceaccount/service-ca.crt \**

***etherpad-backup-20231115113447***

* 1. Monitor the status of the backup and verify that the backup ends with the Completed status. The backup process takes several minutes.
  2. [student@workstation compreview-review1]$ **velero get backup**
  3. NAME STATUS ERRORS WARNINGS ...

etherpad-backup-20231115113447 Completed 0 0 ...

* 1. Restore the backup to the etherpad-test-restore namespace.
  2. [student@workstation compreview-review1]$ **velero restore create etherpad-test \**
  3. **--from-backup etherpad-backup-*20240129143859* \**
  4. **--namespace-mappings compreview-review1:etherpad-test-restore**
  5. Restore request "etherpad-test" submitted successfully.

Run **velero restore describe etherpad-test** or **velero restore logs etherpad-test** for more details.

* 1. Use the velero command to get the status of the restore. Monitor the output to verify that the restore status is Completed. The restore process takes several minutes.
  2. [student@workstation compreview-review1]$ **velero get restore**
  3. NAME ... STATUS ...

etherpad-test ... Completed ...

* 1. Review the restored resources in the etherpad-test-restore project.

The pod for the etherpad deployment requires more time to be ready, because the pod requires the database deployment to be ready first.

[student@workstation compreview-review1]$ **oc get -n etherpad-test-restore \**

**pod,deployment,route**

NAME READY STATUS RESTARTS AGE

pod/etherpad-58f64cfb7d-8qrws 1/1 Running 3 (44s ago) 84s

pod/mariadb-66dc48b5f7-svlst 1/1 Running 0 84s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/etherpad 1/1 1 1 84s

deployment.apps/mariadb 1/1 1 1 84s

NAME HOST/PORT ...

... etherpad-etherpad-test-restore.apps.ocp4.example.com ...

* 1. Visit the https://etherpad-etherpad-test-restore.apps.ocp4.example.com URL to verify that the restored Etherpad works.

1. Use the bucket credentials to create the logging-loki-odf secret in the openshift-logging namespace. Create a LokiStack resource YAML file for an instance called logging-loki in the openshift-logging namespace. Create a ClusterLogging resource YAML file by using the loki log store, the vector collector, and the ocp-console visualization type.
   1. Use the ~/DO380/labs/compreview-review1/s3bucket.env environment file to create the logging-loki-odf secret in the openshift-logging namespace.
   2. [student@workstation compreview-review1]$ **oc create secret generic \**
   3. **logging-loki-odf -n openshift-logging --from-env-file=s3bucket.env**

secret/logging-loki-odf created

* 1. Create a LokiStack resource YAML file for an instance called logging-loki in the openshift-logging namespace. This instance uses the bucket as the storage. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/lokistack.yaml file.
  2. apiVersion: loki.grafana.com/v1
  3. kind: LokiStack
  4. metadata:
  5. name: **logging-loki**
  6. namespace: openshift-logging
  7. spec:
  8. size: 1x.demo
  9. storage:
  10. secret:
  11. name: **logging-loki-odf**
  12. type: s3
  13. tls:
  14. caName: openshift-service-ca.crt
  15. storageClassName: ocs-external-storagecluster-ceph-rbd
  16. tenants:

mode: openshift-logging

* 1. Create the LokiStack resource.
  2. [student@workstation compreview-review1]$ **oc create -f logging/lokistack.yaml**

lokistack.loki.grafana.com/logging-loki created

* 1. Create a ClusterLogging resource YAML file by using the loki log store, the vector collector, and the ocp-console visualization type. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/clusterlogging.yaml file.
  2. apiVersion: logging.openshift.io/v1
  3. kind: ClusterLogging
  4. metadata:
  5. name: instance
  6. namespace: openshift-logging
  7. spec:
  8. managementState: Managed
  9. logStore:
  10. type: **lokistack**
  11. lokistack:
  12. name: logging-loki
  13. collection:
  14. type: **vector**
  15. visualization:

type: **ocp-console**

* 1. Create the ClusterLogging resource.
  2. [student@workstation compreview-review1]$ **oc create -f logging/clusterlogging.yaml**

clusterlogging.logging.openshift.io/instance created

* 1. Verify that the ClusterLogging and LokiStack pods are up and running.
  2. [student@workstation compreview-review1]$ **oc get pods -n openshift-logging**
  3. NAME READY STATUS RESTARTS AGE
  4. cluster-logging-operator-554849f7dd-75hjk 1/1 Running 0 6m7s
  5. collector-2dqsh 1/1 Running 0 18s
  6. collector-5fp2q 1/1 Running 0 19s
  7. collector-c6zjv 1/1 Running 0 19s
  8. collector-gggtc 1/1 Running 0 13s
  9. collector-k5zs8 1/1 Running 0 13s
  10. collector-wgz7q 1/1 Running 0 12s
  11. logging-loki-compactor-0 1/1 Running 0 83s
  12. logging-loki-distributor-c55478c4c-9kt2k 1/1 Running 0 83s
  13. logging-loki-gateway-75d6fccb68-krk49 2/2 Running 0 82s
  14. logging-loki-gateway-75d6fccb68-vghdg 2/2 Running 0 82s
  15. logging-loki-index-gateway-0 1/1 Running 0 82s
  16. logging-loki-ingester-0 1/1 Running 0 83s
  17. logging-loki-querier-6f7d8b7564-4glpp 1/1 Running 0 83s
  18. logging-loki-query-frontend-678dddf864-947hn 1/1 Running 0 83s

logging-view-plugin-5b9b5b7bdc-zrp9t 1/1 Running 0 35s

1. Enable the console plug-in for the OpenShift Logging operator. Verify that you have access to the logs.
   1. Change to the web console browser. Click **Operators** → **Installed Operators**, and select **All Projects** from the drop-down menu.
   2. Click **Red Hat OpenShift Logging**, click **Console plugin**, select **Enable**, and click **Save**.
   3. Reload the web console, and navigate to **Observe** → **Logs**. If the **Observe** → **Logs** menu is not available, then wait until the web console shows the Web console update is available message and reload the web console. You have access to logs for the application and infrastructure resources. By default, the ClusterLogging instance includes logs for the application and infrastructure, but not the audit logs. Observe the application logs, which are selected by default.
2. Include the audit logs by creating a log forwarder for the application, infrastructure, and audit logs to the LokiStack resource.
   1. Change to the terminal window, and create an ClusterLogForwarder resource YAML file for a log forwarder called instance in the openshift-logging namespace. The log forwarder must forward the application, infrastructure, and audit logs. You can find an incomplete example for the resource in the ~/DO380/labs/compreview-review1/logging/forwarder.yaml file.
   2. apiVersion: logging.openshift.io/v1
   3. kind: ClusterLogForwarder
   4. metadata:
   5. name: instance
   6. namespace: openshift-logging
   7. spec:
   8. pipelines:
   9. - name: all-to-default
   10. inputRefs:
   11. - **infrastructure**
   12. - **application**
   13. - **audit**
   14. outputRefs:

- **default**

* 1. Create the ClusterLogForwarder resource.
  2. [student@workstation compreview-review1]$ **oc create -f logging/forwarder.yaml**

clusterlogforwarder.logging.openshift.io/instance created

* 1. Change to the web console browser and reload it. You have access to the audit logs.

1. Give the filipmansur user permission to view the logs in the compreview-review1 project, and verify that the user has access to the logs. Give the permission to the filipmansur user by assigning the cluster-logging-application-view cluster role through the etherpad-devs group.
   1. Change to the terminal window.
   2. Review the required role to provide access to the application logs to the filipmansur user. You can find an example in the ~/DO380/labs/compreview-review1/logging/group-role.yaml file.
   3. apiVersion: rbac.authorization.k8s.io/v1
   4. kind: RoleBinding
   5. metadata:
   6. name: view-application-logs
   7. namespace: compreview-review1
   8. roleRef:
   9. apiGroup: rbac.authorization.k8s.io
   10. kind: ClusterRole
   11. name: cluster-logging-application-view
   12. subjects:
   13. - kind: **Group**
   14. name: **etherpad-devs**

apiGroup: rbac.authorization.k8s.io

* 1. Apply the role to the filipmansur user.
  2. [student@workstation compreview-review1]$ **oc create -f \**
  3. **logging/group-role.yaml**

rolebinding.rbac.authorization.k8s.io/view-application-logs created

* 1. Change to the browser window, open a private window, and navigate to https://console-openshift-console.apps.ocp4.example.com.
  2. Click **RHSSO\_OIDC** and log in as the filipmansur user with redhat\_sso as the password. Click **Skip tour**.
  3. Navigate to **Observe**. Verify that the **compreview-review1** project is selected. Change to the **Logs** tab. The filipmansur user has access to the application logs.

1. Configure alerts.
   1. In the OpenShift web console, change to the non-private window. Navigate to **Administration** → **Cluster Settings**, click the **Configuration** tab, and then click **Alertmanager**.
   2. In the **Alert routing** tile, click **Edit**. Change the **Group interval** and **Repeat inverval** fields to 1m.
   3. In the **Receivers** tile, click **Create Receiver**. Create a receiver with the information in the following table:

| **Field** | **Value** |
| --- | --- |
| Receiver name | persistent |
| Receiver type | **Webhook** |
| URL | http://utility.lab.example.com:8000 |

* 1. Specify the alertname=~Persistent.\* routing label.
  2. Then, click **Create**.

1. Review that monitoring shows that the mariadb persistent volume claims are nearly full.
   1. In the OpenShift web console, navigate to **Observe** → **Alerting**. The console should list the **PersistentVolumeUsageCritical** and **PersistentVolumeUsageNearFull** alerts. Besides the original Etherpad deployment, you created a second deployment with a restore. Based on the disk usage in each deployment, the number and type of alerts can vary.
   2. Change to the terminal window and connect to the utility machine with SSH as the student user.
   3. [student@workstation compreview-review1]$ **ssh utility**

...output omitted...

* 1. After approximately one minute after the receiver is created, the webhook debugger shows the alerts in the /home/student/persistent\_alerts file.
  2. [student@utility ~]$ **head persistent\_alerts**
  3. {'alerts': [{'annotations': {'description': 'PVC mariadb utilization has '
  4. 'crossed 75%. Free up some space '
  5. 'or expand the PVC.',
  6. 'message': 'PVC mariadb is nearing full. Data '
  7. 'deletion or PVC expansion is '
  8. 'required.',
  9. 'severity\_level': 'warning',
  10. 'storage\_type': 'ceph'},
  11. 'endsAt': '0001-01-01T00:00:00Z',
  12. 'fingerprint': 'd051e03da5866d5b',

...output omitted...

* 1. Disconnect from the utility machine.
  2. [student@utility ~]$ **exit**
  3. logout
  4. Connection to utility closed.

[student@workstation compreview-review1]$

* 1. Change to the student HOME directory.
  2. [student@workstation compreview-review1]$ **cd**

[student@workstation ~]$

1. Expand the PVCs.
   1. Navigate to **Storage** → **PersistentVolumeClaims**. Select **All Projects** from the project drop-down menu. Locate the two mariadb persistent volume claims.
   2. For each claim, click its name to view the details. Each claim has 190 MiB capacity and about 40 MiB available.

For each claim, select **Expand PVC** from the **Actions** list. Edit the total size of the claim to 1900 MiB, and then click **Expand**.

* 1. If you navigate to **Observe** → **Alerting**, then the alerts disappear after a few minutes.

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

**Evaluation**

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

[student@workstation ~]$ **lab grade compreview-review1**

**Finish**

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

[student@workstation ~]$ **lab finish compreview-review1**