Red Hat CodeReady Workspaces 2.15

Administration Guide

Administering Red Hat CodeReady Workspaces 2.15

Last Updated: 2022-05-17

Red Hat CodeReady Workspaces 2.15 Administration Guide

Administering Red Hat CodeReady Workspaces 2.15

Robert Kratky
rkratky@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Jana Vrbkova
jvrbkova@redhat.com

Max Leonov
mleonov@redhat.com

Legal Notice

Abstract

Information for administrators operating Red Hat CodeReady Workspaces.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

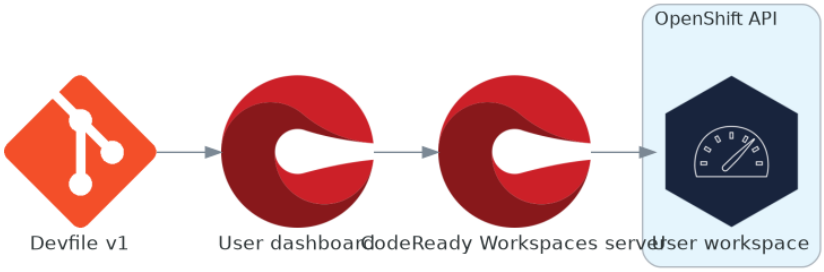
CHAPTER 1. ARCHITECTURE OVERVIEW

CodeReady Workspaces needs a workspace engine to manage the lifecycle of the workspaces. Two workspace engines are available. The choice of a workspace engine defines the architecture.

Section 1.1, "CodeReady Workspaces architecture with CodeReady Workspaces server"

CodeReady Workspaces server is the default workspace engine.

Figure 1.1. High-level CodeReady Workspaces architecture with the CodeReady Workspaces server engine



Section 1.4, "CodeReady Workspaces architecture with Dev Workspace"

The Dev Workspace Operator is a new workspace engine.

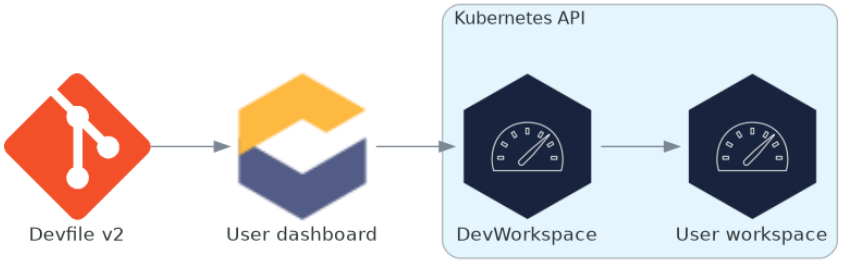
TECHNOLOGY PREVIEW FEATURE

Managing workspaces with the Dev Workspace engine is an experimental feature. Don't use this workspace engine in production.

Known limitations

Workspaces are not secured. Whoever knows the URL of a workspace can have access to it and leak the user credentials.

Figure 1.2. High-level CodeReady Workspaces architecture with the Dev Workspace operator



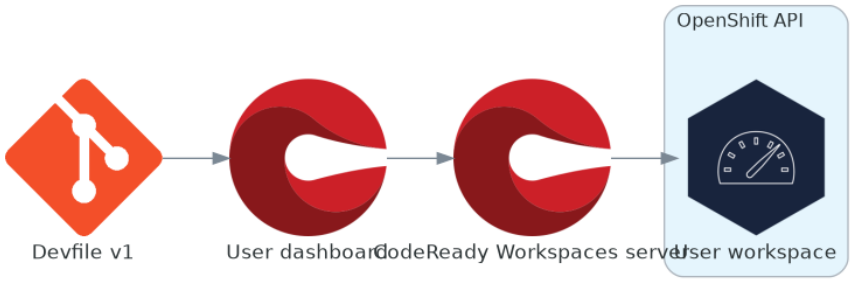
Additional resources

- Section 1.1, “CodeReady Workspaces architecture with CodeReady Workspaces server”
- Section 1.4, “CodeReady Workspaces architecture with Dev Workspace”
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc
- Dev Workspace Operator GitHub repository

1.1. CODEREADY WORKSPACES ARCHITECTURE WITH CODEREADY WORKSPACES SERVER

CodeReady Workspaces server is the default workspace engine.

Figure 1.3. High-level CodeReady Workspaces architecture with the CodeReady Workspaces server engine



Red Hat CodeReady Workspaces components are:

CodeReady Workspaces server

An always-running service that manages user workspaces with the OpenShift API.

User workspaces

Container-based IDEs running on user requests.

Additional resources

- Section 1.2, "Understanding CodeReady Workspaces server"
- Section 1.3, "Understanding CodeReady Workspaces workspaces architecture"

1.2. UNDERSTANDING CODEREADY WORKSPACES SERVER

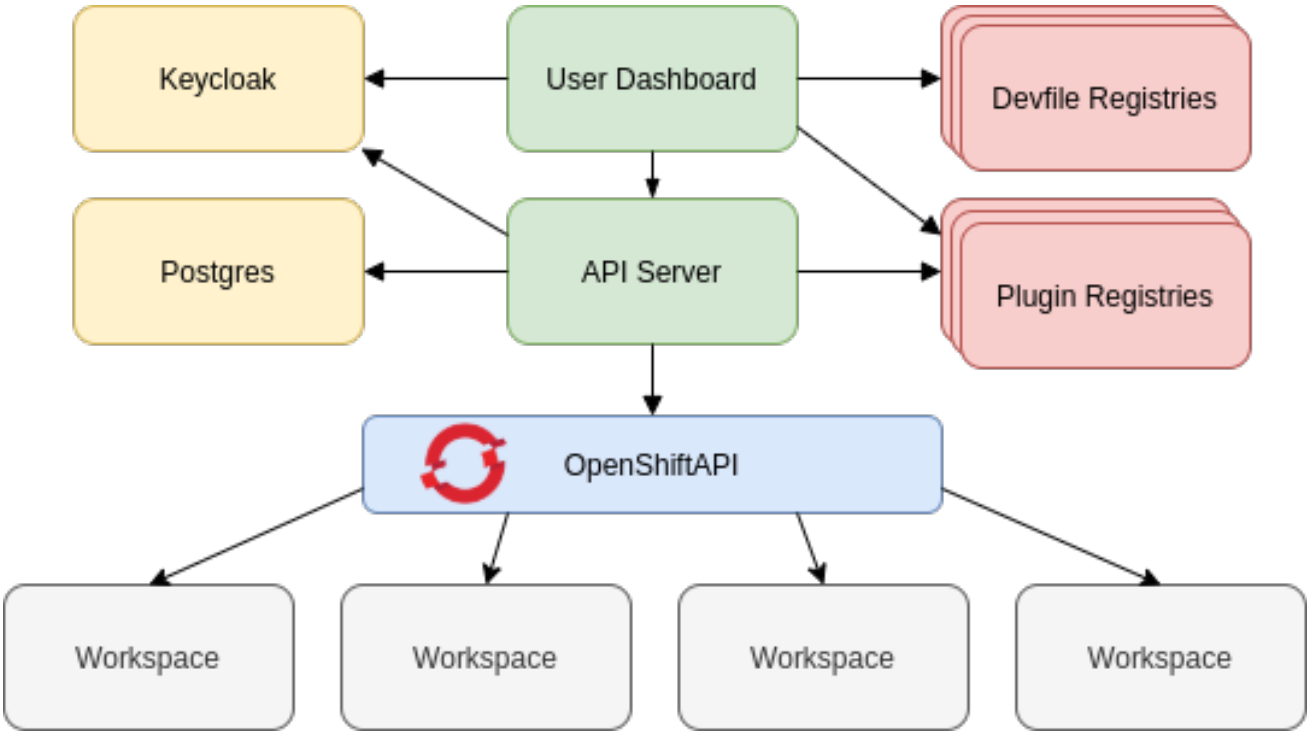
This chapter describes the CodeReady Workspaces controller and the services that are a part of the controller.

1.2.1. CodeReady Workspaces server

The workspaces controller manages the container-based development environments: CodeReady Workspaces workspaces. To secure the development environments with authentication, the deployment is always multiuser and multitenant.

The following diagram shows the different services that are a part of the CodeReady Workspaces workspaces controller.

Figure 1.4. CodeReady Workspaces workspaces controller



Additional resources

- Section 12.1, "Authenticating users"

1.2.2. CodeReady Workspaces server

The CodeReady Workspaces server is the central service of CodeReady Workspaces server-side components. It is a Java web service exposing an HTTP REST API to manage CodeReady Workspaces workspaces and users. It is the default workspace engine.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

1.2.3. CodeReady Workspaces user dashboard

The user dashboard is the landing page of Red Hat CodeReady Workspaces. It is a React application. CodeReady Workspaces users navigate the user dashboard from their browsers to create, start, and manage CodeReady Workspaces workspaces.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#navigating-che.adoc

1.2.4. CodeReady Workspaces devfile registry

The CodeReady Workspaces devfile registry is a service that provides a list of CodeReady Workspaces samples to create ready-to-use workspaces. This list of samples is used in the **Dashboard → Create Workspace** window. The devfile registry runs in a container and can be deployed wherever the user

dashboard can connect.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#creating-a-workspace-from-a-code-sample.adoc
- CodeReady Workspaces devfile registry repository

1.2.5. CodeReady Workspaces plug-in registry

The CodeReady Workspaces plug-in registry is a service that provides the list of plug-ins and editors for CodeReady Workspaces workspaces. A devfile only references a plug-in that is published in a CodeReady Workspaces plug-in registry. It runs in a container and can be deployed wherever CodeReady Workspaces server connects.

1.2.6. CodeReady Workspaces and PostgreSQL

The PostgreSQL database is a prerequisite for CodeReady Workspaces server and RH-SSO.

The CodeReady Workspaces administrator can choose to:

- Connect CodeReady Workspaces to an existing PostgreSQL instance.
- Let the CodeReady Workspaces deployment start a new dedicated PostgreSQL instance.

Services use the database for the following purposes:

CodeReady Workspaces server

Persist user configurations such as workspaces metadata and Git credentials.

RH-SSO

Persist user information.

Additional resources

- Section 8.8, “Backups of PostgreSQL”
- **quay.io/eclipse/che-postgres** container image
- CodeReady Workspaces Postgres repository

1.2.7. CodeReady Workspaces and RH-SSO

RH-SSO is a prerequisite to configure CodeReady Workspaces. The CodeReady Workspaces administrator can choose to connect CodeReady Workspaces to an existing RH-SSO instance or let the CodeReady Workspaces deployment start a new dedicated RH-SSO instance.

The CodeReady Workspaces server uses RH-SSO as an OpenID Connect (OIDC) provider to authenticate CodeReady Workspaces users and secure access to CodeReady Workspaces resources.

Additional resources

- **quay.io/eclipse/che-keycloak** container image

- CodeReady Workspaces RH-SSO repository

1.3. UNDERSTANDING CODEREADY WORKSPACES WORKSPACES ARCHITECTURE

This chapter describes the architecture and components of CodeReady Workspaces.

1.3.1. CodeReady Workspaces workspaces architecture

A CodeReady Workspaces deployment on the cluster consists of the CodeReady Workspaces server component, a database for storing user profile and preferences, and several additional deployments hosting workspaces. The CodeReady Workspaces server orchestrates the creation of workspaces, which consist of a deployment containing the workspace containers and enabled plug-ins, plus the related components, such as:

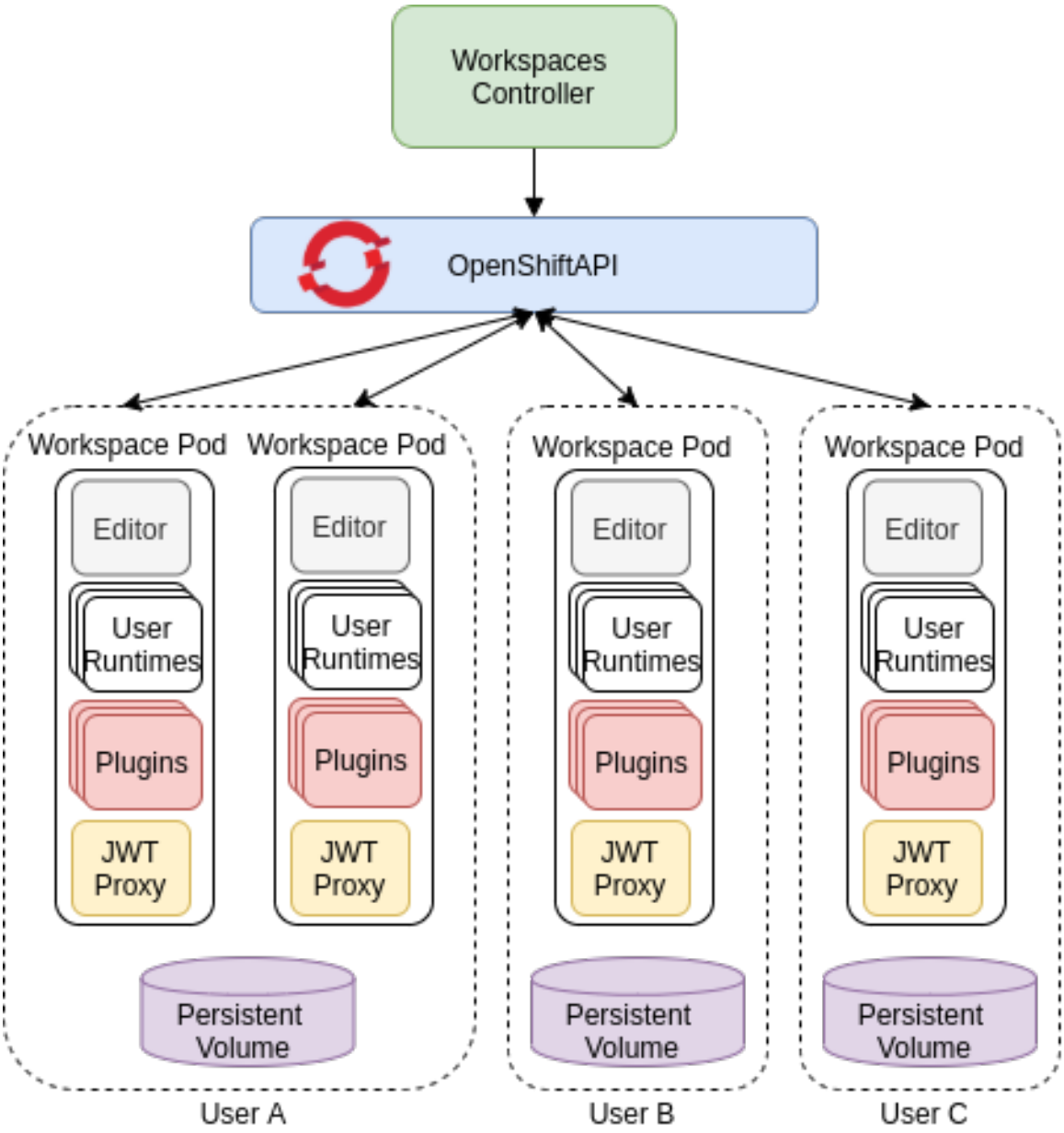
- ConfigMaps
- services
- endpoints
- ingresses or routes
- secrets
- persistent volumes (PVs)

The CodeReady Workspaces workspace is a web application. It is composed of microservices running in containers that provide all the services of a modern IDE such as an editor, language auto-completion, and debugging tools. The IDE services are deployed with the development tools, packaged in containers and user runtime applications, which are defined as OpenShift resources.

The source code of the projects of a CodeReady Workspaces workspace is persisted in a OpenShift **PersistentVolume**. Microservices run in containers that have read-write access to the source code (IDE services, development tools), and runtime applications have read-write access to this shared directory.

The following diagram shows the detailed components of a CodeReady Workspaces workspace.

Figure 1.5. CodeReady Workspaces workspace components



In the diagram, there are four running workspaces: two belonging to **User A**, one to **User B** and one to **User C**.

Use the devfile format to specify the tools and runtime applications of a CodeReady Workspaces workspace.

1.3.2. CodeReady Workspaces workspace components

This section describes the components of a CodeReady Workspaces workspace.

1.3.2.1. Che Editor plug-in

A **Che Editor** plug-in is a CodeReady Workspaces workspace plug-in. It defines the web application that is used as an editor in a workspace. The default CodeReady Workspaces workspace editor is Che-Theia. It is a web-based source-code editor similar to Visual Studio Code (Visual Studio Code). It has a plug-in system that supports Visual Studio Code extensions.

Source code	Che-Theia
Container image	**eclipse/che-theia**
Endpoints	**theia**, **webviews**, **theia-dev**, **theia-redirect-1**, **theia-redirect-2**, **theia-redirect-3**

Additional resources

- Che-Theia
- Eclipse Theia open source project
- Visual Studio Code

1.3.2.2. CodeReady Workspaces user runtimes

Use any non-terminating user container as a user runtime. An application that can be defined as a container image or as a set of OpenShift resources can be included in a CodeReady Workspaces workspace. This makes it easy to test applications in the CodeReady Workspaces workspace.

To test an application in the CodeReady Workspaces workspace, include the application YAML definition used in stage or production in the workspace specification. It is a 12-factor application development / production parity.

Examples of user runtimes are Node.js, SpringBoot or MongoDB, and MySQL.

1.3.2.3. CodeReady Workspaces workspace JWT proxy

The JWT proxy is responsible for securing the communication of the CodeReady Workspaces workspace services.

An HTTP proxy is used to sign outgoing requests from a workspace service to the CodeReady Workspaces server and to authenticate incoming requests from the IDE client running on a browser.

Source code	JWT proxy
Container image	**eclipse/che-jwtproxy**

1.3.2.4. CodeReady Workspaces plug-ins broker

Plug-in brokers are special services that, given a plug-in **meta.yaml** file:

- Gather all the information to provide a plug-in definition that the CodeReady Workspaces server knows.
- Perform preparation actions in the workspace project (download, unpack files, process configuration).

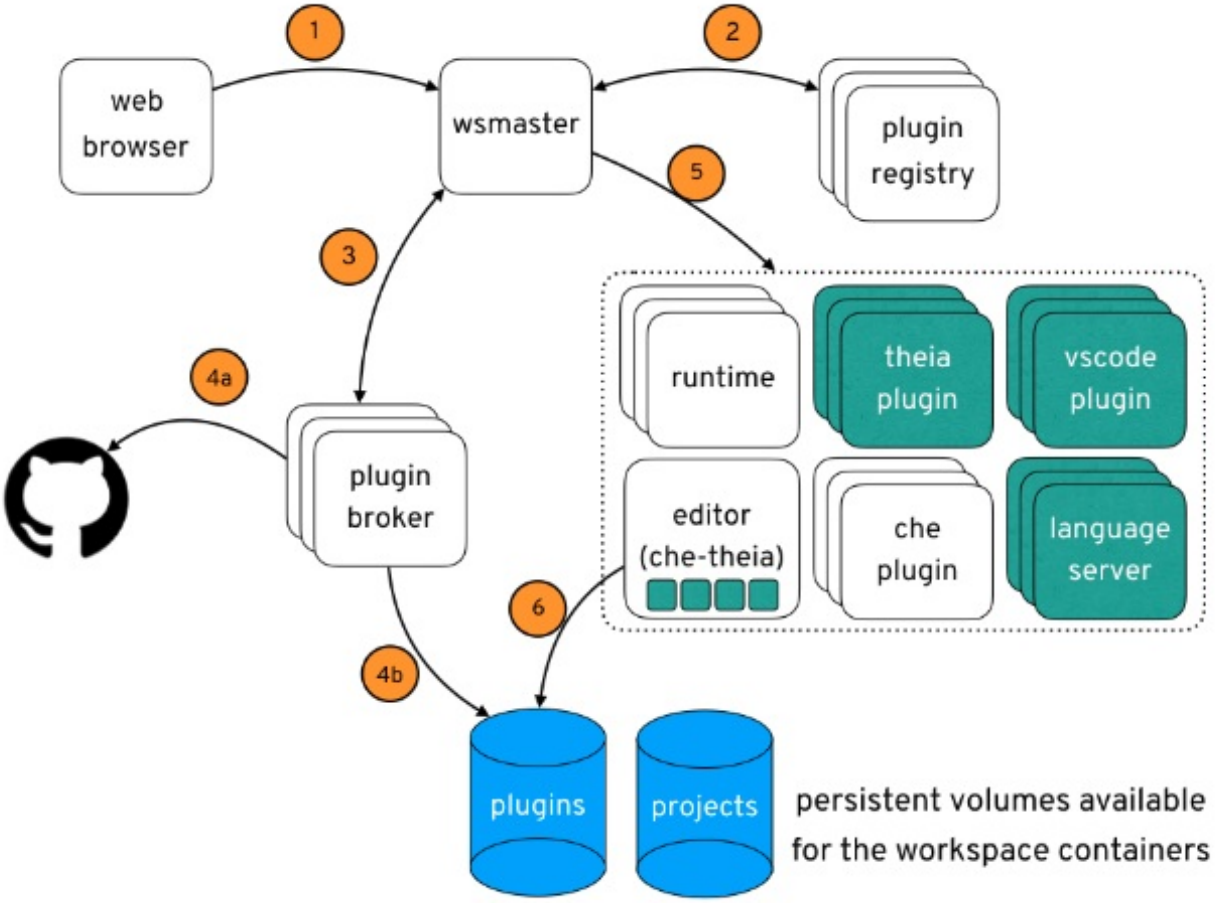
The main goal of the plug-in broker is to decouple the CodeReady Workspaces plug-ins definitions from the actual plug-ins that CodeReady Workspaces can support. With brokers, CodeReady Workspaces can support different plug-ins without updating the CodeReady Workspaces server.

The CodeReady Workspaces server starts the plug-in broker. The plug-in broker runs in the same OpenShift project as the workspace. It has access to the plug-ins and project persistent volumes.

A plug-ins broker is defined as a container image (for example, **eclipse/che-plugin-broker**). The plug-in type determines the type of the broker that is started. Two types of plug-ins are supported: **Che Plugin** and **Che Editor**.

Source code	CodeReady Workspaces Plug-in broker
Container image	**quay.io/eclipse/che-plugin-artifacts-broker** **eclipse/che-plugin-metadata-broker**

1.3.3. CodeReady Workspaces workspace creation flow



The following is a CodeReady Workspaces workspace creation flow:

1. A user starts a CodeReady Workspaces workspace defined by:
   - An editor (the default is Che-Theia)
   - A list of plug-ins (for example, Java and OpenShift tools)
   - A list of runtime applications
2. CodeReady Workspaces server retrieves the editor and plug-in metadata from the plug-in registry.

3. For every plug-in type, CodeReady Workspaces server starts a specific plug-in broker.
4. The CodeReady Workspaces plug-ins broker transforms the plug-in metadata into a **Che Plugin** definition. It executes the following steps:
   a. Downloads a plug-in and extracts its content.
   b. Processes the plug-in **meta.yaml** file and sends it back to CodeReady Workspaces server in the format of a **Che Plugin**.
5. CodeReady Workspaces server starts the editor and the plug-in sidecars.
6. The editor loads the plug-ins from the plug-in persistent volume.

1.4. CODEREADY WORKSPACES ARCHITECTURE WITH DEV WORKSPACE

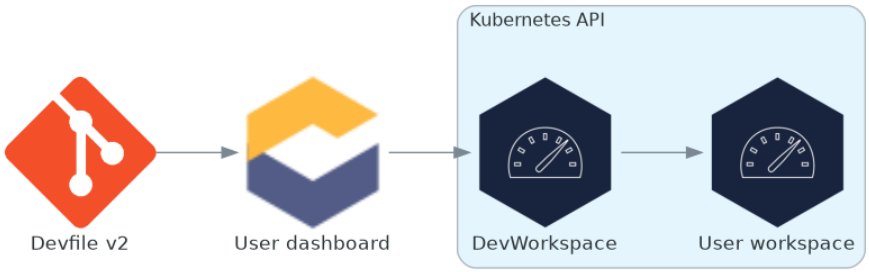
TECHNOLOGY PREVIEW FEATURE

Managing workspaces with the Dev Workspace engine is an experimental feature. Don't use this workspace engine in production.

Known limitations

Workspaces are not secured. Whoever knows the URL of a workspace can have access to it and leak the user credentials.

Figure 1.6. High-level CodeReady Workspaces architecture with the Dev Workspace operator



When CodeReady Workspaces is running with the Dev Workspace operator, it runs on three groups of components:

CodeReady Workspaces server components

Manage User project and workspaces. The main component is the User dashboard, from which users control their workspaces.

Dev Workspace operator

Creates and controls the necessary OpenShift objects to run User workspaces. Including **Pods**, **Services**, and **PeristentVolumes**.

User workspaces

Container-based development environments, the IDE included.

The role of these OpenShift features is central:

Dev Workspace Custom Resources

Valid OpenShift objects representing the User workspaces and manipulated by CodeReady Workspaces. It is the communication channel for the three groups of components.

OpenShift role-based access control (RBAC)

Controls access to all resources.

Additional resources

- Section 1.5, “CodeReady Workspaces server components”
- Section 1.5.2, “Dev Workspace operator”
- Section 1.6, “User workspaces”
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc
- Dev Workspace Operator repository
- Kubernetes documentation - Custom Resources

1.5. CODEREADY WORKSPACES SERVER COMPONENTS

TECHNOLOGY PREVIEW FEATURE

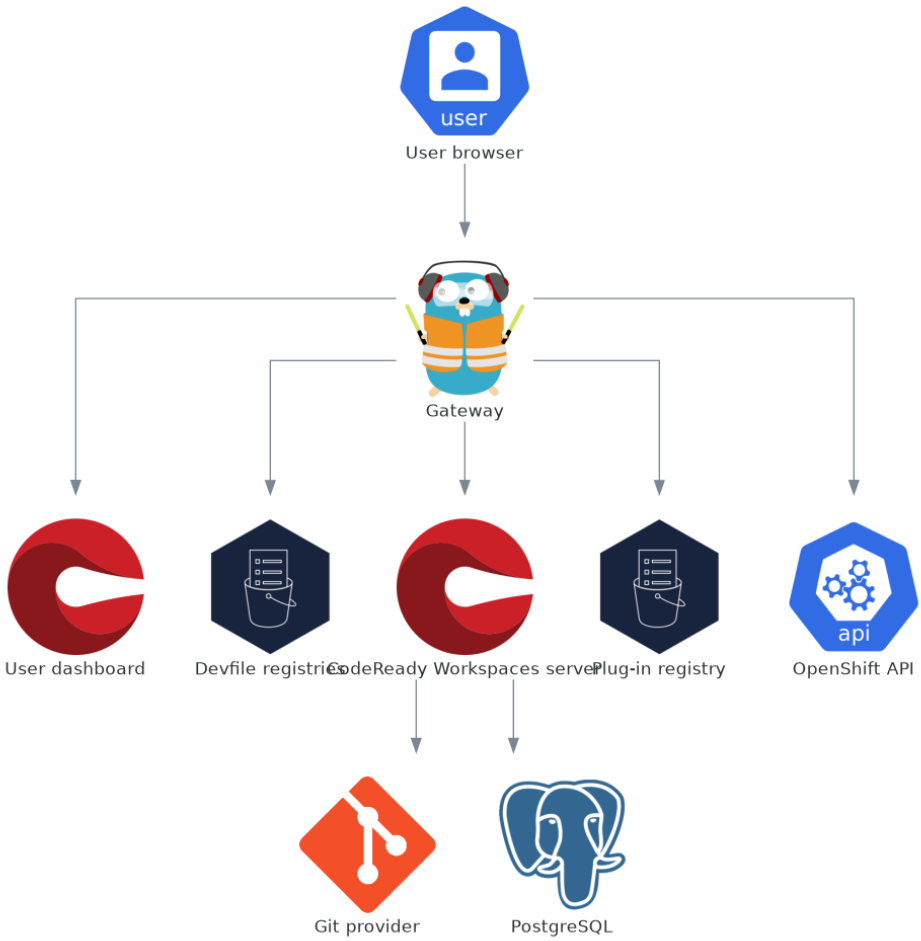
Managing workspaces with the Dev Workspace engine is an experimental feature. Don’t use this workspace engine in production.

Known limitations

Workspaces are not secured. Whoever knows the URL of a workspace can have access to it and leak the user credentials.

The CodeReady Workspaces server components ensure multi-tenancy and workspaces management.

Figure 1.7. CodeReady Workspaces server components interacting with the Dev Workspace operator



Additional resources

- Section 1.5.1, “CodeReady Workspaces operator”
- Section 1.5.2, “Dev Workspace operator”
- Section 1.5.3, “Gateway”
- Section 1.5.4, “User dashboard”
- Section 1.5.5, “Devfile registries”
- Section 1.5.6, “CodeReady Workspaces server”
- Section 1.5.7, “PostgreSQL”

- Section 1.5.8, “Plug-in registry”

1.5.1. CodeReady Workspaces operator

The CodeReady Workspaces operator ensure full lifecycle management of the CodeReady Workspaces server components. It introduces:

CheCluster custom resource definition (CRD)

Defines the **CheCluster** OpenShift object.

CodeReady Workspaces controller

Creates and controls the necessary OpenShift objects to run a CodeReady Workspaces instance, such as pods, services, and persistent volumes.

CheCluster custom resource (CR)

On a cluster with the CodeReady Workspaces operator, it is possible to create a **CheCluster** custom resource (CR). The CodeReady Workspaces operator ensures the full lifecycle management of the CodeReady Workspaces server components on this CodeReady Workspaces instance:

- Section 1.5.2, “Dev Workspace operator”
- Section 1.5.3, “Gateway”
- Section 1.5.4, “User dashboard”
- Section 1.5.5, “Devfile registries”
- Section 1.5.6, “CodeReady Workspaces server”
- Section 1.5.7, “PostgreSQL”
- Section 1.5.8, “Plug-in registry”

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-the-che-installation.adoc
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#installing-che.adoc

1.5.2. Dev Workspace operator

TECHNOLOGY PREVIEW FEATURE

Managing workspaces with the Dev Workspace engine is an experimental feature. Don't use this workspace engine in production.

Known limitations

Workspaces are not secured. Whoever knows the URL of a workspace can have access to it and leak the user credentials.

The Dev Workspace operator extends OpenShift to provide Dev Workspace support. It introduces:

Dev Workspace custom resource definition

Defines the Dev Workspace OpenShift object from the Devfile v2 specification.

Dev Workspace controller

Creates and controls the necessary OpenShift objects to run a Dev Workspace, such as pods, services, and persistent volumes.

Dev Workspace custom resource

On a cluster with the Dev Workspace operator, it is possible to create Dev Workspace custom resources (CR). A Dev Workspace CR is a OpenShift representation of a Devfile. It defines a User workspaces in a OpenShift cluster.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc
- Devfile API repository

1.5.3. Gateway

The CodeReady Workspaces gateway has following roles:

- Routing requests. It uses Traefik.
- Authenticating users with OpenID Connect (OIDC). It uses OpenShift OAuth2 proxy.
- Applying OpenShift Role based access control (RBAC) policies to control access to any CodeReady Workspaces resource. It uses `kube-rbac-proxy`.

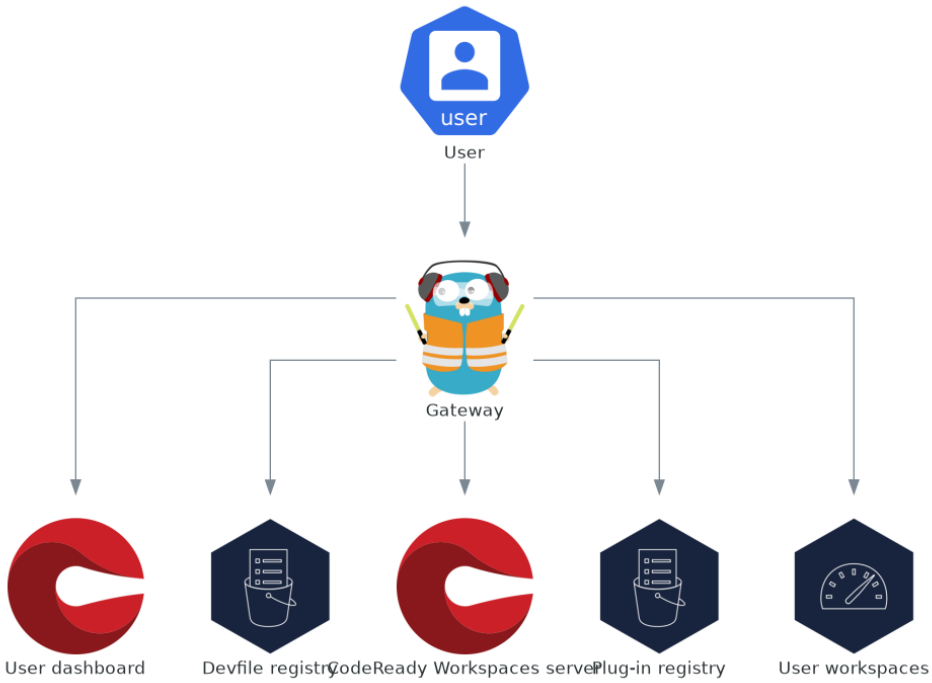
The CodeReady Workspaces operator manages it as the **che-gateway** Deployment.

It controls access to:

- Section 1.5.4, "User dashboard"
- Section 1.5.5, "Devfile registries"
- Section 1.5.6, "CodeReady Workspaces server"

- Section 1.5.8, “Plug-in registry”
- Section 1.6, “User workspaces”

Figure 1.8. CodeReady Workspaces gateway interactions with other components



Additional resources

- Chapter 12, *Managing identities and authorizations*

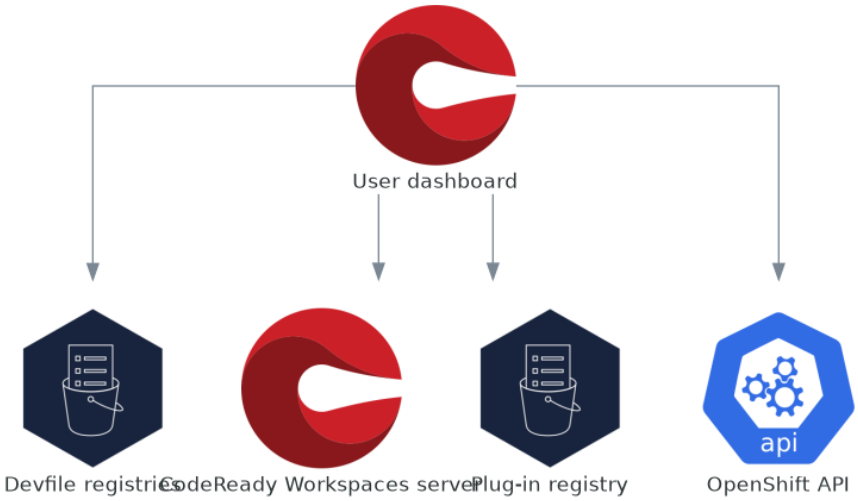
1.5.4. User dashboard

The user dashboard is the landing page of Red Hat CodeReady Workspaces. CodeReady Workspaces end-users browse the user dashboard to access and manage their workspaces. It is a React application. The CodeReady Workspaces deployment starts it in the **codeready-dashboard** Deployment.

It need access to:

- Section 1.5.5, “Devfile registries”
- Section 1.5.6, “CodeReady Workspaces server”
- Section 1.5.8, “Plug-in registry”
- OpenShift API

Figure 1.9. User dashboard interactions with other components



When the user requests the user dashboard to start a workspace, the user dashboard executes this sequence of actions:

1. Collects the devfile from the Section 1.5.5, "Devfile registries", when the user is Creating a workspace from a code sample.
2. Sends the repository URL to Section 1.5.6, "CodeReady Workspaces server" and expects a devfile in return, when the user is Creating a workspace from remote devfile .
3. Reads the devfile describing the workspace.
4. Collects the additional metadata from the Section 1.5.8, "Plug-in registry".
5. Converts the information into a Dev Workspace Custom Resource.
6. Creates the Dev Workspace Custom Resource in the user project using the OpenShift API.
7. Watches the Dev Workspace Custom Resource status.
8. Redirects the user to the running workspace IDE.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#navigating-che.adoc

1.5.5. Devfile registries

The CodeReady Workspaces devfile registries are services providing a list of sample devfiles to create ready-to-use workspaces. The Section 1.5.4, "User dashboard" displays the samples list on the **Dashboard → Create Workspace** page. Each sample includes a Devfile v2. The CodeReady Workspaces deployment starts one devfile registry instance in the **devfile-registry** deployment.

Figure 1.10. Devfile registries interactions with other components

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#creating-a-workspace-from-a-code-sample.adoc
- Devfile v2 documentation
- devfile registry latest community version online instance
- CodeReady Workspaces devfile registry repository

1.5.6. CodeReady Workspaces server

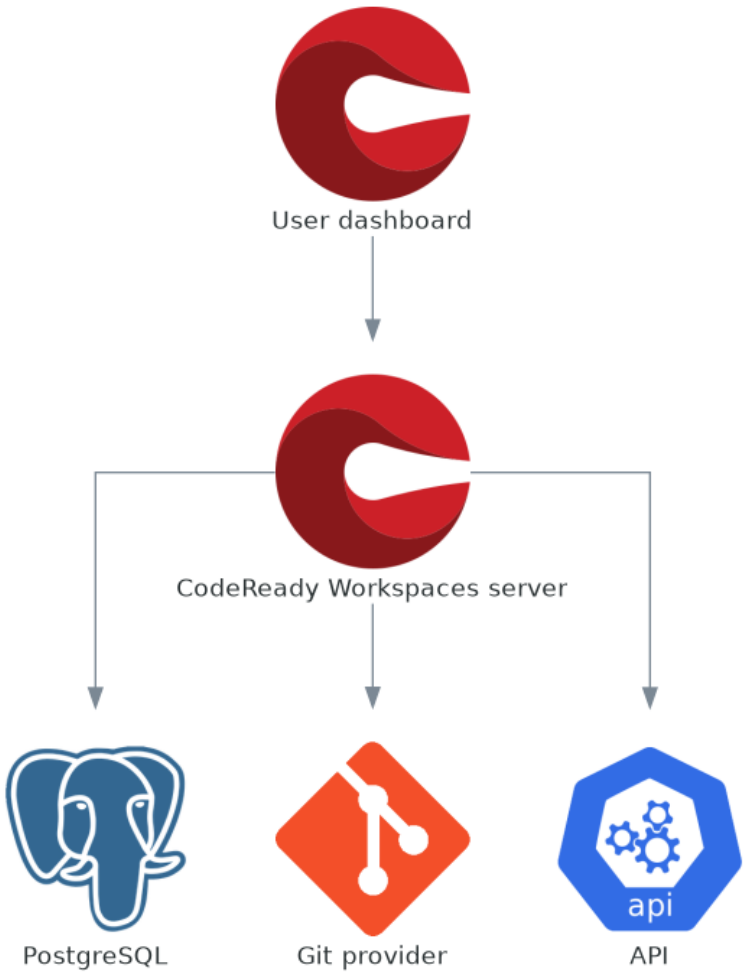
The CodeReady Workspaces server main functions are:

- Creating user namespaces.
- Provisioning user namespaces with required secrets and config maps.
- Integrating with Git services providers, to fetch and validate devfiles and authentication.

The CodeReady Workspaces server is a Java web service exposing an HTTP REST API and needs access to:

- Section 1.5.7, “PostgreSQL”
- Git service providers
- OpenShift API

Figure 1.11. CodeReady Workspaces server interactions with other components



Additional resources

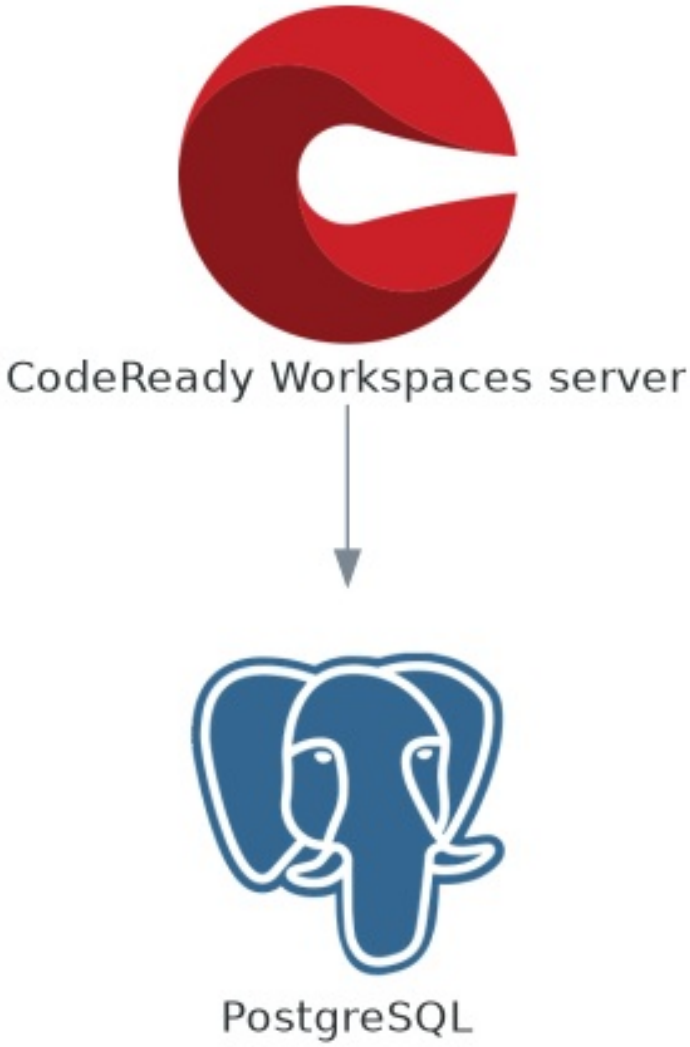
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

1.5.7. PostgreSQL

CodeReady Workspaces server uses the PostgreSQL database to persist user configurations such as workspaces metadata.

The CodeReady Workspaces deployment starts a dedicated PostgreSQL instance in the **postgres** Deployment. You can use an external database instead.

Figure 1.12. PostgreSQL interactions with other components



Additional resources

- Section 8.8, "Backups of PostgreSQL"
- **quay.io/eclipse/che-postgres** container image
- CodeReady Workspaces Postgres repository

1.5.8. Plug-in registry

Each CodeReady Workspaces workspace starts with a specific editor and set of associated extensions. The CodeReady Workspaces plug-in registry provides the list of available editors and editor extensions. A Devfile v2 describes each editor or extension.

The Section 1.5.4, "User dashboard" is reading the content of the registry.

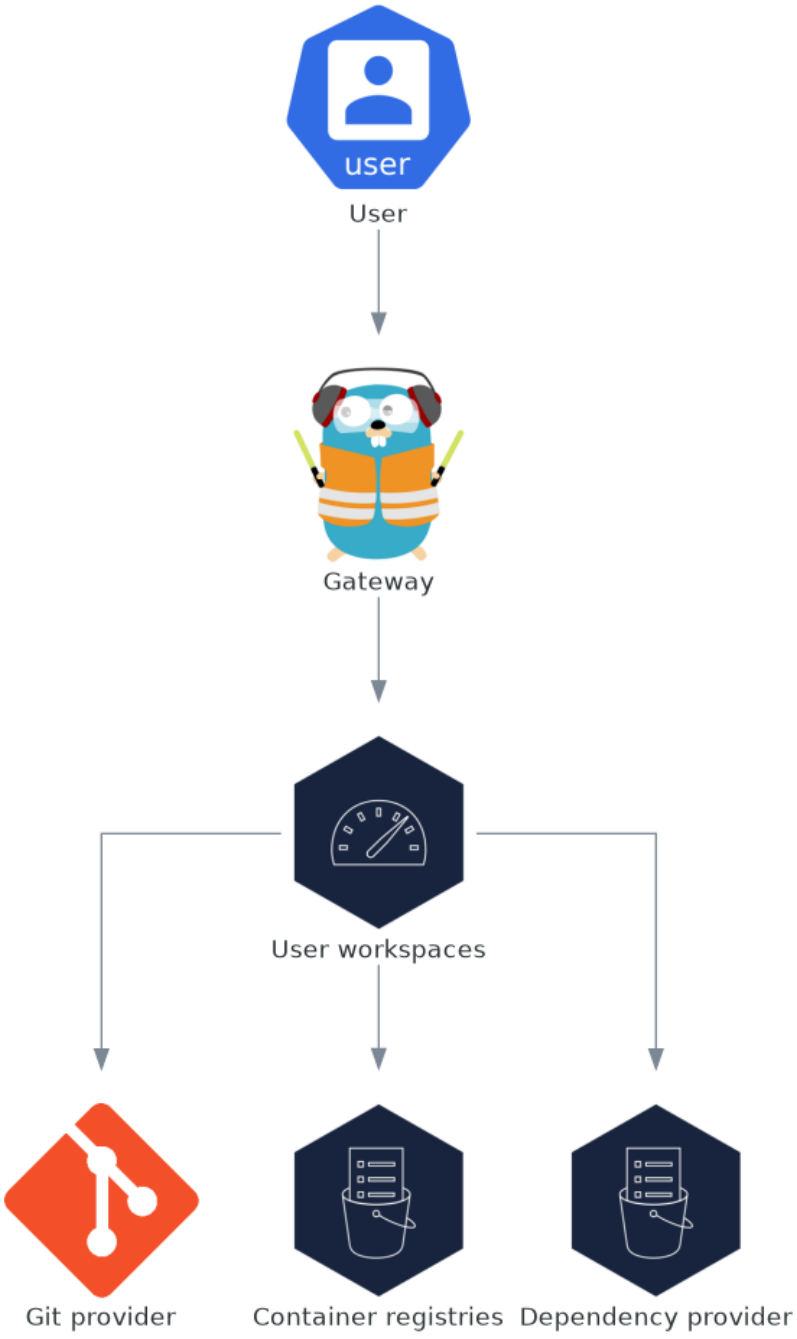
Figure 1.13. Plug-in registries interactions with other components

Additional resources

- Editors definitions in the CodeReady Workspaces plug-in registry repository
- Plug-ins definitions in the CodeReady Workspaces plug-in registry repository
- Plug-in registry latest community version online instance

1.6. USER WORKSPACES

Figure 1.14. User workspaces interactions with other components



User workspaces are web IDEs running in containers.

A User workspace is a web application. It consists of microservices running in containers providing all the services of a modern IDE running in your browser:

- Editor
- Language auto-completion
- Language server
- Debugging tools
- Plug-ins
- Application runtimes

A workspace is one OpenShift Deployment containing the workspace containers and enabled plug-ins, plus related OpenShift components:

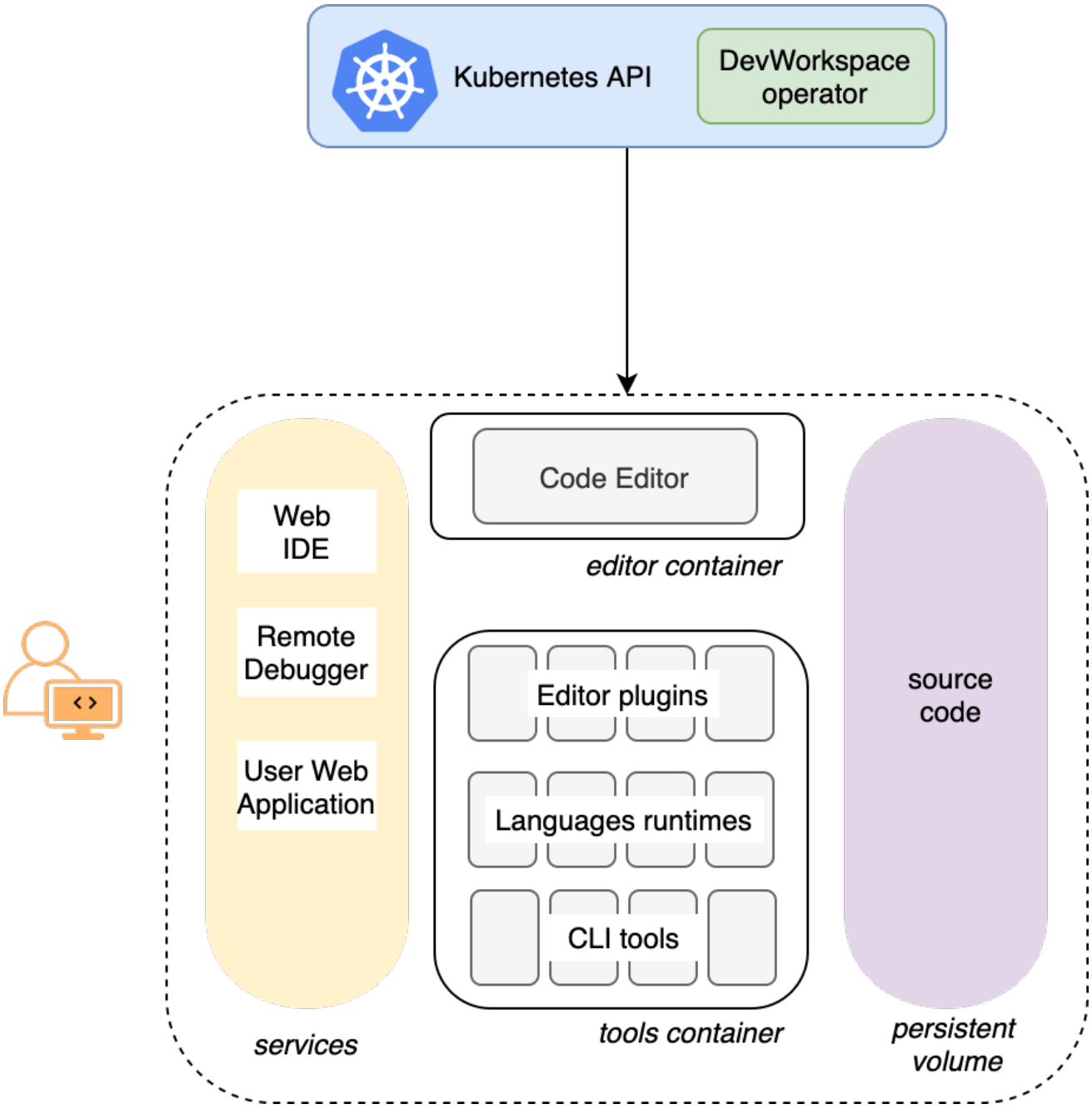
- Containers
- ConfigMaps
- Services
- Endpoints
- Ingresses or Routes
- Secrets
- Persistent Volumes (PVs)

A CodeReady Workspaces workspace contains the source code of the projects, persisted in a OpenShift Persistent Volume (PV). Microservices have read-write access to this shared directory.

Use the devfile v2 format to specify the tools and runtime applications of a CodeReady Workspaces workspace.

The following diagram shows one running CodeReady Workspaces workspace and its components.

Figure 1.15. CodeReady Workspaces workspace components



In the diagram, there is one running workspaces.

CHAPTER 2. CALCULATING CODEREADY WORKSPACES RESOURCE REQUIREMENTS

Additional resources

This section describes how to calculate resources, such as memory and CPU, required to run Red Hat CodeReady Workspaces.

Both the CodeReady Workspaces central controller and user workspaces consist of a set of containers. Those containers contribute to the resources consumption in terms of CPU and RAM limits and requests.

2.1. CONTROLLER REQUIREMENTS

The Workspace Controller consists of a set of five services running in five distinct containers. The following table presents the default resource requirements of each of these services.

Table 2.1. ControllerServices

Pod	Container name	Default memory limit	Default memory request
CodeReady Workspaces Server and Dashboard	che	1 GiB	512 MiB
PostgreSQL	**postgres**	1 GiB	512 MiB
RH-SSO	**keycloak**	2 GiB	512 MiB
Devfile registry	**che-devfile-registry**	256 MiB	16 MiB
Plug-in registry	**che-plugin-registry**	256 MiB	16 MiB

These default values are sufficient when the CodeReady Workspaces Workspace Controller manages a small amount of CodeReady Workspaces workspaces. For larger deployments, increase the memory limit. See the https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc article for instructions on how to override the default requests and limits. For example, the Eclipse Che hosted by Red Hat that runs on https://workspaces.openshift.com uses 1 GB of memory.

Additional resources

- Section 1.2, “Understanding CodeReady Workspaces server” .

2.2. WORKSPACES REQUIREMENTS

This section describes how to calculate the resources required for a workspace. It is the sum of the resources required for each component of this workspace.

These examples demonstrate the necessity of a proper calculation:

- A workspace with ten active plug-ins requires more resources than the same workspace with fewer plug-ins.
- A standard Java workspace requires more resources than a standard Node.js workspace because running builds, tests, and application debugging requires more resources.

Procedure

1. Identify the workspace components explicitly specified in the **components** section of the https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc.
2. Identify the implicit workspace components:
   a. CodeReady Workspaces implicitly loads the default **cheEditor**: **che-theia**, and the **chePlugin** that allows commands execution: **che-machine-exec-plugin**. To change the default editor, add a **cheEditor** component section in the devfile.
   b. The JWT Proxy component is responsible for the authentication and authorization of the external communications of the workspace components.
3. Calculate the requirements for each component:
   a. Default values:
   The following table displays the default requirements for all workspace components, and the corresponding CodeReady Workspaces server properties. Use the CodeReady Workspaces server properties to modify the defaults cluster-wide.

Table 2.2. Default requirements of workspace components by type

Component types	CodeReady Workspaces server property	Default memory limit	Default memory request
chePlugin	**che.workspace.sidecar.default_memory_limit_mb**	128 MiB	64 MiB
cheEditor	**che.workspace.sidecar.default_memory_limit_mb**	128 MiB	64 MiB
kubernetes, **openshift**, **dockerimage**	**che.workspace.default_memory_limit_mb**, **che.workspace.default_memory_request_mb**	1 Gi	200 MiB

Component types	CodeReady Workspaces server property	Default memory limit	Default memory request
JWT Proxy	**che.server.secure_exposer.jwtproxy.memory_limit**, **che.server.secure_exposer.jwtproxy.memory_request**	128 MiB	15 MiB

b. Custom requirements for **chePlugins** and **cheEditors** components:

i. Custom memory limit and request:
Define the **memoryLimit** and **memoryRequest** attributes of the **containers** section of the **meta.yaml** file to configure the memory limit of the **chePlugins** or **cheEditors** components. CodeReady Workspaces automatically sets the memory request to match the memory limit if it is not specified explicitly.

Example 2.1. The chePlugin che-incubator/typescript/latest

meta.yaml spec section:

```
spec:
 containers:
   - image: docker.io/eclipse/che-remote-plugin-node:next
     name: vscode-typescript
     memoryLimit: 512Mi
     memoryRequest: 256Mi
```

This results in a container with the following memory limit and request:

Memory limit	512 MiB
Memory request	256 MiB

NOTE

For IBM Power (ppc64le), the memory limit for some plugins has been increased by up to 1.5G to allow pods sufficient RAM to run. For example, on IBM Power (ppc64le), the Theia editor pod requires 2G; the OpenShift connector pod requires 2.5G. For AMD64 and Intel 64 (x86_64) and IBM Z (s390x), memory requirements remain lower at 512M and 1500M respectively. However, some devfiles may still be configured to set the lower limit valid for AMD64 and Intel 64 (x86_64) and IBM Z (s390x), so to work around this, edit devfiles for workspaces that are crashing to increase the default memoryLimit by at least 1 - 1.5 GB.

NOTE

How to find the meta.yaml file of chePlugin

Community plug-ins are available in the CodeReady Workspaces plug-ins registry repository in folder **v3/plugins/${organization}/${name}/${version}/**.

For non-community or customized plug-ins, the **meta.yaml** files are available on the local OpenShift cluster at **${pluginRegistryEndpoint}/v3/plugins/${organization}/${name}/${version}/meta.yaml**.

ii. Custom CPU limit and request:
CodeReady Workspaces does not set CPU limits and requests by default. However, it is possible to configure CPU limits for the **chePlugin** and **cheEditor** types in the **meta.yaml** file or in the devfile in the same way as it done for memory limits.

Example 2.2. The chePlugin che-incubator/typescript/latest

meta.yaml spec section:

```
spec:
 containers:
   - image: docker.io/eclipse/che-remote-plugin-node:next
     name: vscode-typescript
     cpuLimit: 2000m
     cpuRequest: 500m
```

It results in a container with the following CPU limit and request:

CPU limit	2 cores
CPU request	0.5 cores

To set CPU limits and requests globally, use the following dedicated environment variables:

CPU Limit	**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES**
CPU Request	**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES**

See also https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc.

Note that the **LimitRange** object of the OpenShift project may specify defaults for CPU limits and requests set by cluster administrators. To prevent start errors due to resources overrun, limits on application or workspace levels must comply with those settings.

a. Custom requirements for **dockerimage** components
Define the **memoryLimit** and **memoryRequest** attributes of the devfile to configure the memory limit of a **dockerimage** container. CodeReady Workspaces automatically sets the memory request to match the memory limit if it is not specified explicitly.

```
- alias: maven
  type: dockerimage
  image: eclipse/maven-jdk8:latest
  memoryLimit: 1536M
```

b. Custom requirements for **kubernetes** or **openshift** components:
The referenced manifest may define the memory requirements and limits.

1. Add all previously calculated requirements.

Additional resources

- Section 1.3, “Understanding CodeReady Workspaces workspaces architecture” .

2.3. A WORKSPACE EXAMPLE

This section describes a CodeReady Workspaces workspace example.

The following devfile defines the CodeReady Workspaces workspace:

```
apiVersion: 1.0.0
metadata:
  generateName: nodejs-configmap-
projects:
  - name: nodejs-configmap
    source:
      location: "https://github.com/crw-samples/nodejs-configmap.git"
      branch: 12.x
      type: git
components:
  - id: vscode/typescript-language-features/latest
    type: chePlugin
  - mountSources: true
    type: kubernetes
    entrypoints:
      - command:
          - sleep
        args:
          - infinity
    reference: 'https://raw.githubusercontent.com/crw-samples/nodejs-mongodb-
sample/master/kubernetes-manifests/guestbook-app.deployment.yaml'
    alias: guestbook-frontend
```

This table provides the memory requirements for each workspace component:

Table 2.3. Total workspace memory requirement and limit

Pod	Container name	Default memory limit	Default memory request
Workspace	**theia-ide** (default **cheEditor**)	512 MiB	512 MiB
Workspace	**machine-exec** (default **chePlugin**)	128 MiB	32 MiB
Workspace	**vscode-typescript** (**chePlugin**)	512 MiB	512 MiB
Workspace	**nodejs** (**dockerimage**)	1 GiB	512 MiB
JWT Proxy	verifier	128 MiB	128 MiB
	Total	**2.25 GiB**	**1.38 GiB**

- The **theia-ide** and **machine-exec** components are implicitly added to the workspace, even when not included in the devfile.
- The resources required by **machine-exec** are the default for **chePlugin**.
- The resources for **theia-ide** are specifically set in the **cheEditor meta.yaml** to **512 MiB** as **memoryLimit**.
- The Typescript Visual Studio Code extension has also overridden the default memory limits. In its **meta.yaml** file, the limits are explicitly specified to **512 MiB**.
- CodeReady Workspaces is applying the defaults for the **dockerimage** component type: a memory limit of **1 GiB** and a memory request of **512 MiB**.
- The JWT container requires **128 MiB** of memory.

Adding all together results in **1.38 GiB** of memory requests with a **2.25 GiB** limit.

Additional resources

- Chapter 1, *Architecture overview*
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-the-che-installation.adoc
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc
- Section 12.1, “Authenticating users”
- CodeReady Workspaces plug-ins registry repository

CHAPTER 3. CUSTOMIZING THE REGISTRIES

This chapter describes how to build and run custom registries for CodeReady Workspaces.

3.1. UNDERSTANDING THE CODEREADY WORKSPACES REGISTRIES

CodeReady Workspaces uses two registries: the plug-ins registry and the devfile registry. They are static websites publishing the metadata of CodeReady Workspaces plug-ins and devfiles. When built-in offline mode they also include artifacts.

The devfile and plug-in registries run in two separate Pods. Their deployment is part of the CodeReady Workspaces installation.

The devfile and plug-in registries

The devfile registry

The devfile registry holds the definitions of the CodeReady Workspaces stacks. Stacks are available on the CodeReady Workspaces user dashboard when selecting **Create Workspace**. It contains the list of CodeReady Workspaces technological stack samples with example projects. When built-in offline mode it also contains all sample projects referenced in devfiles as **zip** files.

The plug-in registry

The plug-in registry makes it possible to share a plug-in definition across all the users of the same instance of CodeReady Workspaces. When built-in offline mode it also contains all plug-in or extension artifacts.

Additional resources

- Section 3.2, “Building custom registry images”
- Section 3.3, “Running custom registries”

3.2. BUILDING CUSTOM REGISTRY IMAGES

3.2.1. Building a custom devfile registry image

This section describes how to build a custom devfile registry image. The procedure explains how to add a devfile. The image contains all sample projects referenced in devfiles.

Prerequisites

- A running installation of podman or docker.
- Valid content for the devfile to add. See: https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc.

Procedure

1. Clone the devfile registry repository and check out the version to deploy:

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.15-rhel-8
```

2. In the **./dependencies/che-devfile-registry/devfiles/** directory, create a subdirectory ***<devfile-name>***/ and add the **devfile.yaml** and **meta.yaml** files.

Example 3.1. File organization for a devfile

```
./dependencies/che-devfile-registry/devfiles/
└── <devfile-name>
    ├── devfile.yaml
    └── meta.yaml
```

3. Add valid content in the **devfile.yaml** file. For a detailed description of the devfile format, see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc.

4. Ensure that the **meta.yaml** file conforms to the following structure:

Table 3.1. Parameters for a devfile meta.yaml

Attribute	Description
description	Description as it appears on the user dashboard.
displayName	Name as it appears on the user dashboard.
icon	Link to an **.svg** file that is displayed on the user dashboard.
tags	List of tags. Tags typically include the tools included in the stack.
globalMemoryLimit	Optional parameter: the sum of the expected memory consumed by all the components launched by the devfile. This number will be visible on the user dashboard. It is informative and is not taken into account by the CodeReady Workspaces server.

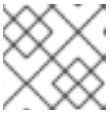
Example 3.2. Example devfile meta.yaml

```
displayName: Rust
description: Rust Stack with Rust 1.39
tags: ["Rust"]
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
globalMemoryLimit: 1686Mi
```

5. Build a custom devfile registry image:

```
$ cd dependencies/che-devfile-registry
$ ./build.sh --organization <my-org> \
```

```
        --registry <my-registry> \
        --tag <my-tag>
```



NOTE

To display full options for the **build.sh** script, use the **--help** parameter.

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc.
- Section 3.3, "Running custom registries".

3.2.2. Building a custom plug-ins registry image

This section describes how to build a custom plug-ins registry image. The procedure explains how to add a plug-in. The image contains plug-ins or extensions metadata.

Prerequisites

- Node.js 12.x
- A running version of yarn. See: Installing Yarn.
- **./node_modules/.bin** is in the **PATH** environment variable.
- A running installation of podman or docker.

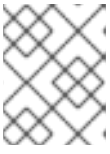
Procedure

1. Clone the plug-ins registry repository and check out the version to deploy:

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.15-rhel-8
```

2. In the **./dependencies/che-plugin-registry/** directory, edit the **che-theia-plugins.yaml** file.
3. Add valid content to the **che-theia-plugins.yaml** file, for detailed information see: https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#adding-a-vs-code-extension-to-the-che-plugin-registry.adoc.
4. Build a custom plug-ins registry image:

```
$ cd dependencies/che-plugin-registry
$ ./build.sh --organization <my-org> \
        --registry <my-registry> \
        --tag <my-tag>
```



NOTE

To display full options for the **build.sh** script, use the **--help** parameter. To include the plug-in binaries in the registry image, add the **--offline** parameter.

5. Observe the contents of **./dependencies/che-plugin-registry/v3/plugins/** present in the container after building the registry. All **meta.yaml** files resulting from a successful plug-ins registry build will be located here.

```
./dependencies/che-plugin-registry/v3/plugins/
├── <publisher>
│   └── <plugin-name>
│       ├── latest
│       │   └── meta.yaml
│       └── latest.txt
```

Additional resources

- Section 3.3, “Running custom registries”.

3.3. RUNNING CUSTOM REGISTRIES

Prerequisites

The **my-plug-in-registry** and **my-devfile-registry** images used in this section are built using the **docker** command. This section assumes that these images are available on the OpenShift cluster where CodeReady Workspaces is deployed.

These images can be then pushed to:

- A public container registry such as **quay.io**, or the DockerHub.
- A private registry.

3.3.1. Deploying registries in OpenShift

Procedure

An OpenShift template to deploy the plug-in registry is available in the **deploy/openshift/** directory of the GitHub repository.

1. To deploy the plug-in registry using the OpenShift template, run the following command:

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-plug-in-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-plugin-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="Always"
```

(1) If installed using crwctl, the default CodeReady Workspaces project is **openshift-workspaces**. The OperatorHub installation method deploys CodeReady Workspaces to the users current project.

2. The devfile registry has an OpenShift template in the **deploy/openshift/** directory of the GitHub repository. To deploy it, run the command:

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-devfile-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-devfile-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="Always"
```

(1) If installed using crwctl, the default CodeReady Workspaces project is **openshift-workspaces**. The OperatorHub installation method deploys CodeReady Workspaces to the users current project.

Verification steps

1. The *<plug-in>* plug-in is available in the plug-in registry.

Example 3.3. Find *<plug-in>* requesting the plug-in registry API.

```
$ URL=$(oc  get route -l app=che,component=plugin-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<plug-in>")'
```

2. The *<devfile>* devfile is available in the devfile registry.

Example 3.4. Find *<devfile>* requesting the devfile registry API.

```
$ URL=$(oc  get route -l app=che,component=devfile-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<devfile>")'
```

3. CodeReady Workspaces server points to the URL of the plug-in registry.

Example 3.5. Compare the value of the CHE_WORKSPACE_PLUGIN__REGISTRY__URL parameter in the che ConfigMap with the URL of the plug-in registry route.

Get the value of the CHE_WORKSPACE_PLUGIN__REGISTRY__URL parameter in the che ConfigMap.

```
$ oc  get cm/che \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_PLUGIN__REGISTRY__URL']" \
  --no-headers
```

Get the URL of the plug-in registry route.

```
$ oc  get route -l app=che,component=plugin-registry \
  -o 'custom-columns=URL: .spec.host' --no-headers
```

4. CodeReady Workspaces server points to the URL of the devfile registry.

Example 3.6. Compare the value of the CHE_WORKSPACE_DEVFILE__REGISTRY__URL parameter in the che ConfigMap with the URL of the devfile registry route.

Get the value of the CHE_WORKSPACE_DEVFILE__REGISTRY__URL parameter in the che ConfigMap.

```
$ oc  get cm/che \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_DEVFILE__REGISTRY__URL']" \
  --no-headers
```

Get the URL of the devfile registry route.

```
$ oc  get route -l app=che,component=devfile-registry \
  -o 'custom-columns=URL: .spec.host' --no-headers
```

5. If the values do not match, update the ConfigMap and restart the CodeReady Workspaces server.

```
$ oc  edit cm/codeready
(...)
$ oc  scale --replicas=0 deployment/codeready
$ oc  scale --replicas=1 deployment/codeready
```

- The plug-ins are available in the:
  - Completion to **chePlugin** components in the **Devfile** tab of a workspace details
- **Plugin** Che-Theia view of a workspace
- The devfiles are available in the **Quick Add** and **Custom Workspace** tab of the **Create Workspace** page on the user dashboard.

3.3.2. Adding a custom plug-in registry in an existing CodeReady Workspaces workspace

The following section describes two methods of adding a custom plug-in registry in an existing CodeReady Workspaces workspace:

- Adding a custom plug-in registry using Command palette - For adding a new custom plug-in registry quickly, with a use of text inputs from Command palette command. This method does not allow a user to edit already existing information, such as plug-in registry URL or name.
- Adding a custom plug-in registry using the **settings.json** file - For adding a new custom plug-in registry and editing of the already existing entries.

3.3.2.1. Adding a custom plug-in registry using Command Palette

Prerequisites

- An instance of CodeReady Workspaces

Procedure

1. In the CodeReady Workspaces IDE, press **F1** to open the Command Palette, or navigate to **View → Find Command** in the top menu.
   The **command palette** can be also activated by pressing **Ctrl**+**Shift**+**p** (or **Cmd**+**Shift**+**p** on macOS).

2. Enter the **Add Registry** command into the search box and pres **Enter** once filled.

3. Enter the registry name and registry URL in next two command prompts.

   - After adding a new plug-in registry, the list of plug-ins in the **Plug-ins** view is refreshed, and if the new plug-in registry is not valid, a user is notified by a warning message.

3.3.2.2. Adding a custom plug-in registry using the settings.json file

The following section describes the use of the main CodeReady Workspaces Settings menu to edit and add a new plug-in registry using the **settings.json** file.

Prerequisites

- An instance of CodeReady Workspaces

Procedure

1. From the main CodeReady Workspaces screen, select **Open Preferences** by pressing **Ctrl**+**,** or using the gear wheel icon on the left bar.

2. Select **Che Plug-ins** and continue by **Edit in setting.json** link.
   The **setting.json** file is displayed.

3. Add a new plug-in registry using the **chePlugins.repositories** attribute as shown below:

```
{
“application.confirmExit”: “never”,
“chePlugins.repositories”: {“test”: “https://test.com”}
}
```

4. Save the changes to add a custom plug-in registry in an existing CodeReady Workspaces workspace.

   - A newly added plug-in validation tool checks the correctness of URL values set in the **chePlugins.repositories** field of the **settings.json** file.

   - After adding a new plug-in registry, the list of plug-ins in the **Plug-ins** view is refreshed, and if the new plug-in registry is not valid, a user is notified by a warning message. This check is also functional for plug-ins added using the Command palette command **Add plugin registry**.

CHAPTER 4. RETRIEVING CODEREADY WORKSPACES LOGS

For information about obtaining various types of logs in CodeReady Workspaces, see the following sections:

- Section 4.1, “Configuring server logging”
- Section 4.2, “Accessing OpenShift events on OpenShift”
- Section 4.4, “Viewing CodeReady Workspaces server logs”
- Section 4.5, “Viewing external service logs”
- Section 4.6, “Viewing the plug-in broker logs”
- Section 4.7, “Collecting logs using crwctl”

4.1. CONFIGURING SERVER LOGGING

It is possible to fine-tune the log levels of individual loggers available in the CodeReady Workspaces server.

The log level of the whole CodeReady Workspaces server is configured globally using the **cheLogLevel** configuration property of the Operator. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#checluster-custom-resource-fields-reference.adoc. To set the global log level in installations not managed by the Operator, specify the **CHE_LOG_LEVEL** environment variable in the **che** ConfigMap.

It is possible to configure the log levels of the individual loggers in the CodeReady Workspaces server using the **CHE_LOGGER_CONFIG** environment variable.

4.1.1. Configuring log levels

The format of the value of the **CHE_LOGGER_CONFIG** property is a list of comma-separated key-value pairs, where keys are the names of the loggers as seen in the CodeReady Workspaces server log output and values are the required log levels.

In Operator-based deployments, the **CHE_LOGGER_CONFIG** variable is specified under the **customCheProperties** of the custom resource.

For example, the following snippet would make the **WorkspaceManager** produce the **DEBUG** log messages.

```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG: "org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

4.1.2. Logger naming

The names of the loggers follow the class names of the internal server classes that use those loggers.

4.1.3. Logging HTTP traffic

It is possible to log the HTTP traffic between the CodeReady Workspaces server and the API server of the Kubernetes or OpenShift cluster. To do that, one has to set the **che.infra.request-logging** logger to the **TRACE** level.

```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

4.2. ACCESSING OPENSHIFT EVENTS ON OPENSHIFT

For high-level monitoring of OpenShift projects, view the OpenShift events that the project performs.

This section describes how to access these events in the OpenShift web console.

Prerequisites

- A running OpenShift web console.

Procedure

1. In the left panel of the OpenShift web console, click the **Home → Events**.
2. To view the list of all events for a particular project, select the project from the list.
3. The details of the events for the current project are displayed.

Additional resources

- For a list of OpenShift events, see Comprehensive List of Events in OpenShift documentation .

4.3. VIEWING THE STATE OF THE CODEREADY WORKSPACES CLUSTER DEPLOYMENT USING OPENSHIFT 4 CLI TOOLS

This section describes how to view the state of the CodeReady Workspaces cluster deployment using OpenShift 4 CLI tools.

Prerequisites

- An instance of Red Hat CodeReady Workspaces running on OpenShift.
- An installation of the OpenShift command-line tool, **oc**.

Procedure

1. Run the following commands to select the **crw** project:

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

2. Run the following commands to get the name and status of the Pods running in the selected project:

```
$ oc get pods
```

3. Check that the status of all the Pods is **Running**.

Example 4.1. Pods with status Running

```
NAME                                  READY     STATUS    RESTARTS   AGE
codeready-8495f4946b-jrzdc            0/1       Running   0          86s
codeready-operator-578765d954-99szc   1/1       Running   0          42m
keycloak-74fbfb9654-g9vp5             1/1       Running   0          4m32s
postgres-5d579c6847-w6wx5             1/1       Running   0          5m14s
```

4. To see the state of the CodeReady Workspaces cluster deployment, run:

```
$ oc logs --tail=10 -f `(oc get pods -o name | grep operator)`
```

Example 4.2. Logs of the Operator:

```
time="2019-07-12T09:48:29Z" level=info msg="Exec successfully completed"
time="2019-07-12T09:48:29Z" level=info msg="Updating eclipse-che CR with status:
provisioned with OpenShift identity provider: true"
time="2019-07-12T09:48:29Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
che"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
custom"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: Deployment,
name: che"
time="2019-07-12T09:48:30Z" level=info msg="Updating eclipse-che CR with status:
CodeReady Workspaces API: Unavailable"
time="2019-07-12T09:48:30Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:30Z" level=info msg="Waiting for deployment che. Default
timeout: 420 seconds"
```

4.4. VIEWING CODEREADY WORKSPACES SERVER LOGS

This section describes how to view the CodeReady Workspaces server logs using the command line.

4.4.1. Viewing the CodeReady Workspaces server logs using the OpenShift CLI

This section describes how to view the CodeReady Workspaces server logs using the OpenShift CLI (command line interface).

Procedure

1. In the terminal, run the following command to get the Pods:

```
$ oc get pods
```

Example

```
$ oc get pods
NAME          READY  STATUS   RESTARTS  AGE
codeready-11-j4w2b    1/1    Running  0        3m
```

2. To get the logs for a deployment, run the following command:

```
$ oc logs <name-of-pod>
```

Example

```
$ oc logs codeready-11-j4w2b
```

4.5. VIEWING EXTERNAL SERVICE LOGS

This section describes how the view the logs from external services related to CodeReady Workspaces server.

4.5.1. Viewing RH-SSO logs

The RH-SSO OpenID provider consists of two parts: Server and IDE. It writes its diagnostics or error information to several logs.

4.5.1.1. Viewing the RH-SSO server logs

This section describes how to view the RH-SSO OpenID provider server logs.

Procedure

1. In the OpenShift Web Console, click **Deployments**.
2. In the **Filter by label** search field, type **keycloak** to see the RH-SSO logs.

. In the **Deployment Configs** section, click the **keycloak** link to open it.

1. In the **History** tab, click the **View log** link for the active RH-SSO deployment.
2. The RH-SSO logs are displayed.

Additional resources

- See the Section 4.4, "Viewing CodeReady Workspaces server logs" for diagnostics and error messages related to the RH-SSO IDE Server.

4.5.1.2. Viewing the RH-SSO client logs on Mozilla Firefox

This section describes how to view the RH-SSO IDE client diagnostics or error information in the Mozilla Firefox **WebConsole**.

Procedure

- Click **Menu** > **WebDeveloper** > **WebConsole**.

4.5.1.3. Viewing the RH-SSO client logs on Google Chrome

This section describes how to view the RH-SSO IDE client diagnostics or error information in the Google Chrome **Console** tab.

Procedure

1. Click **Menu** > **More Tools** > **Developer Tools**.
2. Click the **Console** tab.

4.5.2. Viewing the CodeReady Workspaces database logs

This section describes how to view the database logs in CodeReady Workspaces, such as PostgreSQL server logs.

Procedure

1. In the OpenShift Web Console, click **Deployments**.
2. In the **Find by label** search field, type:
   - **app=che** and press **Enter**
   - **component=postgres** and press **Enter**
     The OpenShift Web Console is searching base on those two keys and displays PostgreSQL logs.
3. Click **postgres** deployment to open it.
4. Click the **View log** link for the active PostgreSQL deployment.
   The OpenShift Web Console displays the database logs.

Additional resources

- Some diagnostics or error messages related to the PostgreSQL server can be found in the active CodeReady Workspaces deployment log. For details to access the active CodeReady Workspaces deployments logs, see the Section 4.4, "Viewing CodeReady Workspaces server logs" section.

4.6. VIEWING THE PLUG-IN BROKER LOGS

This section describes how to view the plug-in broker logs.

The **che-plugin-broker** Pod itself is deleted when its work is complete. Therefore, its event logs are only available while the workspace is starting.

Procedure

To see logged events from temporary Pods:

1. Start a CodeReady Workspaces workspace.
2. From the main OpenShift Container Platform screen, go to **Workload → Pods**.
3. Use the OpenShift terminal console located in the Pod's **Terminal** tab

Verification step

- OpenShift terminal console displays the plug-in broker logs while the workspace is starting

4.7. COLLECTING LOGS USING CRWCTL

It is possible to get all Red Hat CodeReady Workspaces logs from a OpenShift cluster using the **crwctl** tool.

- **crwctl server:deploy** automatically starts collecting Red Hat CodeReady Workspaces servers logs during installation of Red Hat CodeReady Workspaces
- **crwctl server:logs** collects existing Red Hat CodeReady Workspaces server logs
- **crwctl workspace:logs** collects workspace logs

CHAPTER 5. MONITORING CODEREADY WORKSPACES

This chapter describes how to configure CodeReady Workspaces to expose metrics and how to build an example monitoring stack with external tools to process data exposed as metrics by CodeReady Workspaces.

5.1. ENABLING AND EXPOSING CODEREADY WORKSPACES METRICS

This section describes how to enable and expose CodeReady Workspaces metrics.

Procedure

1. Set the **CHE_METRICS_ENABLED=true** environment variable, which will expose the **8087** port as a service on the **che-master** host.

When Red Hat CodeReady Workspaces is installed from the OperatorHub, the environment variable is set automatically if the default **CheCluster** CR is used:


```
spec:
  metrics:
    enable: true
```

5.2. COLLECTING CODEREADY WORKSPACES METRICS WITH PROMETHEUS

This section describes how to use the Prometheus monitoring system to collect, store, and query metrics about CodeReady Workspaces.

Prerequisites

- CodeReady Workspaces is exposing metrics on port **8087**. See Enabling and exposing CodeReady Workspaces metrics.
- Prometheus 2.9.1 or later is running. The Prometheus console is running on port **9090** with a corresponding **service** and **route**. See First steps with Prometheus.

Procedure

- Configure Prometheus to scrape metrics from the **8087** port:

Example 5.1. Prometheus configuration example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval:     5s          (1)
      evaluation_interval: 5s          (2)
    scrape_configs:                    (3)
      - job_name: 'che'
        static_configs:
          - targets: ['[che-host]:8087'] (4)
```

(1) Rate, at which a target is scraped.

(2) Rate, at which recording and alerting rules are re-checked (not used in the system at the moment).

(3) Resources Prometheus monitors. In the default configuration, a single job called **che**, scrapes the time series data exposed by the CodeReady Workspaces server.

(4) Scrape metrics from the **8087** port.

Verification steps

- Use the Prometheus console to query and view metrics.
  Metrics are available at: **http://*<che-server-url>*:9090/metrics**.

  For more information, see Using the expression browser.

Additional resources

- First steps with Prometheus.
- Configuring Prometheus.
- Querying Prometheus.
- Prometheus metric types.

CHAPTER 6. MONITORING THE DEV WORKSPACE OPERATOR

This chapter describes how to configure an example monitoring stack to process metrics exposed by the Dev Workspace operator. You must enable the Dev Workspace operator to follow the instructions in this chapter. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc.

6.1. COLLECTING DEV WORKSPACE OPERATOR METRICS WITH PROMETHEUS

This section describes how to use the Prometheus to collect, store, and query metrics about the Dev Workspace operator.

Prerequisites

- The **devworkspace-controller-metrics** service is exposing metrics on port **8443**.
- The **devworkspace-webhookserver** service is exposing metrics on port **9443**. By default, the service exposes metrics on port **9443**.
- Prometheus 2.26.0 or later is running. The Prometheus console is running on port **9090** with a corresponding **service** and **route**. See First steps with Prometheus.

Procedure

1. Create a **ClusterRoleBinding** to bind the **ServiceAccount** associated with Prometheus to the devworkspace-controller-metrics-reader **ClusterRole**. Without the **ClusterRoleBinding**, you cannot access Dev Workspace metrics because they are protected with role-based access control (RBAC).

Example 6.1. ClusterRole example

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: devworkspace-controller-metrics-reader
rules:
- nonResourceURLs:
  - /metrics
  verbs:
  - get
```

Example 6.2. ClusterRoleBinding example

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: devworkspace-controller-metrics-binding
subjects:
  - kind: ServiceAccount
    name: <ServiceAccount name associated with the Prometheus Pod>
    namespace: <Prometheus namespace>
```

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: devworkspace-controller-metrics-reader
```

2. Configure Prometheus to scrape metrics from the **8443** port exposed by the **devworkspace-controller-metrics** service, and **9443** port exposed by the **devworkspace-webhookserver** service.

Example 6.3. Prometheus configuration example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval:     5s          (1)
      evaluation_interval: 5s          (2)
    scrape_configs:                    (3)
      - job_name: 'DevWorkspace'
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-controller-metrics:8443'] (4)
      - job_name: 'DevWorkspace webhooks'
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-webhookserver:9443'] (5)
```

(1) Rate at which a target is scraped.

(2) Rate at which recording and alerting rules are re-checked.

(3) Resources that Prometheus monitors. In the default configuration, two jobs (**DevWorkspace** and **DevWorkspace webhooks**), scrape the time series data exposed by the **devworkspace-controller-metrics** and **devworkspace-webhookserver** services.

(4) Scrape metrics from the **8443** port.

(5) Scrape metrics from the **9443** port.

Verification steps

- Use the Prometheus console to view targets and metrics.
  For more information, see Using the expression browser.

Additional resources

- First steps with Prometheus.
- Configuring Prometheus.
- Querying Prometheus.
- Prometheus metric types.

6.2. DEV WORKSPACE-SPECIFIC METRICS

This section describes the Dev Workspace-specific metrics exposed by the **devworkspace-controller-metrics** service.

Table 6.1. Metrics

Name	Type	Description	Labels
devworkspace_started_total	Counter	Number of Dev Workspace starting events.	**source**, **routingclass**
devworkspace_started_success_total	Counter	Number of Dev Workspaces successfully entering the **Running** phase.	**source**, **routingclass**
devworkspace_fail_total	Counter	Number of failed Dev Workspaces.	**source**, **reason**
devworkspace_startup_time	Histogram	Total time taken to start a Dev Workspace, in seconds.	**source**, **routingclass**

Table 6.2. Labels

Name	Description	Values
source	The **controller.devfile.io/devworkspace-source** label of the Dev Workspace.	**string**
routingclass	The **spec.routingclass** of the Dev Workspace.	**"basic|cluster|cluster-tls|web-terminal"**

Name	Description	Values
reason	The workspace startup failure reason.	**"BadRequest|InfrastructureFailure|Unknown"**

Table 6.3. Startup failure reasons

Name	Description
BadRequest	Startup failure due to an invalid devfile used to create a Dev Workspace.
InfrastructureFailure	Startup failure due to the following errors: **CreateContainerError**, **RunContainerError**, **FailedScheduling**, **FailedMount**.
Unknown	Unknown failure reason.

CHAPTER 7. TRACING CODEREADY WORKSPACES

Tracing helps gather timing data to troubleshoot latency problems in microservice architectures and helps to understand a complete transaction or workflow as it propagates through a distributed system. Every transaction may reflect performance anomalies in an early phase when new services are being introduced by independent teams.

Tracing the CodeReady Workspaces application may help analyze the execution of various operations, such as workspace creations, workspace startup, breaking down the duration of sub-operations executions, helping finding bottlenecks and improve the overall state of the platform.

Tracers live in applications. They record timing and metadata about operations that take place. They often instrument libraries, so that their use is indiscernible to users. For example, an instrumented web server records when it received a request and when it sent a response. The trace data collected is called a **span**. A span has a context that contains information such as trace and span identifiers and other kinds of data that can be propagated down the line.

7.1. TRACING API

CodeReady Workspaces utilizes OpenTracing API - a vendor-neutral framework for instrumentation. This means that if a developer wants to try a different tracing back end, then rather than repeating the whole instrumentation process for the new distributed tracing system, the developer can simply change the configuration of the tracer back end.

7.2. TRACING BACK END

By default, CodeReady Workspaces uses Jaeger as the tracing back end. Jaeger was inspired by Dapper and OpenZipkin, and it is a distributed tracing system released as open source by Uber Technologies. Jaeger extends a more complex architecture for a larger scale of requests and performance.

7.3. INSTALLING THE JAEGER TRACING TOOL

The following sections describe the installation methods for the Jaeger tracing tool. Jaeger can then be used for gathering metrics in CodeReady Workspaces.

Installation methods available:

- Section 7.3.1, “Installing Jaeger using OperatorHub on OpenShift 4”
- Section 7.3.2, “Installing Jaeger using CLI on OpenShift 4”

For tracing a CodeReady Workspaces instance using Jaeger, version 1.12.0 or above is required. For additional information about Jaeger, see the Jaeger website.

7.3.1. Installing Jaeger using OperatorHub on OpenShift 4

This section provide information about using Jaeger tracing tool for testing an evaluation purposes in production.

To install the Jaeger tracing tool from the OperatorHub interface in OpenShift Container Platform, follow the instructions below.

Prerequisites

- The user is logged in to the OpenShift Container Platform Web Console.
- A CodeReady Workspaces instance is available in a project.

Procedure

1. Open the OpenShift Container Platform console.
2. From the left menu of the main OpenShift Container Platform screen, navigate to **Operators → OperatorHub**.
3. In the **Search by keyword** search bar, type **Jaeger Operator**.
4. Click the **Jaeger Operator** tile.
5. Click the **Install** button in the **Jaeger Operator** pop-up window.
6. Select the installation method: **A specific project on the cluster** where the CodeReady Workspaces is deployed and leave the rest in its default values.
7. Click the **Subscribe** button.
8. From the left menu of the main OpenShift Container Platform screen, navigate to the **Operators → Installed Operators** section.
9. Red Hat CodeReady Workspaces is displayed as an Installed Operator, as indicated by the **InstallSucceeded** status.
10. Click the **Jaeger Operator** name in the list of installed Operators.
11. Navigate to the **Overview** tab.
12. In the Conditions sections at the bottom of the page, wait for this message: **install strategy completed with no errors**.
13. **Jaeger Operator** and additional **Elasticsearch Operator** is installed.
14. Navigate to the **Operators → Installed Operators** section.
15. Click **Jaeger Operator** in the list of installed Operators.
16. The **Jaeger Cluster** page is displayed.
17. In the lower left corner of the window, click **Create Instance**
18. Click **Save**.
19. OpenShift creates the Jaeger cluster **jaeger-all-in-one-inmemory**.
20. Follow the steps in Enabling metrics collection to finish the procedure.

7.3.2. Installing Jaeger using CLI on OpenShift 4

This section provide information about using Jaeger tracing tool for testing an evaluation purposes.

To install the Jaeger tracing tool from a CodeReady Workspaces project in OpenShift Container Platform, follow the instructions in this section.

Prerequisites

- The user is logged in to the OpenShift Container Platform web console.
- A instance of CodeReady Workspaces in an OpenShift Container Platform cluster.

Procedure

1. In the CodeReady Workspaces installation project of the OpenShift Container Platform cluster, use the **oc** client to create a new application for the Jaeger deployment.

```
$ oc new-app -f / ${CHE_LOCAL_GIT_REPO}/deploy/openshift/templates/jaeger-all-in-one-template.yml:

--> Deploying template "<project_name>/jaeger-template-all-in-one" for "/home/user/crw-projects/crw/deploy/openshift/templates/jaeger-all-in-one-template.yml" to project <project_name>

    Jaeger (all-in-one)
    ---------
    Jaeger Distributed Tracing Server (all-in-one)

    * With parameters:
      * Jaeger Service Name=jaeger
      * Image version=latest
      * Jaeger Zipkin Service Name=zipkin

--> Creating resources ...
    deployment.apps "jaeger" created
    service "jaeger-query" created
    service "jaeger-collector" created
    service "jaeger-agent" created
    service "zipkin" created
    route.route.openshift.io "jaeger-query" created
--> Success
    Access your application using the route: 'jaeger-query-<project_name>.apps.ci-ln-whx0352-d5d6b.origin-ci-int-aws.dev.rhcloud.com'
    Run 'oc status' to view your app.
```

2. Using the **Workloads → Deployments** from the left menu of main OpenShift Container Platform screen, monitor the Jaeger deployment until it finishes successfully.
3. Select **Networking → Routes** from the left menu of the main OpenShift Container Platform screen, and click the URL link to access the Jaeger dashboard.
4. Follow the steps in Enabling metrics collection to finish the procedure.

7.4. ENABLING METRICS COLLECTION

Prerequisites

- Installed Jaeger v1.12.0 or above. See instructions at Section 7.3, “Installing the Jaeger tracing tool”

Procedure

For Jaeger tracing to work, enable the following environment variables in your CodeReady Workspaces deployment:

```
# Activating CodeReady Workspaces tracing modules
CHE_TRACING_ENABLED=true

# Following variables are the basic Jaeger client library configuration.
JAEGER_ENDPOINT="http://jaeger-collector:14268/api/traces"

# Service name
JAEGER_SERVICE_NAME="che-server"

# URL to remote sampler
JAEGER_SAMPLER_MANAGER_HOST_PORT="jaeger:5778"

# Type and param of sampler (constant sampler for all traces)
JAEGER_SAMPLER_TYPE="const"
JAEGER_SAMPLER_PARAM="1"

# Maximum queue size of reporter
JAEGER_REPORTER_MAX_QUEUE_SIZE="10000"
```

To enable the following environment variables:

1. In the **yaml** source code of the CodeReady Workspaces deployment, add the following configuration variables under **spec.server.customCheProperties**.

```
customCheProperties:
    CHE_TRACING_ENABLED: 'true'
    JAEGER_SAMPLER_TYPE: const
    DEFAULT_JAEGER_REPORTER_MAX_QUEUE_SIZE: '10000'
    JAEGER_SERVICE_NAME: che-server
    JAEGER_ENDPOINT: 'http://jaeger-collector:14268/api/traces'
    JAEGER_SAMPLER_MANAGER_HOST_PORT: 'jaeger:5778'
    JAEGER_SAMPLER_PARAM: '1'
```

2. Edit the **JAEGER_ENDPOINT** value to match the name of the Jaeger collector service in your deployment.
   From the left menu of the main OpenShift Container Platform screen, obtain the value of JAEGER_ENDPOINT by navigation to **Networking → Services**. Alternatively, execute the following **oc** command:

```
$ oc get services
```

   The requested value is included in the service name that contains the **collector** string.

Additional resources

- For additional information about custom environment properties and how to define them in CheCluster Custom Resource, see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc.
- For custom configuration of Jaeger, see the list of Jaeger client environment variables.

7.5. VIEWING CODEREADY WORKSPACES TRACES IN JAEGER UI

This section demonstrates how to use the Jaeger UI to overview traces of CodeReady Workspaces operations.

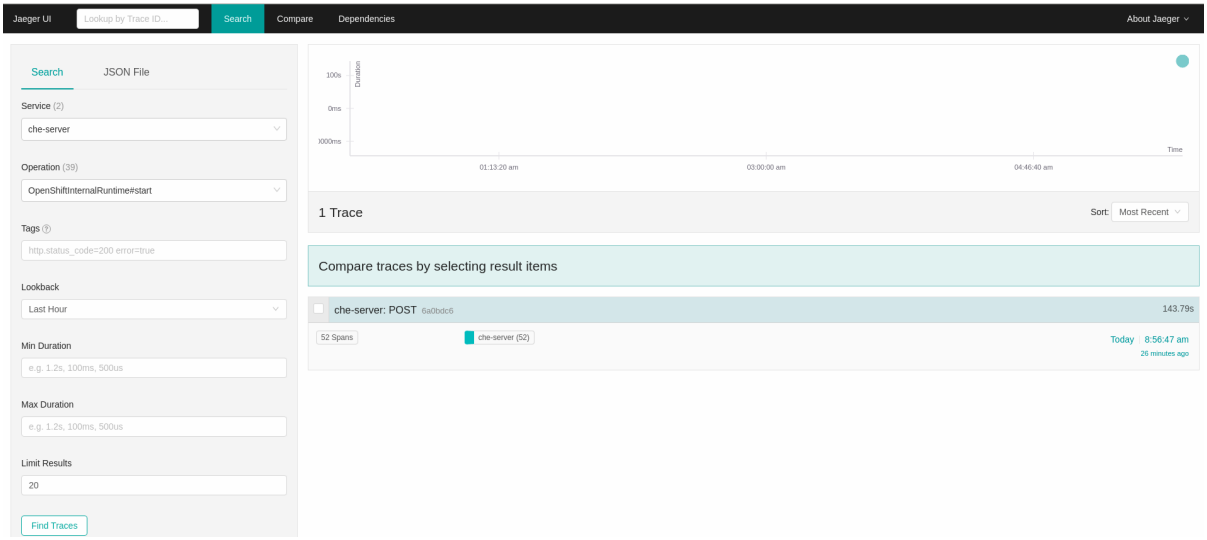
Procedure

In this example, the CodeReady Workspaces instance has been running for some time and one workspace start has occurred.

To inspect the trace of the workspace start:

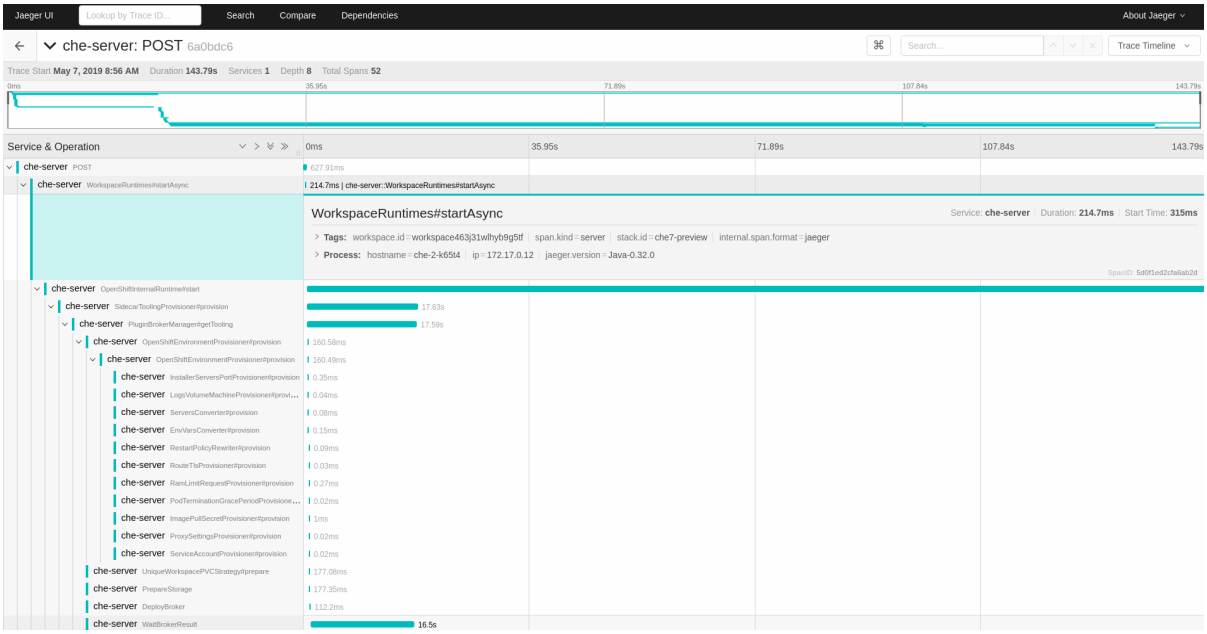
1. In the **Search** panel on the left, filter spans by the operation name (span name), tags, or time and duration.

Figure 7.1. Using Jaeger UI to trace CodeReady Workspaces



2. Select the trace to expand it and show the tree of nested spans and additional information about the highlighted span, such as tags or durations.

Figure 7.2. Expanded tracing tree



7.6. CODEREADY WORKSPACES TRACING CODEBASE OVERVIEW AND EXTENSION GUIDE

The core of the tracing implementation for CodeReady Workspaces is in the **che-core-tracing-core** and **che-core-tracing-web** modules.

All HTTP requests to the tracing API have their own trace. This is done by **TracingFilter** from the OpenTracing library, which is bound for the whole server application. Adding a **@Traced** annotation to methods causes the **TracingInterceptor** to add tracing spans for them.

7.6.1. Tagging

Spans may contain standard tags, such as operation name, span origin, error, and other tags that may help users with querying and filtering spans. Workspace-related operations (such as starting or stopping workspaces) have additional tags, including **userId**, **workspaceID**, and **stackId**. Spans created by **TracingFilter** also have an HTTP status code tag.

Declaring tags in a traced method is done statically by setting fields from the **TracingTags** class:

```
TracingTags.WORKSPACE_ID.set(workspace.getId());
```

TracingTags is a class where all commonly used tags are declared, as respective **AnnotationAware** tag implementations.

Additional resources

For more information about how to use Jaeger UI, visit Jaeger documentation: Jaeger Getting Started Guide.

CHAPTER 8. BACKUP AND RECOVERY

Backing up CodeReady Workspaces involves a combination of the following processes that back up different data:

- Use the CodeReady Workspaces Operator and a crwctl.

NOTE

Use the internal backup server to test this process.

- Use backups of persistent volumes to back up and restore the source code stored in users' workspaces.

TIP

Whether or not you implement backups of persistent volumes, advise users to commit and push their changes to avoid losing their work.

- Use backups of the external PostgreSQL database to back up and restore persisting data about the state of CodeReady Workspaces.

8.1. SUPPORTED RESTIC-COMPATIBLE BACKUP SERVERS

CodeReady Workspaces uses the CodeReady Workspaces Operator and integrated Restic to back up and restore CodeReady Workspaces instances from backup snapshots on a configured backup server. The CodeReady Workspaces Operator automates the creation of a Restic backup repository on the backup server. To back up data, the CodeReady Workspaces Operator gathers the data required for a backup snapshot and uses Restic to create and manage the snapshot. To restore data, the CodeReady Workspaces Operator uses Restic to retrieve and decrypt the snapshot, and then the CodeReady Workspaces Operator applies the retrieved data to perform the recovery.

CodeReady Workspaces can use the following backup servers that are compatible with the integrated Restic:

SFTP

See the documentation for the SFTP server solution you plan to use (OpenSSH or a derived commercial product) and the Restic Docs on SFTP.

Amazon S3

See the documentation for Amazon S3 (or the chosen S3 API compatible storage) and the Restic Docs on Amazon S3.

REST

See the README for Rest Server and the Restic Docs on Rest Server.

For testing the backing up and restoring of CodeReady Workspaces instances, CodeReady Workspaces offers the internal backup server.

Additional resources

- Restic Documentation
- restic · Backups done right!

- OpenSSH.com
- How to configure an sftp server with restricted chroot users with ssh keys

8.2. BACKING UP OF CODEREADY WORKSPACES INSTANCES TO AN SFTP BACKUP SERVER

You can send backups of CodeReady Workspaces instances to an SFTP backup server with custom resources or crwctl:

- Section 8.2.1, "Backing up a CodeReady Workspaces instance to an SFTP backup server by using custom resources"
- Section 8.2.2, "Backing up a CodeReady Workspaces instance to an SFTP backup server by using crwctl"

8.2.1. Backing up a CodeReady Workspaces instance to an SFTP backup server by using custom resources

Backing up a CodeReady Workspaces instance to an SFTP backup server by using custom resources requires two custom objects:

- First you create a custom object to configure CodeReady Workspaces to use an SFTP backup server.
- Then you create a custom object to make and send a backup snapshot of a CodeReady Workspaces instance to the configured SFTP backup server.

8.2.1.1. Configuring CodeReady Workspaces with custom resources to use an SFTP backup server

To configure CodeReady Workspaces to use an SFTP backup server:

Prerequisites

- Configured SFTP backup server. See Section 8.1, "Supported Restic-compatible backup servers".

Procedure

1. Create a Secret containing the **repo-password** key with a password:

```
apiVersion: v1
kind: Secret
metadata:
  name: backup-encryption-password-secret
  namespace: eclipse-che
type: Opaque
stringData:
  repo-password: <password>
```

(The CodeReady Workspaces Operator will set up this password for the backup repository that the CodeReady Workspaces Operator will create from this custom object on the backup server.)

WARNING

The backup repository password is used to encrypt the backup data. If you lose this password, you will lose the backup data.

2. Create a Secret (for example, **name: ssh-key-secret**) containing the **ssh-privatekey** key with a private SSH key for logging in to the SFTP server without a password.

3. Create the **CheBackupServerConfiguration** custom object:

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
  namespace: openshift-workspaces
spec: (1)
  sftp:
    username: <username> (2)
    hostname: <hostname> (3)
    port: 1234 (4)
    repositoryPath: red-hat-codeready-workspaces-backups (5)
    repositoryPasswordSecretRef: backup-encryption-password-secret (6)
    sshKeySecretRef: ssh-key-secret (7)
```

(1) Must only contain one section (such as **sftp**).

(2) User name on the remote server to log in using the SSH protocol.

(3) Remote server hostname.

(4) Optional property that specifies the port on which an SFTP server is running. The default value is **22**.

(5) Absolute or relative path on the server where backup snapshots are stored.

(6) Secret created in step 1.

(7) Secret created in step 2.

4. Optional: To configure multiple backup servers, create a separate **CheBackupServerConfiguration** custom object for each backup server.

IMPORTANT

The CodeReady Workspaces Operator automatically backs up the CodeReady Workspaces instance before every CodeReady Workspaces update, permitting rollback to previous CodeReady Workspaces version if needed. If you configure only one backup server, that backup server is automatically used for pre-update backups by default. If you configure multiple backup servers, you must add the **che.eclipse.org/backup-before-update: true** annotation to the custom object of only one of them to specify it as the default backup server for pre-update backups. (If you don't add this annotation for one of multiple backup servers, or if you add this annotation for multiple backup servers, then the CodeReady Workspaces Operator defaults to using the internal backup server for pre-update backups.)

8.2.1.2. Backing up a CodeReady Workspaces instance to an SFTP backup server by using the CheClusterBackup custom object

You can use a **CheClusterBackup** custom object to make a backup snapshot of a CodeReady Workspaces instance and to send the snapshot to the configured backup server. To create each and every backup snapshot requires a new **CheClusterBackup** custom object; that is, editing an already consumed **CheClusterBackup** custom object, during or after backing up, has no effect.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Configured backup server. See Section 8.1, "Supported Restic-compatible backup servers" .
- Created **CheBackupServerConfiguration** custom object. See the previous section of this guide.

Procedure

1. Create the **CheClusterBackup** custom object, which creates a backup snapshot:

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: red-hat-codeready-workspaces-backup
  namespace: openshift-workspaces
spec:
  backupServerConfigRef: backup-server-configuration (1)
  useInternalBackupServer: false (2)
```

(1) Name of the **CheBackupServerConfiguration** custom object defining what backup server to use.

2 Configures the Operator through this custom resource to use the CodeReady Workspaces-managed internal backup server or an administrator-managed external

TIP

If you intend to reuse a **name** for **CheClusterBackup** custom objects, first delete any existing custom object with the same **name**. To delete it on the command line, use oc:

```
$ oc delete CheClusterBackup <name> -n openshift-workspaces
```

2. Read the **status** section of the **CheClusterBackup** custom object to verify the backup process, for example:

```
status:
  message: 'Backup is in progress. Start time: <timestamp>' 1
  stage: Collecting CodeReady Workspaces installation data 2
  state: InProgress 3
```

1 Summary of the latest state of the backup process.

2 Current stage of the backup process in a human-readable format.

3 Backup process state: **InProgress** or **Succeeded** or **Failed**.

The CodeReady Workspaces instance is backed up in a snapshot when **state** is **Succeeded**:

```
status:
  ...
  message: >-
    Backup successfully finished at 2021-12-03 10:07:51.151090621 +0000 UTC
    m=+999.553272281 1
  snapshotId: 9f0adce2 2
  state: Succeeded 3
```

1 Summary of the latest state of the backup process.

2 ID of the created backup snapshot. This field appears only when **state** is **Succeeded**.

3 Backup process state: **Succeeded** (or **Failed**).

8.2.2. Backing up a CodeReady Workspaces instance to an SFTP backup server by using crwctl

You can use crwctl to make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to a configured SFTP backup server. To do so, enter crwctl with the command-line options or set the environment variables.

8.2.2.1. Backing up a CodeReady Workspaces instance to an SFTP backup server by using crwctl with command-line options

To make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to a configured SFTP backup server, run crwctl with the command-line options.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Installed **crwctl**.
- Configured SFTP backup server. See Section 8.1, "Supported Restic-compatible backup servers".

Procedure

1. Enter the **crwctl server:backup** command with the following arguments:

```
$ crwctl server:backup \
--repository-url=<repository_url> \ 1
--repository-password=<repository_password> \ 2
--ssh-key-file=<ssh_key_file> 3
```

1. Backup repository URL as an argument using the **--repository-url** or **-r** option. Syntax for the backup repository URL: **sftp:*<user>*@*<host>*:/*<repository_path_on_sftp_server>***. An example of a repository path on the SFTP server: **/srv/restic-repo**. For more details about repository URL syntax, see Restic Documentation.
2. Backup repository password as an argument using the **--repository-password** or **-p** option.
3. Path to a private SSH key file for authenticating on the SFTP server.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--repository-url** and **--repository-password** options with the **crwctl server:backup** command just once and omit them onward when entering **crwctl server:backup** or **crwctl server:restore**.

2. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling backup...OK
 ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.2.2.2. Backing up a CodeReady Workspaces instance to an SFTP backup server by using crwctl and a CheBackupServerConfiguration custom object

You can back up a CodeReady Workspaces instance by entering the **name** of a **CheBackupServerConfiguration** custom object as an argument with the **crwctl server:backup** command.

Prerequisites

- Installed **crwctl**.
- Configured backup server. See Section 8.1, "Supported Restic-compatible backup servers".
- Created **CheBackupServerConfiguration** custom object.

Procedure

1. Run the following command on a command line:

```
$ crwctl server:backup \
--backup-server-config-name=<name_of_CheBackupServerConfiguration> (1)
```

(1) This option points crwctl to a **CheBackupServerConfiguration** custom object. You can find the **name** of the **CheBackupServerConfiguration** custom object under **metadata** in the custom object.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--backup-server-config-name** option with the **crwctl server:backup** command just once and omit this option onward when entering **crwctl server:backup**.

2. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling backup...OK
 ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.2.2.3. Configuring crwctl for an SFTP backup server with environment variables

As an alternative to using command-line options for **crwctl server:backup**, you can configure crwctl for an SFTP backup server with environment variables. This enables you to run **crwctl server:backup** without having to enter those options with it.

Prerequisites

- Installed **crwctl**.

- Configured SFTP backup server. See Section 8.1, “Supported Restic-compatible backup servers”.

Procedure

Which environment variables you need to set depends on your use case:

- If you do not have a **CheBackupServerConfiguration** custom object yet, then you can set the following environment variables:
  - **BACKUP_REPOSITORY_URL** for the backup repository.
  - **BACKUP_REPOSITORY_PASSWORD** for the backup repository password.
  - One of the two environment variables for the SSH key (for logging in without a password):
    - **SSH_KEY_FILE** for the path to the SSH key file.
    - **SSH_KEY** for the SSH key.
- If you already have a **CheBackupServerConfiguration** custom object, you can set the following environment variable that will point crwctl to that **CheBackupServerConfiguration** custom object:
  - **BACKUP_SERVER_CONFIG_NAME** for the **name** of a **CheBackupServerConfiguration** custom object to point crwctl to. (You can find **name** under **metadata** in the custom object.)

8.3. BACKING UP OF CODEREADY WORKSPACES INSTANCES TO AMAZON S3

You can send backups of CodeReady Workspaces instances to Amazon S3 (or S3 API compatible storage) with custom resources or crwctl:

- Section 8.3.1, “Backing up a CodeReady Workspaces instance to Amazon S3 by using custom resources”
- Section 8.3.2, “Backing up a CodeReady Workspaces instance to Amazon S3 by using crwctl”

8.3.1. Backing up a CodeReady Workspaces instance to Amazon S3 by using custom resources

Backing up a CodeReady Workspaces instance to Amazon S3 (or S3 API compatible storage) by using custom resources requires two custom objects:

- First you create a custom object to configure CodeReady Workspaces to use Amazon S3.
- Then you create a custom object to make and send a backup snapshot of a CodeReady Workspaces instance to the configured Amazon S3.

8.3.1.1. Configuring CodeReady Workspaces with custom resources to use Amazon S3

To configure CodeReady Workspaces to use Amazon S3:

Prerequisites

- Configured Amazon S3. See Section 8.1, “Supported Restic-compatible backup servers” .

Procedure

1. Create a Secret containing the **repo-password** key with a password:

```
apiVersion: v1
kind: Secret
metadata:
  name: backup-encryption-password-secret
  namespace: eclipse-che
type: Opaque
stringData:
  repo-password: <password>
```

(The CodeReady Workspaces Operator will set up this password for the backup repository that the CodeReady Workspaces Operator will create from this custom object on the backup server.)

WARNING

The backup repository password is used to encrypt the backup data. If you lose this password, you will lose the backup data.

2. Create a Secret (for example, **name: aws-user-credentials-secret**) containing:
   - **awsAccessKeyId** key with the AWS access key ID of the user
   - **awsSecretAccessKey** key with the AWS secret access key of the user
3. Create the **CheBackupServerConfiguration** custom object:

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
  namespace: openshift-workspaces
spec: (1)
  awss3:
    protocol: https (2)
    hostname: s3.amazonaws.com (3)
    port: 443 (4)
    repositoryPath: red-hat-codeready-workspaces-backups (5)
    repositoryPasswordSecretRef: backup-encryption-password-secret (6)
    awsAccessKeySecretRef: aws-user-credentials-secret (7)
```

(1) Must only contain one section (such as **awss3**).

(2) Optional property that specifies the protocol to be used. **https** is the default value; **http** is a permitted value.

3. Optional property that specifies the S3 hostname. The default value is **s3.amazonaws.com**.
4. Optional property that specifies the port on which the backup server is running. The default value is **443**.
5. Name of the bucket resource that stores the backup snapshots. You must manually create the bucket resource before configuring it here.
6. Secret created in step 1.
7. Secret created in step 2.

4. Optional: To configure multiple backup servers, create a separate **CheBackupServerConfiguration** custom object for each backup server.

IMPORTANT

The CodeReady Workspaces Operator automatically backs up the CodeReady Workspaces instance before every CodeReady Workspaces update, permitting rollback to previous CodeReady Workspaces version if needed. If you configure only one backup server, that backup server is automatically used for pre-update backups by default. If you configure multiple backup servers, you must add the **che.eclipse.org/backup-before-update: true** annotation to the custom object of only one of them to specify it as the default backup server for pre-update backups. (If you don't add this annotation for one of multiple backup servers, or if you add this annotation for multiple backup servers, then the CodeReady Workspaces Operator defaults to using the internal backup server for pre-update backups.)

8.3.1.2. Backing up a CodeReady Workspaces instance to Amazon S3 by using the CheClusterBackup custom object

You can use a **CheClusterBackup** custom object to make a backup snapshot of a CodeReady Workspaces instance and to send the snapshot to the configured backup server. To create each and every backup snapshot requires a new **CheClusterBackup** custom object; that is, editing an already consumed **CheClusterBackup** custom object, during or after backing up, has no effect.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Configured backup server. See Section 8.1, "Supported Restic-compatible backup servers" .
- Created **CheBackupServerConfiguration** custom object. See the previous section of this guide.

Procedure

1. Create the **CheClusterBackup** custom object, which creates a backup snapshot:

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: red-hat-codeready-workspaces-backup
  namespace: openshift-workspaces
spec:
  backupServerConfigRef: backup-server-configuration (1)
  useInternalBackupServer: false (2)
```

(1) Name of the **CheBackupServerConfiguration** custom object defining what backup server to use.

(2) Configures the Operator through this custom resource to use the CodeReady Workspaces-managed internal backup server or an administrator-managed external backup server (SFTP, Amazon S3 or S3 API compatible storage, or REST).

TIP

If you intend to reuse a **name** for **CheClusterBackup** custom objects, first delete any existing custom object with the same **name**. To delete it on the command line, use oc:

```
$ oc delete CheClusterBackup <name> -n openshift-workspaces
```

2. Read the **status** section of the **CheClusterBackup** custom object to verify the backup process, for example:

```
status:
  message: 'Backup is in progress. Start time: <timestamp>' (1)
  stage: Collecting CodeReady Workspaces installation data (2)
  state: InProgress (3)
```

(1) Summary of the latest state of the backup process.

(2) Current stage of the backup process in a human-readable format.

(3) Backup process state: **InProgress** or **Succeeded** or **Failed**.

The CodeReady Workspaces instance is backed up in a snapshot when **state** is **Succeeded**:

```
status:
  ...
  message: >-
    Backup successfully finished at 2021-12-03 10:07:51.151090621 +0000 UTC
    m=+999.553272281 (1)
  snapshotId: 9f0adce2 (2)
  state: Succeeded (3)
```

1. Summary of the latest state of the backup process.
2. ID of the created backup snapshot. This field appears only when **state** is **Succeeded**.
3. Backup process state: **Succeeded** (or **Failed**).

8.3.2. Backing up a CodeReady Workspaces instance to Amazon S3 by using crwctl

You can use crwctl to make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to the configured Amazon S3 (or S3 API compatible storage). To do so, enter crwctl with the command-line options or set the environment variables.

8.3.2.1. Backing up a CodeReady Workspaces instance to Amazon S3 by using crwctl with command-line options

To make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to the configured Amazon S3, run crwctl with the command-line options.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Installed **crwctl**.
- Configured Amazon S3. See Section 8.1, "Supported Restic-compatible backup servers" .

Procedure

1. Enter the **crwctl server:backup** command with the following arguments:

```
$ crwctl server:backup \
--repository-url=<repository_url> \ 1
--repository-password=<repository_password> \ 2
--aws-access-key-id=<aws-access-key-id> \ 3
--aws-secret-access-key==<aws-secret-access-key> 4
```

1. Backup repository URL as an argument using the **--repository-url** or **-r** option. Syntax for the backup repository URL: **s3:s3.amazonaws.com/*<bucket_name>***. For more details about repository URL syntax, see Restic Documentation.
2. Backup repository password as an argument using the **--repository-password** or **-p** option.
3. User's AWS access key ID.

4 User's AWS secret access key.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--repository-url** and **--repository-password** options with the **crwctl server:backup** command just once and omit them onward when entering **crwctl server:backup** or **crwctl server:restore**.

2. Verify the output of the entered command. For example:

```
...
  ✔ Scheduling backup...OK
  ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.3.2.2. Backing up a CodeReady Workspaces instance to Amazon S3 by using crwctl and a CheBackupServerConfiguration custom object

You can back up a CodeReady Workspaces instance by entering the **name** of a **CheBackupServerConfiguration** custom object as an argument with the **crwctl server:backup** command.

Prerequisites

- Installed **crwctl**.
- Configured backup server. See Section 8.1, "Supported Restic-compatible backup servers".
- Created **CheBackupServerConfiguration** custom object.

Procedure

1. Run the following command on a command line:

```
$ crwctl server:backup \
--backup-server-config-name=<name_of_CheBackupServerConfiguration> 1
```

1 This option points crwctl to a **CheBackupServerConfiguration** custom object. You can find the **name** of the **CheBackupServerConfiguration** custom object under **metadata** in the custom object.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--backup-server-config-name** option with the **crwctl server:backup** command just once and omit this option onward when entering **crwctl server:backup**.

2. Verify the output of the entered command. For example:

```
...
  ✔ Scheduling backup...OK
  ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.3.2.3. Configuring crwctl with environment variables to use Amazon S3

As an alternative to using command-line options for **crwctl server:backup**, you can configure crwctl for Amazon S3 with environment variables. This enables you to run **crwctl server:backup** without having to enter those options with it.

Prerequisites

- Installed **crwctl**.
- Configured Amazon S3. See Section 8.1, “Supported Restic-compatible backup servers” .

Procedure

Which environment variables you need to set depends on your use case:

- If you do not have a **CheBackupServerConfiguration** custom object yet, then you can set the following environment variables:
  - **BACKUP_REPOSITORY_URL** for the backup repository.
  - **BACKUP_REPOSITORY_PASSWORD** for the backup repository password.
  - **AWS_ACCESS_KEY_ID** for the user’s AWS access key ID.
  - **AWS_SECRET_ACCESS_KEY** for the user’s AWS secret access key.
- If you already have a **CheBackupServerConfiguration** custom object, you can set the following environment variable that will point crwctl to that **CheBackupServerConfiguration** custom object:
  - **BACKUP_SERVER_CONFIG_NAME** for the **name** of a **CheBackupServerConfiguration** custom object to point crwctl to. (You can find **name** under **metadata** in the custom object.)

8.4. BACKING UP OF CODEREADY WORKSPACES INSTANCES TO A REST BACKUP SERVER

You can send backups of CodeReady Workspaces instances to a REST backup server with custom resources or crwctl:

- Section 8.4.1, “Backing up a CodeReady Workspaces instance to a REST backup server by using custom resources”
- Section 8.4.2, “Backing up a CodeReady Workspaces instance to a REST backup server by using crwctl”

8.4.1. Backing up a CodeReady Workspaces instance to a REST backup server by using custom resources

Backing up a CodeReady Workspaces instance to a REST backup server by using custom resources requires two custom objects:

- First you create a custom object to configure CodeReady Workspaces to use a REST backup server.
- Then you create a custom object to make and send a backup snapshot of a CodeReady Workspaces instance to the configured REST backup server.

8.4.1.1. Configuring CodeReady Workspaces with custom resources to use a REST backup server

To configure CodeReady Workspaces to use a REST backup server:

Prerequisites

- Configured REST backup server. See Section 8.1, "Supported Restic-compatible backup servers".

Procedure

1. Create a Secret containing the **repo-password** key with a password:

```
apiVersion: v1
kind: Secret
metadata:
  name: backup-encryption-password-secret
  namespace: eclipse-che
type: Opaque
stringData:
  repo-password: <password>
```

(The CodeReady Workspaces Operator will set up this password for the backup repository that the CodeReady Workspaces Operator will create from this custom object on the backup server.)

WARNING

The backup repository password is used to encrypt the backup data. If you lose this password, you will lose the backup data.

2. Optional: Create a Secret (for example, **name: rest-server-auth-secret**) containing the **username** and **password** keys for the REST server user credentials.

NOTE

Skip this step if not using the REST server user credentials.

3. Create the **CheBackupServerConfiguration** custom object:

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
  namespace: openshift-workspaces
spec: (1)
  rest:
    protocol: https (2)
    hostname: <hostname> (3)
    port: 1234 (4)
    repositoryPath: red-hat-codeready-workspaces-backups (5)
    repositoryPasswordSecretRef: backup-encryption-password-secret (6)
    credentialsSecretRef: rest-server-auth-secret (7)
```

1. Must only contain one section (such as **rest**).
2. Optional property that specifies the protocol to be used: **https** is the default value; **http** is a permitted value.
3. Backup server hostname.
4. Optional property that specifies the port on which the backup server is running. The default value is **8000**.
5. Path on the backup server where the backup snapshots are stored.
6. Secret created in step 1.
7. Optional property that specifies the Secret created in step 2.

4. Optional: To configure multiple backup servers, create a separate **CheBackupServerConfiguration** custom object for each backup server.

IMPORTANT

The CodeReady Workspaces Operator automatically backs up the CodeReady Workspaces instance before every CodeReady Workspaces update, permitting rollback to previous CodeReady Workspaces version if needed. If you configure only one backup server, that backup server is automatically used for pre-update backups by default. If you configure multiple backup servers, you must add the **che.eclipse.org/backup-before-update: true** annotation to the custom object of only one of them to specify it as the default backup server for pre-update backups. (If you don't add this annotation for one of multiple backup servers, or if you add this annotation for multiple backup servers, then the CodeReady Workspaces Operator defaults to using the internal backup server for pre-update backups.)

8.4.1.2. Backing up a CodeReady Workspaces instance to a REST backup server by using the CheClusterBackup custom object

You can use a **CheClusterBackup** custom object to make a backup snapshot of a CodeReady Workspaces instance and to send the snapshot to the configured backup server. To create each and every backup snapshot requires a new **CheClusterBackup** custom object; that is, editing an already consumed **CheClusterBackup** custom object, during or after backing up, has no effect.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Configured backup server. See Section 8.1, "Supported Restic-compatible backup servers" .
- Created **CheBackupServerConfiguration** custom object. See the previous section of this guide.

Procedure

1. Create the **CheClusterBackup** custom object, which creates a backup snapshot:

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: red-hat-codeready-workspaces-backup
  namespace: openshift-workspaces
spec:
  backupServerConfigRef: backup-server-configuration (1)
  useInternalBackupServer: false (2)
```

(1) Name of the **CheBackupServerConfiguration** custom object defining what backup server to use.

(2) Configures the Operator through this custom resource to use the CodeReady Workspaces-managed internal backup server or an administrator-managed external backup server (SFTP, Amazon S3 or S3 API compatible storage, or REST).

TIP

If you intend to reuse a **name** for **CheClusterBackup** custom objects, first delete any existing custom object with the same **name**. To delete it on the command line, use oc:

```
$ oc delete CheClusterBackup <name> -n openshift-workspaces
```

2. Read the **status** section of the **CheClusterBackup** custom object to verify the backup process, for example:

```
status:
  message: 'Backup is in progress. Start time: <timestamp>' (1)
  stage: Collecting CodeReady Workspaces installation data (2)
  state: InProgress (3)
```

1. Summary of the latest state of the backup process.
2. Current stage of the backup process in a human-readable format.
3. Backup process state: **InProgress** or **Succeeded** or **Failed**.

The CodeReady Workspaces instance is backed up in a snapshot when **state** is **Succeeded**:

```
status:
  ...
  message: >-
    Backup successfully finished at 2021-12-03 10:07:51.151090621 +0000 UTC
    m=+999.553272281 (1)
  snapshotId: 9f0adce2 (2)
  state: Succeeded (3)
```

1. Summary of the latest state of the backup process.
2. ID of the created backup snapshot. This field appears only when **state** is **Succeeded**.
3. Backup process state: **Succeeded** (or **Failed**).

8.4.2. Backing up a CodeReady Workspaces instance to a REST backup server by using crwctl

You can use crwctl to make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to a configured REST backup server. To do so, enter crwctl with the command-line options or set the environment variables.

8.4.2.1. Backing up a CodeReady Workspaces instance to a REST backup server by using crwctl with command-line options

To make a backup snapshot of a CodeReady Workspaces instance and send the snapshot to a configured REST backup server, run crwctl with the command-line options.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Installed **crwctl**.
- Configured REST backup server. See Section 8.1, “Supported Restic-compatible backup servers”.

Procedure

1. Enter the **crwctl server:backup** command with the following arguments:

```
$ crwctl server:backup \
--repository-url=<repository_url> \ 1
--repository-password=<repository_password> \ 2
--username=<username> \ 3
--password=<password> 4
```

1. Backup repository URL as an argument using the **--repository-url** or **-r** option. Syntax for the backup repository URL: **rest:http://*<host>*:8000/** or **rest:https://*<user>*:*<password>*@*<host>*:8000/** or **rest:https://*<user>*:*<password>*@*<host>*:8000/*<repository_path_on_rest_server>*/**. For more details about repository URL syntax, see Restic Documentation.

2. Backup repository password as an argument using the **--repository-password** or **-p** option.

3. When authentication is required, the username for authenticating to the backup REST server as the **--username=*<username>*** argument.

4. When authentication is required, the password for authenticating to the backup REST server as the **--password=*<password>*** argument.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--repository-url** and **--repository-password** options with the **crwctl server:backup** command just once and omit them onward when entering **crwctl server:backup** or **crwctl server:restore**.

2. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling backup...OK
 ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.4.2.2. Backing up a CodeReady Workspaces instance to a REST backup server by using crwctl and a CheBackupServerConfiguration custom object

You can back up a CodeReady Workspaces instance by entering the **name** of a **CheBackupServerConfiguration** custom object as an argument with the **crwctl server:backup** command.

Prerequisites

- Installed **crwctl**.
- Configured backup server. See Section 8.1, “Supported Restic-compatible backup servers” .

- Created **CheBackupServerConfiguration** custom object.

Procedure

1. Run the following command on a command line:

```
$ crwctl server:backup \
--backup-server-config-name=<name_of_CheBackupServerConfiguration> (1)
```

(1) This option points crwctl to a **CheBackupServerConfiguration** custom object. You can find the **name** of the **CheBackupServerConfiguration** custom object under **metadata** in the custom object.

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--backup-server-config-name** option with the **crwctl server:backup** command just once and omit this option onward when entering **crwctl server:backup**.

2. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling backup...OK
 ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.4.2.3. Configuring crwctl with environment variables to use a REST backup server

As an alternative to using command-line options for **crwctl server:backup**, you can configure crwctl for a REST backup server with environment variables. This enables you to run **crwctl server:backup** without having to enter those options with it.

Prerequisites

- Installed **crwctl**.
- Configured REST backup server. See Section 8.1, “Supported Restic-compatible backup servers”.

Procedure

Which environment variables you need to set depends on your use case:

- If you do not have a **CheBackupServerConfiguration** custom object yet, then you can set the following environment variables:
  - **BACKUP_REPOSITORY_URL** for the backup repository.
  - **BACKUP_REPOSITORY_PASSWORD** for the backup repository password.
  - When authentication to the REST backup server is required:

- **REST_SERVER_USERNAME** for the username.
- **REST_SERVER_PASSWORD** for the password.

- If you already have a **CheBackupServerConfiguration** custom object, you can set the following environment variable that will point crwctl to that **CheBackupServerConfiguration** custom object:
  - **BACKUP_SERVER_CONFIG_NAME** for the **name** of a **CheBackupServerConfiguration** custom object to point crwctl to. (You can find **name** under **metadata** in the custom object.)

8.5. BACKING UP OF CODEREADY WORKSPACES INSTANCES TO THE INTERNAL BACKUP SERVER

To create backups for testing purposes, you can use the internal backup server. The internal backup server is a REST backup server that is managed by the CodeReady Workspaces Operator.

IMPORTANT

The CodeReady Workspaces Operator defaults to using the internal backup server for pre-update backups (to allow the CodeReady Workspaces version rollback) if multiple backup servers are configured and if the **che.eclipse.org/backup-before-update: true** annotation is added to more than one of the multiple backup servers or is not added to the custom object of any one of the multiple backup servers.

8.5.1. Backing up a CodeReady Workspaces instance to the internal server by using the CheClusterBackup custom object

You can create a custom object to back up a CodeReady Workspaces instance to the internal backup server.

NOTE

In this case, the CodeReady Workspaces Operator automatically configures the internal backup server and creates a **CheBackupServerConfiguration** custom object with **name: backup-rest-server-configuration**.

WARNING

Configuring a backup server inside the same cluster, for example the internal backup server, is not the best practice because a cluster failure can be one of the scenarios when a backed up CodeReady Workspaces instance needs to be restored.

Procedure

1. Create the **CheClusterBackup** custom object to send a backup snapshot to the internal backup server:

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: red-hat-codeready-workspaces-backup
  namespace: openshift-workspaces
spec:
  useInternalBackupServer: true (1)
```

(1) Configures the custom resource to back up to the internal backup server.

2. Read the **status** section of the **CheClusterBackup** object to verify the backup process:

```
...
status:
  message: 'Backup is in progress. Start time: <timestamp>' (1)
  stage: Collecting CodeReady Workspaces installation data (2)
  state: InProgress (3)
...
```

(1) Displays the overall state or error message.

(2) Current phase of the backup process in a human-readable format.

(3) Backup process state: **InProgress** or **Succeeded** or **Failed**.

```
...
status:
  cheVersion: 7.37.0 (1)
  message: >- (2)
    Backup successfully finished at 2021-10-07 11:18:28.116103737 +0000 UTC
    m=+2384.430366711
  snapshotId: 40a09756 (3)
  state: Succeeded (4)
...
```

(1) The version of CodeReady Workspaces from which the backup snapshot was created. This field appears only when **state** is **Succeeded**.

(2) Displays the overall state or error message.

(3) The ID of the created backup snapshot. This field appears only when **state** is **Succeeded**.

(4) Backup process state: **InProgress** or **Succeeded** or **Failed**.

8.5.2. Backing up a CodeReady Workspaces instance to the internal server by using crwctl

You can use crwctl to take a backup snapshot of a CodeReady Workspaces instance and send it to the internal backup server.

NOTE

In this case, the CodeReady Workspaces Operator automatically configures the internal backup server and creates a **CheBackupServerConfiguration** custom object with **name: backup-rest-server-configuration**.

WARNING

This procedure does not back up the source code stored in users' workspaces. To back up the source code stored in users' workspaces, see Backups of persistent volumes.

Prerequisites

- Installed **crwctl**.
- CodeReady Workspaces has not been configured to use a backup server.

Procedure

1. On a command line, enter the **crwctl server:backup** command with no arguments.
2. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling backup...OK
 ✔ Waiting until backup process finishes...OK
Backup snapshot ID: 9f0adce2
Command server:backup has completed successfully in 00:10.
```

8.6. RESTORING A CODEREADY WORKSPACES INSTANCE FROM A BACKUP

You can restore a CodeReady Workspaces instance from a backup snapshot by using the **CheClusterRestore** custom object or **crwctl**:

- Section 8.6.1, “Restoring a CodeReady Workspaces instance from a backup by using the CheClusterRestore custom object”
- Section 8.6.2, “Restoring a CodeReady Workspaces instance from a backup by using crwctl”

8.6.1. Restoring a CodeReady Workspaces instance from a backup by using the CheClusterRestore custom object

You can restore a CodeReady Workspaces instance from a backup snapshot by using custom resources. A new **CheClusterRestore** custom object is required every time you restore a CodeReady Workspaces instance from a backup snapshot. (So editing an already consumed **CheClusterRestore** custom object, during or after backing up, has no effect.)

WARNING

Backup snapshots are bound to a OpenShift cluster. To restore a CodeReady Workspaces instance, you must only use a backup snapshot that was created in the same OpenShift cluster.

IMPORTANT

Do not use the **CheClusterRestore** custom object to recover a CodeReady Workspaces instance of an earlier version of CodeReady Workspaces! Use only **crwctl** to recover a CodeReady Workspaces instance of an earlier version of CodeReady Workspaces; see Section 8.6.2, "Restoring a CodeReady Workspaces instance from a backup by using crwctl"!

Prerequisites

- Backup snapshot of a CodeReady Workspaces instance on any of the following:
  - SFTP backup server
  - Amazon S3 or S3 API compatible backup server
  - REST backup server
  - Internal backup server

Procedure

1. Create the **CheClusterRestore** custom object, which restores a backup:

```
apiVersion: org.eclipse.che/v1
kind: CheClusterRestore
metadata:
  name: red-hat-codeready-workspaces-restore
  namespace: openshift-workspaces
spec:
  backupServerConfigRef: backup-server-configuration (1)
  snapshotId: <snapshot_id> (2)
```

(1) Name of the **CheBackupServerConfiguration** object that defines what backup server to use. You can find this name as **backupServerConfigRef** in **spec** of the **CheClusterBackup** custom object. (You can also find it as **name** in **metadata** of the **CheBackupServerConfiguration** custom object used to take the backup snapshot you are restoring.)

(2) Optional parameter defining the Snapshot ID to restore from. The default value is the last snapshot in the backup repository on the backup server.

TIP

To reuse a **name** for **CheClusterRestore** custom objects, first delete any existing custom object with the same **name**. To delete it on the command line, use oc:

```
$ oc delete CheClusterRestore <name> -n openshift-workspaces
```

2. Read the **status** section of the **CheClusterRestore** object to monitor the recovery process, for example:

```
status:
  message: 'Restore is in progress. Start time: <timestamp>' 1
  stage: Restoring CodeReady Workspaces related cluster objects 2
  state: InProgress 3
```

1. Overall state or error message; for example, **Restore successfully finished**.
2. Current phase of the recovery process in a human-readable format.
3. Recovery process state. One of **InProgress**, **Succeeded**, or **Failed**.

3. Check that the CodeReady Workspaces instance has been recovered.

TIP

If errors occur in your browser after the recovery, clean up the browser data for the CodeReady Workspaces domain.

4. Delete the **CheClusterRestore** object.

8.6.2. Restoring a CodeReady Workspaces instance from a backup by using crwctl

Using **crwctl**, you can restore a CodeReady Workspaces instance from a backup snapshot by using any of the following options (as applicable):

- Section 8.6.2.1, “Restoring a CodeReady Workspaces instance from its latest backup by using crwctl”
- Section 8.6.2.2, “Restoring a CodeReady Workspaces instance from a backup by snapshot ID”
- Section 8.6.2.3, “Restoring a CodeReady Workspaces instance by using crwctl and a CheClusterBackup custom object”

8.6.2.1. Restoring a CodeReady Workspaces instance from its latest backup by using crwctl

To restore the CodeReady Workspaces instance from its latest backup by using **crwctl**, you can enter **latest** as the snapshot ID.

WARNING

Backup snapshots are bound to a OpenShift cluster. To restore a CodeReady Workspaces instance, you must only use a backup snapshot that was created in the same OpenShift cluster.

IMPORTANT

If the backup snapshot was taken in an earlier version of Che, add the **--version=*<earlier_version_number>*** option to the **crwctl server:restore** command to roll back Che to that earlier version.

Prerequisites

- Installed **crwctl**.
- Backup snapshot of a CodeReady Workspaces instance on any of the following:
  - SFTP backup server
  - Amazon S3 or S3 API compatible backup server
  - REST backup server
  - Internal backup server

Procedure

1. Run the **crwctl server:restore** command with the following arguments:

```
$ crwctl server:restore \
--snapshot-id=latest \ 1
--repository-url=<repository-url> \ 2
--repository-password=<repository-password> 3
```

1. The latest snapshot.
2. The backup repository URL as an argument using the **--repository-url** or **-r** option. (Omit if using the internal backup server!)
3. The backup repository password as an argument using the **--repository-password** or **-p** option. (Omit if using the internal backup server!)

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--repository-url** and **--repository-password** options with the **crwctl server:backup** command just once and omit them onward when entering **crwctl server:backup** or **crwctl server:restore**.

2. When prompted in the output, enter **y** to confirm:

```
...
 ✔ Snapshot:          ...
 Asking for restore confirmation: Do you want to proceed? [y/n]
...
```

TIP

To skip this question, add the **--batch** option to the **server:restore** command.

3. Verify the output of the entered command. For example:

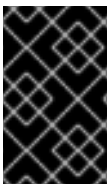
```
...
 ✔ Scheduling restore...OK
 ✔ Waiting until restore process finishes...OK
Command server:restore has completed successfully in 05:59.
```

8.6.2.2. Restoring a CodeReady Workspaces instance from a backup by snapshot ID

To restore a CodeReady Workspaces instance from a particular backup snapshot, you can enter the snapshot ID with **crwctl**.

WARNING

Backup snapshots are bound to a OpenShift cluster. To restore a CodeReady Workspaces instance, you must only use a backup snapshot that was created in the same OpenShift cluster.



IMPORTANT

If the backup snapshot was taken in an earlier version of Che, add the **--version=*<earlier_version_number>*** option to the **crwctl server:restore** command to roll back Che to that earlier version.

Prerequisites

- Installed **crwctl**.
- Backup snapshot of a CodeReady Workspaces instance on any of the following:
  - SFTP backup server
  - Amazon S3 or S3 API compatible backup server
  - REST backup server
  - Internal backup server

- Backup snapshot ID, which can be found in:
  - Command-line output when creating a backup
  - **status** of a **CheClusterBackup** custom object
  - Command-line output of the Restic command that can list snapshots in the backup repository

Procedure

1. Run the **crwctl server:restore** command with the backup snapshot ID specified using the **--snapshot-id=** option:

```
$ crwctl server:restore \
--snapshot-id=<snapshot-id> \ 1
--repository-url=<repository-url> \ 2
--repository-password=<repository-password> 3
```

1. The snapshot ID.
2. The backup repository URL as an argument using the **--repository-url** or **-r** option. (Omit if using the internal backup server!)
3. The backup repository password as an argument using the **--repository-password** or **-p** option. (Omit if using the internal backup server!)

TIP

Last used backup server information is stored in a Secret inside the CodeReady Workspaces cluster. When consistently using the same backup server, you can enter the **--repository-url** and **--repository-password** options with the **crwctl server:backup** command just once and omit them onward when entering **crwctl server:backup** or **crwctl server:restore**.

2. When prompted in the output, enter **y** to confirm:

```
...
 ✔ Snapshot:          ...
Asking for restore confirmation: Do you want to proceed? [y/n]
...
```

TIP

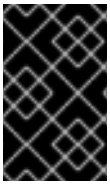
To skip this question, add the **--batch** option to the **server:restore** command.

3. Verify the output of the entered command. For example:

```
...
  ✔ Scheduling restore...OK
  ✔ Waiting until restore process finishes...OK
Command server:restore has completed successfully in 05:59.
```

8.6.2.3. Restoring a CodeReady Workspaces instance by using crwctl and a CheClusterBackup custom object

You can restore a CodeReady Workspaces instance by entering the name of the **CheClusterBackup** custom object as an argument with the **crwctl server:restore** command.



IMPORTANT

If the backup snapshot was taken in an earlier version of Che, then **crwctl server:restore --backup-cr-name=*<name_of_custom_object>*** rolls back Che to that earlier version, which is specified in the **CheClusterBackup** custom object.

Prerequisites

- Installed **crwctl**.
- The **CheClusterBackup** custom object that was used to create the backup snapshot.
- The backup snapshot that was taken using the **CheClusterBackup** custom object and stored on any of the following:
  - SFTP backup server
  - Amazon S3 or S3 API compatible backup server
  - REST backup server
  - Internal backup server

Procedure

1. Run the following command on a command line:

```
$ crwctl server:restore \
--backup-cr-name=<name_of_custom_object> 1
```

1 The name of the **CheClusterBackup** custom object.

2. When prompted in the output, enter **y** to confirm:

```
...
 Asking for restore confirmation: Do you want to proceed? [y/n]
...
```

TIP

To skip this question, add the **--batch** option to the **server:restore** command.

3. Verify the output of the entered command. For example:

```
...
 ✔ Scheduling restore...OK
 ✔ Waiting until restore process finishes...OK
```

```
Command server:restore has completed successfully in 05:59.
```

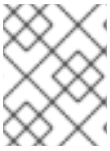
8.7. BACKUPS OF PERSISTENT VOLUMES

Persistent Volumes (PVs) store workspace data for CodeReady Workspaces similarly to how workspace data are stored for desktop IDEs on a local hard disk drive. PVs must be backed up periodically to prevent data loss. Storage-agnostic tools are available for backing up and restoring OpenShift resources, including PVs.

8.7.1. Velero

An open source tool for backing up OpenShift applications and their PVs is Velero. This tool can perform the following operations:

- Deploy in the cloud or on premises.
- Back up the cluster and restore the backed up data in case of data loss.
- Migrate cluster resources to other clusters.
- Replicate a production cluster to development and testing clusters.



NOTE

Alternatively, you can use backup solutions that depend on the underlying storage system: for example, solutions specific to Gluster or Ceph.

Additional resources

- Kubernetes Documentation - Persistent Volumes
- Velero Docs - How Velero Works
- Velero Docs - Disaster Recovery
- Velero Docs - Backup Reference
- Velero Docs - Restore Reference
- Gluster Documentation
- Ceph Documentation

8.8. BACKUPS OF POSTGRESQL

The CodeReady Workspaces server uses a PostgreSQL database for persisting data about the state of CodeReady Workspaces. The database contains information about user accounts, workspaces, preferences, and other details.

8.8.1. External PostgreSQL setup

By default, the CodeReady Workspaces Operator creates and manages deployment of the internal PostgreSQL database. However, the CodeReady Workspaces Operator does not support full lifecycle capabilities such as backups and recovery.

A business-critical setup must use an external PostgreSQL database that is configured:

- for High Availability (HA) and Point In Time Recovery (PITR)
- on-premises or using a cloud service such as Amazon Relational Database Service (Amazon RDS)

For example, Amazon RDS enables deployment of production databases in a Multi-Availability Zone configuration for a resilient disaster recovery strategy with daily and on-demand snapshots.

An example configuration is as follows:

Parameter	Value
Instance class	db.t2.small
vCPU	1
RAM	2 GB
Multi-az	true, 2 replicas
Engine version	9.6.11
TLS	enabled
Automated backups	enabled (30 days)

You can make workspace metadata and user information persistent by configuring the external PostgreSQL database and configuring CodeReady Workspaces to use the configured external PostgreSQL database:

- Section 8.8.2, “Configuring the external PostgreSQL”
- Section 8.8.3, “Configuring CodeReady Workspaces to work with the external PostgreSQL”

Additional resources

- PostgreSQL Documentation
- PostgreSQL Documentation - Backup and Restore
- Amazon RDS

8.8.2. Configuring the external PostgreSQL

To make workspace metadata and user information persistent, configure an external PostgreSQL database as follows:

Procedure

1. Define the values of the following placeholders:

<database-user>

CodeReady Workspaces server database user name

<database-password>

CodeReady Workspaces server database password

<database>

CodeReady Workspaces server database name

2. Use the following SQL script to create a user and database for the CodeReady Workspaces server to make workspace metadata persistent:

```
CREATE USER <database-user> WITH PASSWORD '<database-password>'
CREATE DATABASE <database>
GRANT ALL PRIVILEGES ON DATABASE <database> TO <database-user>
ALTER USER <database-user> WITH SUPERUSER
```

3. Define the value of the following placeholder:

<identity-database-password>

RH-SSO database password

4. Use the following SQL script to create a database for the RH-SSO back end to make user information persistent:

```
CREATE USER keycloak WITH PASSWORD '<identity-database-password>'
CREATE DATABASE keycloak
GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak
```

8.8.3. Configuring CodeReady Workspaces to work with the external PostgreSQL

To make workspace metadata and user information persistent, you must configure CodeReady Workspaces to work with the configured external PostgreSQL database.

Prerequisites

- Section 8.8.2, “Configuring the external PostgreSQL”
- Installed **oc**

Procedure

1. Create a project for CodeReady Workspaces:

```
$ oc create namespace openshift-workspaces
```

2. Create a Secret to store CodeReady Workspaces server database credentials:

```
$ oc create secret generic <server-database-credentials> \ 1
--from-literal=user=<database-user> \ 2
--from-literal=password=<database-password> \ 3
-n openshift-workspaces
```

1. Secret name to store CodeReady Workspaces server database credentials
2. CodeReady Workspaces server database username
3. CodeReady Workspaces server database password

3. Create a Secret to store RH-SSO database credentials:

```
$ oc create secret generic <identity-database-credentials> \ 1
--from-literal=password=<identity-database-password> \ 2
-n openshift-workspaces
```

1. Secret name to store RH-SSO database credentials
2. RH-SSO database password

4. Deploy Red Hat CodeReady Workspaces by executing the **crwctl** command and applying a patch. For example:

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

5. Ensure that **patch.yaml** contains the following lines to make the Operator skip the deployment of a database and pass connection details of an existing database to a CodeReady Workspaces server:

```
spec:
  database:
    externalDb: true
    chePostgresHostName: <hostname> 1
    chePostgresPort: <port> 2
    chePostgresSecret: <server-database-credentials> 3
    chePostgresDb: <database> 4
spec:
  auth:
    identityProviderPostgresSecret: <identity-database-credentials> 5
```

1. External database hostname
2. External database port
3. Secret name with CodeReady Workspaces server database credentials
4. CodeReady Workspaces server database name
5. Secret name with RH-SSO database credentials

CHAPTER 9. MIGRATION FROM POSTGRESQL 9 TO POSTGRESQL 13

By the 11th of November, 2021, the PostgreSQL version 9.6 came out of support, and CodeReady Workspaces team recommends that all users undergo migrating to version 13.

Follow the procedure below to migrate to a newer version of PostgreSQL successfully without any data loss.

Prerequisites

- The **oc** tool is available.
- An instance of CodeReady Workspaces running in OpenShift.

Procedure

1. Save and push changes back to the Git repositories for all running workspaces of the CodeReady Workspaces instance.
2. Stop all workspaces in the CodeReady Workspaces instance.
3. Scale down the CodeReady Workspaces and RH-SSO deployments:

```
oc scale deployment codeready --replicas=0 -n openshift-workspaces
oc scale deployment keycloak --replicas=0 -n openshift-workspaces
```

4. Backup available databases:

```
POSTGRES_POD=$(oc get pods -n openshift-workspaces | grep postgres | awk '{print $1}')
CHE_POSTGRES_DB=$(oc get checluster/codeready-workspaces -n openshift-workspaces
-o json  | jq '.spec.database.chePostgresDb')
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "pg_dump
$CHE_POSTGRES_DB > /tmp/che.sql"
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "pg_dump keycloak >
/tmp/keycloak.sql"
```

5. Copy the obtained backups to a local file system:

```
oc cp openshift-workspaces/$POSTGRES_POD:/tmp/che.sql che.sql
oc cp openshift-workspaces/$POSTGRES_POD:/tmp/keycloak.sql keycloak.sql
```

6. Scale down the PostgreSQL deployment:

```
oc scale deployment postgres --replicas=0 -n openshift-workspaces
```

7. Delete the corresponding PVC unit to clean up old data:

```
oc delete pvc postgres-data -n openshift-workspaces
```

After deleting the PVC from the step above, a new PVC will automatically appear in a few seconds.

8. Set the version of the new PostgreSQL database to 13.3:

```
oc patch checluster codeready-workspaces -n openshift-workspaces --type=json -p '[{"op":
"replace", "path": "/spec/database/postgresVersion", "value": "13.3"}]'
```

9. Scale up the PostgreSQL deployments:

```
oc scale deployment postgres --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=postgres -n openshift-
workspaces --timeout=120s
```

10. Provision a database:

```
POSTGRES_POD=$(oc get pods -n openshift-workspaces | grep postgres | awk '{print $1}')
OPERATOR_POD=$(oc get pods -n openshift-workspaces | grep codeready-operator | awk
'{print $1}')

IDENTITY_POSTGRES_SECRET=$(oc get checluster/codeready-workspaces -n openshift-
workspaces -o json | jq -r '.spec.auth.identityProviderPostgresSecret')
IDENTITY_POSTGRES_PASSWORD=$(if [ -z "$IDENTITY_POSTGRES_SECRET" ] || [
$IDENTITY_POSTGRES_SECRET = "null" ]; then oc get checluster/codeready-workspaces
-n openshift-workspaces -o json | jq -r '.spec.auth.identityProviderPostgresPassword'; else oc
get secret $IDENTITY_POSTGRES_SECRET -n openshift-workspaces -o json | jq -r
'.data.password' | base64 -d; fi)

oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"CREATE USER keycloak WITH PASSWORD '$IDENTITY_POSTGRES_PASSWORD'\""
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"CREATE DATABASE keycloak\""
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak\""

POSTGRES_SECRET=$(oc get checluster/codeready-workspaces -n openshift-workspaces
-o json | jq -r '.spec.database.chePostgresSecret')
CHE_USER=$(if [ -z "$POSTGRES_SECRET" ] || [ $POSTGRES_SECRET = "null" ]; then
oc get checluster/codeready-workspaces  -n openshift-workspaces -o json | jq -r
'.spec.database.chePostgresUser'; else oc get secret $POSTGRES_SECRET -n openshift-
workspaces -o json | jq -r '.data.user' | base64 -d; fi)

oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"ALTER USER $CHE_USER WITH SUPERUSER\""
```

11. Copy the backups to the PostgreSQL Pod:

```
oc cp che.sql openshift-workspaces/$POSTGRES_POD:/tmp/che.sql
oc cp keycloak.sql openshift-workspaces/$POSTGRES_POD:/tmp/keycloak.sql
```

12. Restore the database:

```
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql keycloak <
/tmp/keycloak.sql"
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql
$CHE_POSTGRES_DB < /tmp/che.sql"
```

13. Scale up the RH-SSO and CodeReady Workspaces deployments:

```
oc scale deployment keycloak --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=keycloak -n openshift-
workspaces --timeout=120s
oc scale deployment codeready --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=codeready -n openshift-
workspaces --timeout=120s
```

CHAPTER 10. READINESS INIT CONTAINERS

CodeReady Workspaces Operator installs CodeReady Workspaces and starts its containers in the correct order. If a node with CodeReady Workspaces is restarted and all CodeReady Workspaces containers start simultaneously, some containers may fail because some other component they depend on is not ready. To avoid such failures, the readiness init containers queue the containers for CodeReady Workspaces components to start in the correct order.

The readiness init containers are disabled by default. If you choose to enable them, proceed according to the installation method used to install CodeReady Workspaces:

- Section 10.1, “Enabling and disabling the readiness init containers for the Operator installer”
- Section 10.4, “Enabling and disabling the readiness init containers for the OLM installer”

10.1. ENABLING AND DISABLING THE READINESS INIT CONTAINERS FOR THE OPERATOR INSTALLER

You can enable and disable the readiness init containers for the CodeReady Workspaces installed by the Operator installer:

- Enabling the readiness init containers for the Operator installer
- Disabling the readiness init containers for the Operator installer

10.2. ENABLING THE READINESS INIT CONTAINERS FOR THE OPERATOR INSTALLER

The readiness init containers are not enabled by default, so to use them you first have to enable them. To enable the readiness init containers for the CodeReady Workspaces installed by the Operator installer:

Prerequisites

- Red Hat CodeReady Workspaces installed by the Operator installer.

Procedure

1. Find the name of the CodeReady Workspaces Operator Deployment. Usually it is **codeready-workspaces-operator**:

```
$ oc get deployments -n openshift-workspaces
```

2. Edit the Deployment as follows: Under **spec.template.spec.containers[0].env** of the Operator Deployment, insert the following lines:

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS (1)
  value: "true"
```

(1) **ADD_COMPONENT_READINESS_INIT_CONTAINERS** is an environment variable.

3. Wait while CodeReady Workspaces Operator restarts some components.

NOTE

Repeat these steps after each CodeReady Workspaces upgrade when a new Operator Deployment is created.

10.3. DISABLING THE READINESS INIT CONTAINERS FOR THE OPERATOR INSTALLER

To disable the previously enabled readiness init containers for the CodeReady Workspaces installed by the Operator installer:

Prerequisites

- Red Hat CodeReady Workspaces installed by the Operator installer.

Procedure

1. Find the name of the CodeReady Workspaces Operator Deployment. Usually it is **codeready-workspaces-operator**:

```
$ oc get deployments -n openshift-workspaces
```

2. Edit the Deployment as follows: Under **spec.template.spec.containers[0].env** of the Operator Deployment, remove the following lines:

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS (1)
  value: "true"
```

(1) **ADD_COMPONENT_READINESS_INIT_CONTAINERS** is an environment variable.

3. Wait while CodeReady Workspaces Operator restarts some components.

10.4. ENABLING AND DISABLING THE READINESS INIT CONTAINERS FOR THE OLM INSTALLER

You can enable and disable the readiness init containers for the CodeReady Workspaces installed by the OLM installer. (Available through **crwctl**, the OLM installer uses the Operator Lifecycle Manager to install CodeReady Workspaces.)

- Enabling the readiness init containers for the OLM installer
- Disabling the readiness init containers for the OLM installer

10.5. ENABLING THE READINESS INIT CONTAINERS FOR THE OLM INSTALLER

The readiness init containers are not enabled by default, so to use them you first have to enable them. To enable the readiness init containers for the CodeReady Workspaces installed by the OLM installer:

Prerequisites

- CodeReady Workspaces is installed by the OLM installer.

Procedure

1. Find the CodeReady Workspaces Operator subscription name:

```
$ oc get subscriptions -n openshift-workspaces
```

2. Get the CSV (Cluster Service Version) name from the CodeReady Workspaces Operator subscription:

```
$ oc get subscription <subscription-name> -n openshift-workspaces -o yaml | grep
installedCSV
```

3. Edit the **ClusterServiceVersion** YAML manifest:

```
$ oc edit csv <csv-name> -n openshift-workspaces
```

4. Add the following environment variable to the Operator Deployment spec:

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS
  value: "true"
```

5. Wait for the CodeReady Workspaces Operator restart to finish. The restarted Operator will then continue with restarting some of its components.

NOTE

Repeat these steps after each CodeReady Workspaces upgrade when a new CSV is created by OLM.

10.6. DISABLING THE READINESS INIT CONTAINERS FOR THE OLM INSTALLER

To disable the previously enabled readiness init containers for the CodeReady Workspaces installed by the OLM installer:

Prerequisites

- CodeReady Workspaces is installed by the OLM installer.

Procedure

1. Find the CodeReady Workspaces Operator subscription name:

```
$ oc get subscriptions -n openshift-workspaces
```

2. Get the CSV (Cluster Service Version) name from the CodeReady Workspaces Operator subscription:

```
$ oc get subscription <subscription-name> -n openshift-workspaces -o yaml | grep
installedCSV
```

3. Edit the **ClusterServiceVersion** YAML manifest:

```
$ oc edit csv <csv-name> -n openshift-workspaces
```

4. Remove the following environment variable from the Operator Deployment spec:

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS
  value: "true"
```

5. Wait for the CodeReady Workspaces Operator restart to finish. The restarted Operator will then continue with restarting some of its components.

CHAPTER 11. CACHING IMAGES FOR FASTER WORKSPACE START

To improve the start time performance of CodeReady Workspaces workspaces, use the Image Puller, a CodeReady Workspaces-agnostic component that can be used to pre-pull images for OpenShift clusters. The Image Puller is an additional OpenShift deployment which creates a *DaemonSet* that can be configured to pre-pull relevant CodeReady Workspaces workspace images on each node. These images would already be available when a CodeReady Workspaces workspace starts, therefore improving the workspace start time.

The Image Puller provides the following parameters for configuration.

Table 11.1. Image Puller parameters

Parameter	Usage	Default
CACHING_INTERVAL_HOURS	DaemonSets health checks interval in hours	**"1"**
CACHING_MEMORY_REQUEST	The memory request for each cached image when the puller is running. See Section 11.2, "Defining the memory parameters for the Image Puller".	**10Mi**
CACHING_MEMORY_LIMIT	The memory limit for each cached image when the puller is running. See Section 11.2, "Defining the memory parameters for the Image Puller".	**20Mi**
CACHING_CPU_REQUEST	The processor request for each cached image when the puller is running	**.05** or 50 millicores
CACHING_CPU_LIMIT	The processor limit for each cached image when the puller is running	**.2** or 200 millicores
DAEMONSET_NAME	Name of DaemonSet to create	**kubernetes-image-puller**
DEPLOYMENT_NAME	Name of the Deployment to create	**kubernetes-image-puller**
NAMESPACE	OpenShift project containing DaemonSet to create	**k8s-image-puller**

Parameter	Usage	Default
IMAGES	Semicolon separated list of images to pull, in the format ***<name1>=<image1>;<name2>=<image2>*** See Section 11.1, "Defining the list of images to pull".	
NODE_SELECTOR	Node selector to apply to the Pods created by the DaemonSet	**'{}'**
AFFINITY	Affinity applied to pods created by the DaemonSet	**'{}'**
IMAGE_PULL_SECRETS	List of image pull secrets, in the format **pullsecret1;...** to add to pods created by the DaemonSet. Those secrets need to be in the image puller's namespace and a cluster administrator must create them.	**""**

Additional resources

- Section 11.1, "Defining the list of images to pull"
- Section 11.2, "Defining the memory parameters for the Image Puller".
- Section 11.3, "Installing Image Puller using the CodeReady Workspaces Operator"
- Section 11.4, "Installing Image Puller on OpenShift 4 using OperatorHub"
- Section 11.5, "Installing Image Puller on OpenShift using OpenShift templates"
- Kubernetes Image Puller source code repository

11.1. DEFINING THE LIST OF IMAGES TO PULL

The Image Puller can pre-pull most images, including scratch images such as **che-machine-exec**. However, images that mount volumes in the Dockerfile, such as **traefik**, are not supported for pre-pulling on OpenShift 3.11.

Pre-pulling images involved in workspace startup will reduce workspace start times. For example:

- Che-Theia
- broker images
- plug-in sidecar images

Prerequisites

- The **curl** tool is available. See curl homepage.
- The **jq** tool is available. See jq homepage.
- The **yq** tool is available. See yq homepage.

Procedure

1. Gather a list of relevant container images for the OpenShift platform:

   Example 11.1. Getting the list of all images for CodeReady Workspaces 2.15

   ```
   $ curl -sSLo- https://raw.githubusercontent.com/redhat-developer/codeready-workspaces-images/crw-2.15-rhel-8/codeready-workspaces-operator-metadata-generated/manifests/codeready-workspaces.csv.yaml \
     | yq -r '.spec.relatedImages[]'
   ```

2. Retain the images involved on the workspace startup phase:

   - **eap**
   - **machineexec**
   - **mongodb**
   - **pluginbroker**
   - **plugin-**
   - **stacks**
   - **theia**
   - **ubi-minimal**

3. Exclude from the list the container images not supported by the target platform.

   - For AMD64 and Intel 64 (x86_64).

     Example 11.2. Image list for AMD64 and Intel 64 (x86_64)

     ```
     che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:6d13003539fcbda201065eae2e66dc67fed007ba3ba41fb3b8ec841650c52bc2;
     che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:de8ede01ce5d3b06ae8b1866bb482bb937f020f7dee5dfb20b041f02c1e63f68;
     codeready_workspaces_machineexec_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:dc0e082c9522158cb12345b1d184c3803d8a4a63a7189940e853e51557e43acf;
     codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-java11-
     ```

```
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3a
c537;
codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnik=registry.redhat.
o/codeready-workspaces/plugin-java11-
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3a
c537;
codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnik=registry.redhat.i
o/codeready-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c2
4c0cc;
codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnik=registry.redhat.ic
/codeready-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c2
4c0cc;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnik=registry.re
dhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:75fe8823dea867489b68169b764dc8b0b03290a456e9bfec5fe0cc413ee
c7355;
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnik=registry.redh
at.io/codeready-workspaces/plugin-openshift-
rhel8@sha256:d7603582f7ace76283641809b0c61dbcb78621735e536b789428e5a910
d35af3;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71c
fc13;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnik=registry.redhat.io/c
odeready-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71c
fc13;
codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnik=registry.redhat
io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf
5c8;
codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcnik=registry.redhat.
o/codeready-workspaces/stacks-dotnet-
rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf
5c8;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnik=registry.redhat
.io/codeready-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb3
9c1b3;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnik=registry.redhat
io/codeready-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb3
9c1b3;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2
ead3b;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2
ead3b;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnik=registry.redhat
.io/codeready-workspaces/theia-endpoint-
```

```
rhel8@sha256:5d26cf000924716d8d03969121a4c636e7fc8ef08aa21148eafa28a2c4ae
aff7;
codeready_workspaces_theia_plugin_registry_image_gixdcnik=registry.redhat.io/codere
ady-workspaces/theia-
rhel8@sha256:6000d00ef1029583642c01fec588f92addb95f16d56d0c23991a8f19314b
0f06;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____
_=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c8923
58632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdacq_=registr
y.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:bb3072afdbf31ddd1071fea37ed5308db3bf8a2478b5aa5aff8373e8042d
6aeb;
pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:2e4bbb2be6e7aff711ddc93f0b07e49c93d41e4c2ffc8ea75f804ad6fe2
5564e;
rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mo
ngodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;
```

- For IBM Z and IBM Power, exclude **dotnet**.

Example 11.3. Image list for IBM Z and IBM Power: excluding dotnet

```
che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:6d13003539fcbda201065eae2e66dc67fed007ba3ba41fb3b8ec841650c
52bc2;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:de8ede01ce5d3b06ae8b1866bb482bb937f020f7dee5dfb20b041f02c1e
63f68;
codeready_workspaces_machineexec_plugin_registry_image_gixdcnik=registry.redhat.i
o/codeready-workspaces/machineexec-
rhel8@sha256:dc0e082c9522158cb12345b1d184c3803d8a4a63a7189940e853e51557
e43acf;
codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnik=registry.redhat
io/codeready-workspaces/plugin-java11-
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3a
c537;
codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnik=registry.redhat.
o/codeready-workspaces/plugin-java11-
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3a
c537;
codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnik=registry.redhat.i
o/codeready-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c2
4c0cc;
codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnik=registry.redhat.io
/codeready-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c2
4c0cc;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnik=registry.re
```

```
dhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:75fe8823dea867489b68169b764dc8b0b03290a456e9bfec5fe0cc413ee
c7355;
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnik=registry.redh
at.io/codeready-workspaces/plugin-openshift-
rhel8@sha256:d7603582f7ace76283641809b0c61dbcb78621735e536b789428e5a910
d35af3;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71c
fc13;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnik=registry.redhat.io/c
odeready-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71c
fc13;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnik=registry.redhat
.io/codeready-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb3
9c1b3;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnik=registry.redhat
io/codeready-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb3
9c1b3;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2
ead3b;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcnik=registry.redhat.io/
codeready-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2
ead3b;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnik=registry.redhat
.io/codeready-workspaces/theia-endpoint-
rhel8@sha256:5d26cf000924716d8d03969121a4c636e7fc8ef08aa21148eafa28a2c4ae
aff7;
codeready_workspaces_theia_plugin_registry_image_gixdcnik=registry.redhat.io/codere
ady-workspaces/theia-
rhel8@sha256:6000d00ef1029583642c01fec588f92addb95f16d56d0c23991a8f19314b
0f06;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____
_=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c8923
58632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdacq_=registr
y.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:bb3072afdbf31ddd1071fea37ed5308db3bf8a2478b5aa5aff8373e8042d
6aeb;
pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:2e4bbb2be6e7aff711ddc93f0b07e49c93d41e4c2ffc8ea75f804ad6fe2
5564e;
rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mo
ngodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;
```

4. Determine images from the list for pre-pulling.
   For faster workspace startup times, consider pre-pulling the workspace-related images:

   - **theia-rhel8**
   - **theia-endpoint-rhel8**
   - **pluginbroker-artifacts-rhel8**
   - **pluginbroker-metadata-rhel8**
   - **stacks-*-rhel8**
   - **plugin-*-rhel8**
     - The list of stacks images: Container images - Stacks
     - The list of plug-in images: Container images - Plug-ins

Additional resources

- Section 11.2, “Defining the memory parameters for the Image Puller” .
- Section 11.4, “Installing Image Puller on OpenShift 4 using OperatorHub”
- Section 11.5, “Installing Image Puller on OpenShift using OpenShift templates”

11.2. DEFINING THE MEMORY PARAMETERS FOR THE IMAGE PULLER

Define the memory requests and limits parameters to ensure pulled containers and the platform have enough memory to run.

Prerequisites

- Section 11.1, “Defining the list of images to pull”

Procedure

1. To define the minimal value for **CACHING_MEMORY_REQUEST** or **CACHING_MEMORY_LIMIT**, consider the necessary amount of memory required to run each of the container images to pull.
2. To define the maximal value for **CACHING_MEMORY_REQUEST** or **CACHING_MEMORY_LIMIT**, consider the total memory allocated to the DaemonSet Pods in the cluster:

   ```
   (memory limit) * (number of images) * (number of nodes in the cluster)
   ```

   Pulling 5 images on 20 nodes, with a container memory limit of **20Mi** requires **2000Mi** of memory.

Additional resources

- Section 11.4, “Installing Image Puller on OpenShift 4 using OperatorHub”
- Section 11.5, “Installing Image Puller on OpenShift using OpenShift templates”

11.3. INSTALLING IMAGE PULLER USING THE CODEREADY WORKSPACES OPERATOR

This section describes how to use the CodeReady Workspaces Operator to install the Image Puller, which is a community-supported feature in the technology preview state.

Prerequisites

- Section 11.1, "Defining the list of images to pull"
- Section 11.2, "Defining the memory parameters for the Image Puller"
- Operator Lifecycle Manager and OperatorHub are available on the OpenShift instance. OpenShift provides them starting with version 4.2.
- The CodeReady Workspaces Operator is available. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#installing-che-on-openshift-4-using-operatorhub.adoc

Procedure

1. Enable Image Puller in the **CheCluster** Custom Resource by setting **.spec.imagePuller.enable** to **true**:

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: codeready-workspaces
spec:
  # ...
  imagePuller:
    enable: true
```

2. Configure Image Puller in the **CheCluster** Custom Resource:

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: codeready-workspaces
spec:
  ...
  imagePuller:
    enable: true
    spec:
      configMapName: <kubernetes-image-puller>
      daemonsetName: <kubernetes-image-puller>
      deploymentName: <kubernetes-image-puller>
      images: 'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:6d13003539fcbda201065eae2e66dc67fed007ba3ba41fb3b8ec841650c52bc2;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:de8ede01ce5d3b06ae8b1866bb482bb937f020f7dee5dfb20b041f02c1e63f68;codeready_workspaces_machineexec_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/machineexec-
```

```
rhel8@sha256:dc0e082c9522158cb12345b1d184c3803d8a4a63a7189940e853e51557e43acf;
codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnik=registry.redhat.io/code
ready-workspaces/plugin-java11-
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3ac537;co
deready_workspaces_plugin_java11_plugin_registry_image_gixdcnik=registry.redhat.io/coderea
dy-workspaces/plugin-java11-
rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3ac537;co
deready_workspaces_plugin_java8_devfile_registry_image_gixdcnik=registry.redhat.io/codereа
dy-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c24c0cc;c
odeready_workspaces_plugin_java8_plugin_registry_image_gixdcnik=registry.redhat.io/coderea
dy-workspaces/plugin-java8-
rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c24c0cc;c
odeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnik=registry.redhat.io/cc
deready-workspaces/plugin-kubernetes-
rhel8@sha256:75fe8823dea867489b68169b764dc8b0b03290a456e9bfec5fe0cc413eec7355;c
odeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnik=registry.redhat.io/cod
eready-workspaces/plugin-openshift-
rhel8@sha256:d7603582f7ace76283641809b0c61dbcb78621735e536b789428e5a910d35af3;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnik=registry.redhat.io/coderea
dy-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71cfc13;co
deready_workspaces_stacks_cpp_plugin_registry_image_gixdcnik=registry.redhat.io/codeready
-workspaces/stacks-cpp-
rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71cfc13;co
deready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnik=registry.redhat.io/codere
ady-workspaces/stacks-dotnet-
rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf5c8;cod
eready_workspaces_stacks_dotnet_plugin_registry_image_gixdcnik=registry.redhat.io/codereac
y-workspaces/stacks-dotnet-
rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf5c8;cod
eready_workspaces_stacks_golang_devfile_registry_image_gixdcnik=registry.redhat.io/coderea
dy-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb39c1b3;c
odeready_workspaces_stacks_golang_plugin_registry_image_gixdcnik=registry.redhat.io/coder
eady-workspaces/stacks-golang-
rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb39c1b3;c
odeready_workspaces_stacks_php_devfile_registry_image_gixdcnik=registry.redhat.io/coderea
dy-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2ead3b;c
odeready_workspaces_stacks_php_plugin_registry_image_gixdcnik=registry.redhat.io/codereac
y-workspaces/stacks-php-
rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2ead3b;c
odeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnik=registry.redhat.io/coder
eady-workspaces/theia-endpoint-
rhel8@sha256:5d26cf000924716d8d03969121a4c636e7fc8ef08aa21148eafa28a2c4aeaff7;co
deready_workspaces_theia_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-
workspaces/theia-
rhel8@sha256:6000d00ef1029583642c01fec588f92addb95f16d56d0c23991a8f19314b0f06;jbo
ss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____=registry
redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdacq_=registry.redha
.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:bb3072afdbf31ddd1071fea37ed5308db3bf8a2478b5aa5aff8373e8042d6aeb;pv
c_jobs=registry.redhat.io/ubi8/ubi-
```

```
minimal@sha256:2e4bbb2be6e7aff711ddc93f0b07e49c93d41e4c2ffc8ea75f804ad6fe25564e;r
hscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mongodb-
36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;'
```

NOTE

To use the supported Image Puller, install it separately from the KubernetesImagePuller Operator. Red Hat official build benefits from extra testing and validation provided by Red Hat.

Enabling the use of KubernetesImagePuller in Operator Hub during CodeReady Workspaces installation, sets the Community supported version for use.

- Community build
- Red Hat official build

Default images

- The CodeReady Workspaces Operator populates the **.spec.imagePuller.spec.images** field with default images used for workspace startup (Theia images, plug-in broker images, sidecar plug-in images), provided that no images were added to this field before creating the **CheCluster** Custom Resource. The CodeReady Workspaces Operator updates the default images in the **.spec.imagePuller.spec.images** field after every rollout update of CodeReady Workspaces. However, if images were added to the **.spec.imagePuller.spec.images** field before creating the **CheCluster** Custom Resource, the CodeReady Workspaces Operator will not add default images.
- If user-provided images are added to the **.spec.imagePuller.spec.images** field **after** creating the **CheCluster** Custom Resource, the CodeReady Workspaces Operator will still update default images on subsequent CodeReady Workspaces rollout updates. Non-default images remain unchanged in the **.spec.imagePuller.spec.images** field after rollout updates.

Verification

- OpenShift creates a **kubernetes-image-puller-operator** Subscription.
- The **eclipse-che namespace** contains a **community supported Kubernetes Image Puller Operator ClusterServiceVersion**:

```
$ oc get clusterserviceversions
```

- The **eclipse-che namespace** contains these deployments: **kubernetes-image-puller** and **kubernetes-image-puller-operator**.

```
$ oc get deployments
```

- The community supported Kubernetes Image Puller Operator creates a **KubernetesImagePuller** Custom Resource:

```
$ oc get kubernetesimagepullers
```

Uninstalling Image Puller using CodeReady Workspaces Operator

1. Edit the **CheCluster** Custom Resource and set **.spec.imagePuller.enable** to **false**.
2. Edit the **CheCluster** Custom Resource and set the **.spec.imagePuller.spec** to configure the optional Image Puller parameters for the CodeReady Workspaces Operator.

11.4. INSTALLING IMAGE PULLER ON OPENSHIFT 4 USING OPERATORHUB

This procedure describes how to install the community supported Kubernetes Image Puller Operator on OpenShift 4 using the Operator.

Prerequisites

- An administrator account on a running instance of OpenShift 4.
- Section 11.1, "Defining the list of images to pull"
- Section 11.2, "Defining the memory parameters for the Image Puller".

Procedure

1. To create an OpenShift project *<kubernetes-image-puller>* to host the Image Puller, open the OpenShift web console, navigate to the **Home → Projects** section and click **Create Project**.
2. Specify the project details:
   - **Name**: *<kubernetes-image-puller>*
   - **Display Name**: *<Image Puller>*
   - **Description**: *<Kubernetes Image Puller>*
3. Navigate to **Operators → OperatorHub**.
4. Use the **Filter by keyword** box to search for **community supported Kubernetes Image Puller Operator**. Click the **community supported Kubernetes Image Puller Operator**.
5. Read the description of the Operator. Click **Continue → Install**.
6. Select **A specific project on the cluster** for the **Installation Mode**. In the drop-down find the OpenShift project *<kubernetes-image-puller>*. Click **Subscribe**.
7. Wait for the community supported Kubernetes Image Puller Operator to install. Click the **KubernetesImagePuller → Create instance**.
8. In a redirected window with a YAML editor, make modifications to the **KubernetesImagePuller** Custom Resource and click **Create**.
9. Navigate to the **Workloads** and **Pods** menu in the *<kubernetes-image-puller>* OpenShift project. Verify that the Image Puller is available.

11.5. INSTALLING IMAGE PULLER ON OPENSHIFT USING OPENSHIFT TEMPLATES

This procedure describes how to install the Kubernetes Image Puller on OpenShift using OpenShift templates.

Prerequisites

- A running OpenShift cluster.
- The **oc** tool is available.
- Section 11.1, "Defining the list of images to pull" .
- Section 11.2, "Defining the memory parameters for the Image Puller" .

Procedure

1. Clone the Image Puller repository and get in the directory containing the OpenShift templates:

```
$ git clone https://github.com/che-incubator/kubernetes-image-puller
$ cd kubernetes-image-puller/deploy/openshift
```

2. Configure the **app.yaml**, **configmap.yaml** and **serviceaccount.yaml** OpenShift templates using following parameters:

Table 11.2. Image Puller OpenShift templates parameters in app.yaml

Value	Usage	Default
DEPLOYMENT_NAME	The value of **DEPLOYMENT_NAME** in the ConfigMap	**kubernetes-image-puller**
IMAGE	Image used for the **kubernetes-image-puller** deployment	**registry.redhat.io/codeready-workspaces/imagepuller-rhel8:2.15**
IMAGE_TAG	The image tag to pull	**latest**
SERVICEACCOUNT_NAME	The name of the ServiceAccount created and used by the deployment	**kubernetes-image-puller**

Table 11.3. Image Puller OpenShift templates parameters in configmap.yaml

Value	Usage	Default
CACHING_CPU_LIMIT	The value of **CACHING_CPU_LIMIT** in the ConfigMap	**.2**

Value	Usage	Default
CACHING_CPU_REQUEST	The value of **CACHING_CPU_REQUEST** in the ConfigMap	**.05**
CACHING_INTERVAL_HOURS	The value of **CACHING_INTERVAL_HOURS** in the ConfigMap	**"1"**
CACHING_MEMORY_LIMIT	The value of **CACHING_MEMORY_LIMIT** in the ConfigMap	**"20Mi"**
CACHING_MEMORY_REQUEST	The value of **CACHING_MEMORY_REQUEST** in the ConfigMap	**"10Mi"**
DAEMONSET_NAME	The value of **DAEMONSET_NAME** in the ConfigMap	**kubernetes-image-puller**
DEPLOYMENT_NAME	The value of **DEPLOYMENT_NAME** in the ConfigMap	**kubernetes-image-puller**
IMAGES	The value of **IMAGES** in the ConfigMap	**'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:6d13003539fcbda201065eae2e66dc67fed007ba3ba41fb3b8ec841650c52bc2;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:de8ede01ce5d3b06ae8b1866bb482bb937f020f7dee5dfb20b041f02c1e63f68;codeready_workspaces_machineexec_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:dc0e082c9522158cb12345b1d184c3803d8a4a63a7189940e853e51557e43acf;codeready_wor**

Value	Usage	Default
		kspaces_plugin_java11_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3ac537;codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:315273182e1f4dc884365fc3330ada3937b40369f3faf7762847ec433c3ac537;codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c24c0cc;codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:8cb1e495825051b83cf903bb317e55823a6f57b3bad92e9407dc8fa59c24c0cc;codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-kubernetes-rhel8@sha256:75fe8823dea867489b68169b764dc8b0b03290a456e9bfec5fe0cc413eec7355;codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:d7603582f7ace76283641809b0c61dbcb78621735e536b789428e5

Value	Usage	Default
		a910d35af3;codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71cfc13;codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:c2f38140f52112b2a7688c2a179afcaa930ad6216925eb322cfd9634a71cfc13;codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf5c8;codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:f48fe1caa5be1ae91140681bee159ca8b11dc687fa50fbf9dc5644f4852bf5c8;codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:db76d04752973223e2c0de9401ebf06b84263e1bb6d29f1455daaff0cb39c1b3;codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:db76d0475

Value	Usage	Default
		2973223e2c0de9401ebf06b3423c31bb6d29f1455daaff0cb39c1b3;codeready_workspaces_stacks_php_devfile_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2ead3b;codeready_workspaces_stacks_php_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:d120c41ee8dd80fb960dd4c1657bede536d32f13f3c3ca84e986a830ec2ead3b;codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:5d26cf000924716d8d03969121a4c636e7fc8ef08aa21148eafa28a2c4aeaff7;codeready_workspaces_theia_plugin_registry_image_gixdcnik=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:6000d00ef1029583642c01fec588f92addb95f16d56d0c23991a8f19314b0f06;jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdacq_=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-rhel8@sha256:bb3072afdbf31ddd1071fea37ed5308db

Value	Usage	Default
		3bf8a2478b5aa5aff8373e8 3f2d3daeb;pvc_jobs=regist ry.redhat.io/ubi8/ubi-minimal@sha256:2e4bbb2 be6e7aff711ddc93f0b07e4 9c93d41e4c2ffc8ea75f804a d6fe25564e;rhscl_mongod b_36_rhel7_devfile_registr y_image_gewtkmak=regis try.redhat.io/rhscl/mongod b-36-rhel7@sha256:9f799d356d 7d2e442bde9d401b720600 fd9059a3d8eefea6f3b2ffa7 21c0dc73;'
NAMESPACE	The value of **NAMESPACE** in the ConfigMap	**k8s-image-puller**
NODE_SELECTOR	The value of **NODE_SELECTOR** in the ConfigMap	**"{}"**

Table 11.4. Image Puller OpenShift templates parameters in serviceaccount.yaml

Value	Usage	Default
SERVICEACCOUNT_NAME	The name of the ServiceAccount created and used by the deployment	**kubernetes-image-puller**

3. Create an OpenShift project to host the Image Puller:

```
$ oc new-project <k8s-image-puller>
```

4. Process and apply the templates to install the puller:

```
$ oc process -f serviceaccount.yaml | oc apply -f -
$ oc process -f configmap.yaml | oc apply -f -
$ oc process -f app.yaml | oc apply -f -
```

Verification steps

1. Verify the existence of a *<kubernetes-image-puller>* deployment and a *<kubernetes-image-puller>* DaemonSet. The DaemonSet needs to have a Pod for each node in the cluster:

```
$ oc get deployment,daemonset,pod --namespace <k8s-image-puller>
```

2. Verify the values of the *<kubernetes-image-puller>* **ConfigMap**.

```
$ oc get configmap <kubernetes-image-puller> --output yaml
```

CHAPTER 12. MANAGING IDENTITIES AND AUTHORIZATIONS

This section describes different aspects of managing identities and authorizations of Red Hat CodeReady Workspaces.

12.1. AUTHENTICATING USERS

This document covers all aspects of user authentication in Red Hat CodeReady Workspaces, both on the CodeReady Workspaces server and in workspaces. This includes securing all REST API endpoints, WebSocket or JSON RPC connections, and some web resources.

All authentication types use the JWT open standard as a container for transferring user identity information. In addition, CodeReady Workspaces server authentication is based on the OpenID Connect protocol implementation, which is provided by default by RH-SSO.

Authentication in workspaces implies the issuance of self-signed per-workspace JWT tokens and their verification on a dedicated service based on JWTProxy.

12.1.1. Authenticating to the CodeReady Workspaces server

12.1.1.1. Authenticating to the CodeReady Workspaces server using other authentication implementations

This procedure describes how to use an OpenID Connect (OIDC) authentication implementation other than RH-SSO.

Procedure

1. Update the authentication configuration parameters that are stored in the **multiuser.properties** file (such as client ID, authentication URL, realm name).
2. Write a single filter or a chain of filters to validate tokens, create the user in the CodeReady Workspaces dashboard, and compose the **subject** object.
3. If the new authorization provider supports the OpenID protocol, use the OIDC JS client library available at the settings endpoint because it is decoupled from specific implementations.
4. If the selected provider stores additional data about the user (first and last name, job title), it is recommended to write a provider-specific **ProfileDao** implementation that provides this information.

12.1.1.2. Authenticating to the CodeReady Workspaces server using OAuth

For easy user interaction with third-party services, the CodeReady Workspaces server supports OAuth authentication. OAuth tokens are also used for GitHub-related plug-ins.

OAuth authentication has two main flows:

delegated

Default. Delegates OAuth authentication to RH-SSO server.

embedded

Uses built-in CodeReady Workspaces server mechanism to communicate with OAuth providers.

To switch between the two implementations, use the **che.oauth.service_mode=*<embedded|delegated>*** configuration property.

The main REST endpoint in the OAuth API is /**api**/**oauth**, which contains:

- An authentication method, /**authenticate**, that the OAuth authentication flow can start with.
- A callback method, /**callback**, to process callbacks from the provider.
- A token GET method, /**token**, to retrieve the current user's OAuth token.
- A token DELETE method, /**token**, to invalidated the current user's OAuth token.
- A GET method, /, to get the list of configured identity providers.

12.1.1.3. Using Swagger or REST clients to execute queries

The user's RH-SSO token is used to execute queries to the secured API on the user's behalf through REST clients. A valid token must be attached as the **Request** header or the **?token=$token** query parameter.

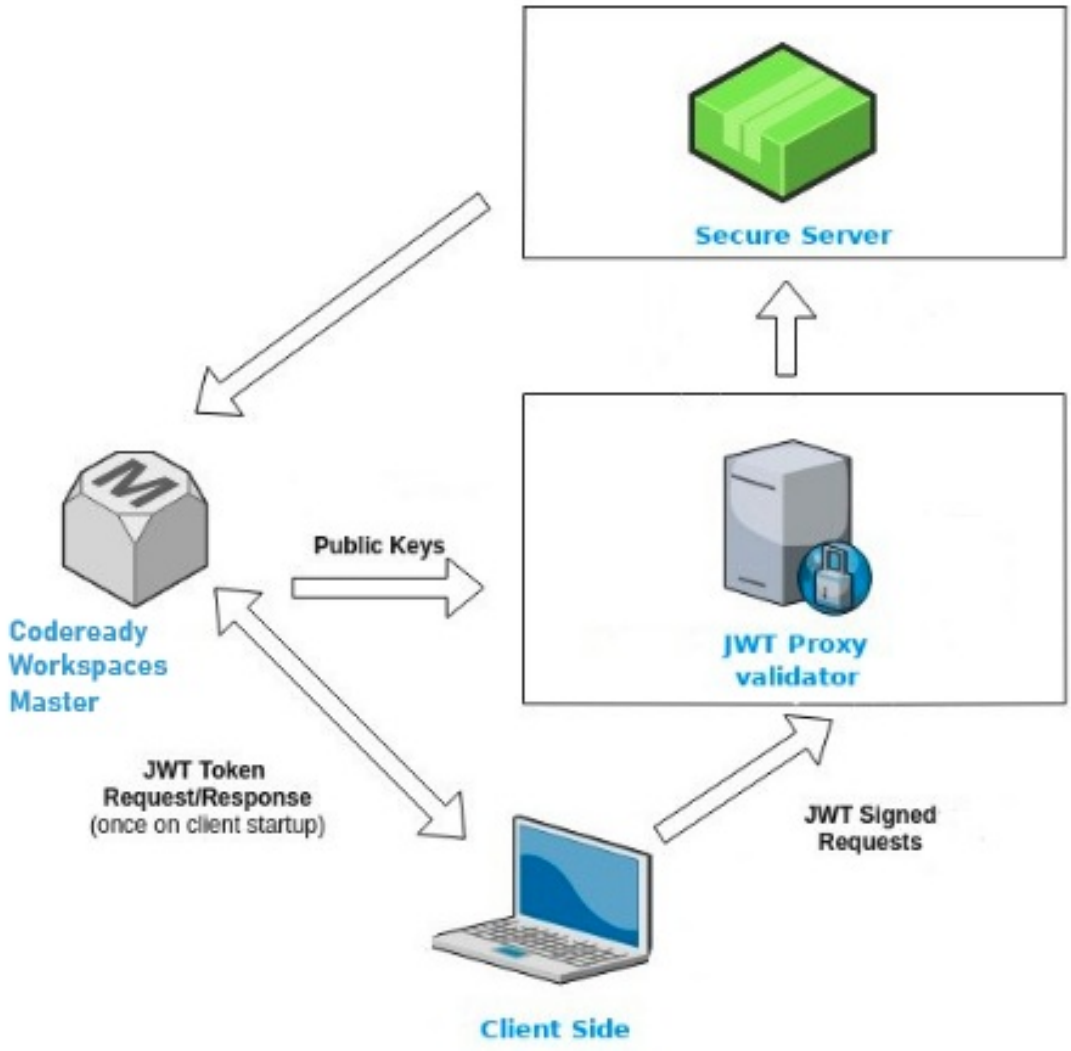
Access the CodeReady Workspaces Swagger interface at **https://codeready-<openshift_deployment_name>.<domain_name>/swagger**. The user must be signed in through RH-SSO, so that the access token is included in the **Request** header.

12.1.2. Authenticating in a CodeReady Workspaces workspace

Workspace containers may contain services that must be protected with authentication. Such protected services are called **secure**. To secure these services, use a machine authentication mechanism.

JWT tokens avoid the need to pass RH-SSO tokens to workspace containers (which can be insecure). Also, RH-SSO tokens may have a relatively shorter lifetime and require periodic renewals or refreshes, which is difficult to manage and keep in sync with the same user session tokens on clients.

Figure 12.1. Authentication inside a workspace



12.1.2.1. Creating secure servers

To create secure servers in CodeReady Workspaces workspaces, set the **secure** attribute of the endpoint to **true** in the **dockerimage** type component in the devfile.

Devfile snippet for a secure server

```
components:
  - type: dockerimage
    endpoints:
      - attributes:
          secure: 'true'
```

12.1.2.2. Workspace JWT token

Workspace tokens are JSON web tokens (JWT) that contain the following information in their claims:

- **uid**: The ID of the user who owns this token
- **uname**: The name of the user who owns this token
- **wsid**: The ID of a workspace which can be queried with this token

Every user is provided with a unique personal token for each workspace. The structure of a token and the signature are different from what they are in RH-SSO. The following is an example token view:

```
# Header
{
  "alg": "RS512",
  "kind": "machine_token"
}
# Payload
{
  "wsid": "workspacekrh99xjenek3h571",
  "uid": "b07e3a58-ed50-4a6e-be17-fcf49ff8b242",
  "uname": "john",
  "jti": "06c73349-2242-45f8-a94c-722e081bb6fd"
}
# Signature
{
  "value": "RSASHA256(base64UrlEncode(header) + . +  base64UrlEncode(payload))"
}
```

The SHA-256 cipher with the RSA algorithm is used for signing JWT tokens. It is not configurable. Also, there is no public service that distributes the public part of the key pair with which the token is signed.

12.1.2.3. Machine token validation

The validation of machine tokens (JWT tokens) is performed using a dedicated per-workspace service with **JWTProxy** running on it in a separate Pod. When the workspace starts, this service receives the public part of the SHA key from the CodeReady Workspaces server. A separate verification endpoint is created for each secure server. When traffic comes to that endpoint, **JWTProxy** tries to extract the token from the cookies or headers and validates it using the public-key part.

To query the CodeReady Workspaces server, a workspace server can use the machine token provided in the **CHE_MACHINE_TOKEN** environment variable. This token is the user's who starts the workspace. The scope of such requests is restricted to the current workspace only. The list of allowed operations is also strictly limited.

12.2. AUTHORIZING USERS

User authorization in CodeReady Workspaces is based on the permissions model. Permissions are used to control the allowed actions of users and establish a security model. Every request is verified for the presence of the required permission in the current user subject after it passes authentication. You can control resources managed by CodeReady Workspaces and allow certain actions by assigning permissions to users.

Permissions can be applied to the following entities:

- Workspace
- System

All permissions can be managed using the provided REST API. The APIs are documented using Swagger at **https://codeready-<openshift_deployment_name>.<domain_name>/swagger/#!/permissions**.

12.2.1. CodeReady Workspaces workspace permissions

The user who creates a workspace is the workspace owner. By default, the workspace owner has the following permissions: **read**, **use**, **run**, **configure**, **setPermissions**, and **delete**. Workspace owners can invite users into the workspace and control workspace permissions for other users.

The following permissions are associated with workspaces:

Table 12.1. CodeReady Workspaces workspace permissions

Permission	Description
read	Allows reading the workspace configuration.
use	Allows using a workspace and interacting with it.
run	Allows starting and stopping a workspace.
configure	Allows defining and changing the workspace configuration.
setPermissions	Allows updating the workspace permissions for other users.
delete	Allows deleting the workspace.

12.2.2. CodeReady Workspaces system permissions

CodeReady Workspaces system permissions control aspects of the whole CodeReady Workspaces installation. The following permissions are applicable to the system:

Table 12.2. CodeReady Workspaces system permission

Permission	Description
manageSystem	Allows control of the system and workspaces.
setPermissions	Allows updating the permissions for users on the system.
manageUsers	Allows creating and managing users.
monitorSystem	Allows accessing endpoints used for monitoring the state of the server.

All system permissions are granted to the administrative user. To configure the administrative user, use the **CHE_SYSTEM_ADMIN__NAME** property. The default value is **admin**. The system permissions are granted when the CodeReady Workspaces server starts. If the record of the user is not in the CodeReady Workspaces user database, the permissions are granted after the first login of the user.

12.2.3. manageSystem permission

Users with the **manageSystem** permission have access to the following services:

Path	HTTP Method	Description
/resource/free/	GET	Get free resource limits.
/resource/free/{accountId}	GET	Get free resource limits for the given account.
/resource/free/{accountId}	POST	Edit free resource limit for the given account.
/resource/free/{accountId}	DELETE	Remove free resource limit for the given account.
/installer/	POST	Add installer to the registry.
/installer/{key}	PUT	Update installer in the registry.
/installer/{key}	DELETE	Remove installer from the registry.
/logger/	GET	Get logging configurations in the CodeReady Workspaces server.
/logger/{name}	GET	Get configurations of logger by its name in the CodeReady Workspaces server.
/logger/{name}	PUT	Create logger in the CodeReady Workspaces server.
/logger/{name}	POST	Edit logger in the CodeReady Workspaces server.
/resource/{accountId}/details	GET	Get detailed information about resources for the given account.
/system/stop	POST	Shutdown all system services, prepare CodeReady Workspaces to stop.

12.2.4. monitorSystem permission

Users with the **monitorSystem** permission have access to the following services.

Path	HTTP Method	Description
/activity	GET	Get workspaces in a certain state for a certain amount of time.

12.2.5. Listing CodeReady Workspaces permissions

To list CodeReady Workspaces permissions that apply to a specific **resource**, perform the **GET /permissions** request.

To list the permissions that apply to a **user**, perform the **GET /permissions/{domain}** request.

To list the permissions that apply to **all users**, perform the **GET /permissions/{domain}/all** request. The user must have **manageSystem** permissions to see this information.

The suitable domain values are:

- system
- organization
- workspace

NOTE

The domain is optional. If no domain is specified, the API returns all possible permissions for all the domains.

12.2.6. Assigning CodeReady Workspaces permissions

To assign permissions to a resource, perform the **POST /permissions** request. The suitable domain values are:

- system
- organization
- workspace

The following is a message body that requests permissions for a user with a **userId** to a workspace with a **workspaceID**:

Requesting CodeReady Workspaces user permissions

```
{
  "actions": [
    "read",
    "use",
    "run",
    "configure",
    "setPermissions"
  ],
```

```
  "userId": "userID",          1
  "domainId": "workspace",
  "instanceId": "workspaceID"  2
}
```

1 The **userId** parameter is the ID of the user that has been granted certain permissions.

2 The **instanceId** parameter is the ID of the resource that retrieves the permission for all users.

12.3. CONFIGURING AUTHORIZATION

CodeReady Workspaces uses the permissions model for user authorization.

12.3.1. Authorization and user management

Red Hat CodeReady Workspaces uses RH-SSO to create, import, manage, delete, and authenticate users. RH-SSO uses built-in authentication mechanisms and user storage. It can use third-party identity management systems to create and authenticate users. Red Hat CodeReady Workspaces requires a RH-SSO token when you request access to CodeReady Workspaces resources.

Local users and imported federation users must have an email address in their profile.

The default RH-SSO credentials are **admin:admin**. You can use the **admin:admin** credentials when logging into Red Hat CodeReady Workspaces for the first time. It has system privileges.

Identifying the RH-SSO URL

Go to the OpenShift web console and to the **RH-SSO** project.

12.3.2. Configuring CodeReady Workspaces to work with RH-SSO

The deployment script configures RH-SSO. It creates a **codeready-public** client with the following fields:

- **Valid Redirect URIs**: Use this URL to access CodeReady Workspaces.
- **Web Origins**

The following are common errors when configuring CodeReady Workspaces to work with RH-SSO:

Invalid redirectURI error

Occurs when you access CodeReady Workspaces at **myhost**, which is an alias, and your original **CHE_HOST** is **1.1.1.1**. If this error occurs, go to the RH-SSO administration console and ensure that the valid redirect URIs are configured.

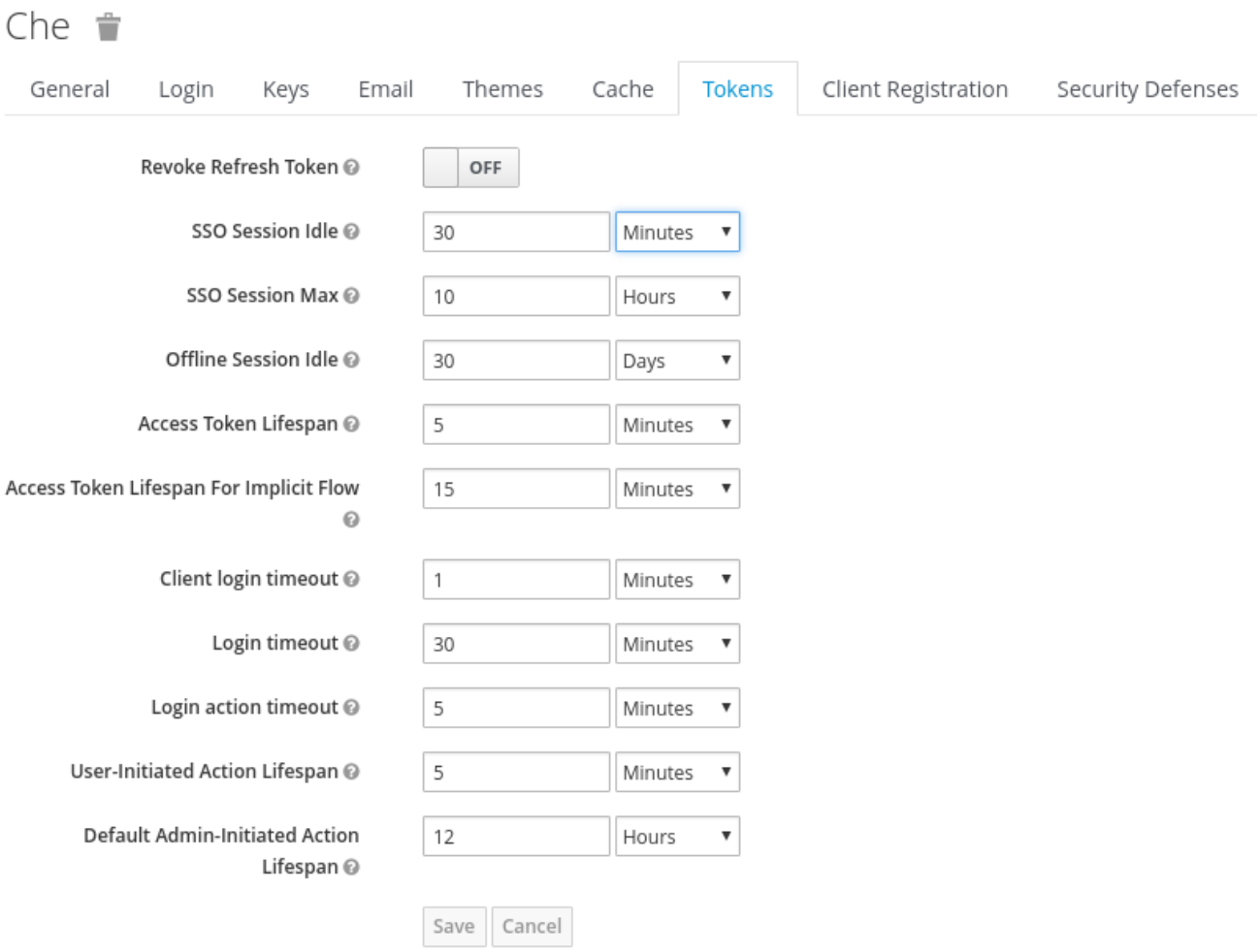
CORS error

Occurs when you have an invalid web origin.

12.3.3. Configuring RH-SSO tokens

A user token expires after 30 minutes by default.

You can change the following RH-SSO token settings:



12.3.4. Setting up user federation

RH-SSO federates external user databases and supports LDAP and Active Directory. You can test the connection and authenticate users before choosing a storage provider.

See the User storage federation page in RH-SSO documentation to learn how to add a provider.

See the LDAP and Active Directory page in RH-SSO documentation to specify multiple LDAP servers.

12.3.5. Enabling authentication with social accounts and brokering

RH-SSO provides built-in support for GitHub, OpenShift, and most common social networks such as Facebook and Twitter. See RH-SSO documentation to learn how to enable Login with GitHub.

12.3.5.1. Configuring GitHub OAuth

OAuth for GitHub allows for automatic SSH key upload to GitHub.

Prerequisites

- The **oc** tool is available.

Procedure

1. Create an OAuth application in GitHub with the following URLs:

- Enter the CodeReady Workspaces URL as **Homepage URL**. The default value is **https://codeready-openshift-workspaces.*<domain>***/.
- Enter the **Authorization callback URL** as follows:
  - If the Dev Workspace engine is not enabled, enter the RH-SSO GitHub endpoint URL as **Authorization callback URL**. The default value is **https://keycloak-openshift-workspaces.*<domain>*/auth/realms/codeready/broker/github/endpoint**.
  - If the Dev Workspace engine is enabled, enter the CodeReady Workspaces OAuth callback URL as **Authorization callback URL**. The default value is **https://codeready-openshift-workspaces.*<domain>*/api/oauth/callback**.

2. Create a new secret in the project where CodeReady Workspaces is deployed.

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: <...> 1
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: github
type: Opaque
data:
  id: <...> 2
  secret: <...> 3
EOF
```

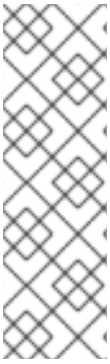
1. CodeReady Workspaces namespace. The default is openshift-workspaces
2. base64 encoded GitHub OAuth Client ID
3. base64 encoded GitHub OAuth Client Secret

3. If CodeReady Workspaces was already installed wait until rollout of RH-SSO component finishes.

12.3.5.2. Configuring a Bitbucket server that uses self-signed TLS certificates

The following chapter describes how to configure a Bitbucket (BB) server that uses self-signed TLS certificates so that the CodeReady Workspaces server and workspace components can establish a trusted connection with BB.

- Creating ConfigMaps for additional TLS and **gitSelfSign** certificates. This enables:
  - Launching a factory using a devfile URL.
  - Importing and cloning a project.



NOTE

- Configure the OAuth 1 authentication on the BB server side. For more information, see Configuring Bitbucket Server OAuth 1
- Creating a ConfigMap for importing additional certificates is necessary only if a BB server is setup with self-signed TLS certificates. These certificates are needed for the proper functionality of CodeReady Workspaces server and tools inside of a workspace, which use them for performing Git operations related to a specific repository.

Prerequisites

- A value of the BB server certification authority (CA) exported in the Base64 ASCII format and stored in a **ca.crt** file.
- An instance of CodeReady Workspaces.

Procedure

1. Provision the CA of the BB server to the CodeReady Workspaces server to enable it to read the devfiles stored in the BB server. To do so, add the following ConfigMap to the **openshift-workspaces** project:

```
$ oc create configmap bitbucket-ca-cert-for-factory --from-file=ca.crt -n openshift-workspaces
```

```
$ oc label configmap bitbucket-ca-cert-for-factory app.kubernetes.io/part-of=che.eclipse.org
app.kubernetes.io/component=ca-bundle -n openshift-workspaces
```

2. Provision the CA of the BB server to the CodeReady Workspaces server to be able to use Git operations. To do so, add a new ConfigMap to the **openshift-workspaces** project:

```
$ oc create configmap che-git-self-signed-cert --from-file=ca.crt --from-
literal=githost=<bitbucket_server_url> -n openshift-workspaces
```

3. Edit the CheCluster Custom Resource (CR) to configure the CodeReady Workspaces server.

```
spec:
  server:
    # ...
    gitSelfSignedCert: <boolean> 1
```

1 Use **true** for a BB server that use a self-signed cert. Default value: **false**.

- For more information, see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc.

Reference

- For adding a Bitbucket CA certificate into CodeReady Workspaces, see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc.

12.3.5.3. Configuring the Bitbucket and CodeReady Workspaces integration to use OAuth1

The following section describes the configuration of the OAuth 1 authentication that is needed for performing read and write operations with Bitbucket (BB) repositories. To use BB repositories with allowed Git operations, such as **clone** and **push**, register a BB endpoint with CodeReady Workspaces first, and configure the OAuth 1 authentication.

NOTE

This procedure requires:

- generating RSA key pairs
- generating a consumer key-secret pair
- creating an application link on the BB side
- configuring BB on the CodeReady Workspaces-server side

This procedure also describes how to activate OAuth 1 for Bitbucket Server to:

- Use devfiles hosted on a Bitbucket Server.
- Enable CodeReady Workspaces to obtain and renew Bitbucket Server Personal access tokens .

Prerequisites

- The **oc** tool is available.
- Bitbucket Server is available from CodeReady Workspaces server.
- An instance of CodeReady Workspaces.

Procedure

1. Generate an RSA key pair and a stripped-down version of the public key:

```
$ openssl genrsa -out <private.pem> 2048
```

```
$ openssl rsa -in <private.pem> -pubout > <public.pub>
```

```
$ openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in <private.pem> -out
<privatepkcs8.pem>
```

```
$ cat <public.pub> | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY----
-//g' | tr -d '\n' > <public-stripped.pub>
```

2. Generate a consumer key and a shared secret.

```
$ openssl rand -base64 24 > <bitbucket_server_consumer_key>
```

```
$ openssl rand -base64 24 > <bitbucket_shared_secret>
```

3. Configure an Application Link in Bitbucket to enable the communication from CodeReady Workspaces to Bitbucket Server.

   a. In Bitbucket Server, click the cog in the top navigation bar to navigate to **Administration** > **Application Links**.

   b. Enter the application URL: **https://codeready-<openshift_deployment_name>.<domain_name>** and click the **Create new link** button.

   c. In the warning message stating **No response was received from the URL** click the **Continue** button.

   d. Complete the **Link Applications** form and click the **Continue** button.

      Application Name

      <CodeReady Workspaces>

      Application Type

      Generic Application.

      Service Provider Name

      <CodeReady Workspaces>

      Consumer Key

      Paste the content of the ***<bitbucket_server_consumer_key>*** file.

      Shared secret

      Paste the content of the ***<bitbucket_shared_secret>*** file.

      Request Token URL

      ***<Bitbucket Server URL>*/plugins/servlet/oauth/request-token**

      Access token URL

      ***<Bitbucket Server URL>*/plugins/servlet/oauth/access-token**

      Authorize URL

      ***<Bitbucket Server URL>*/plugins/servlet/oauth/access-token**

      Create incoming link

      Enabled.

   e. Complete the **Link Applications** form and click the **Continue** button.

      Consumer Key

      Paste the content of the ***<bitbucket_server_consumer_key>*** file.

      Consumer name

      <CodeReady Workspaces>

      Public Key

      Paste the content of the ***<public-stripped.pub>*** file.

4. Create a OpenShift Secret in CodeReady Workspaces project containing the consumer and private keys.

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: <CodeReady Workspaces-namespace> 1
  labels:
    app.kubernetes.io/component: oauth-scm-configuration
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: '<scm-server-endpoint>' 2
type: Opaque
data:
  private.key: '<user-private-key>' 3
  consumer.key: '<bitbucket_server_consumer_key>' 4
EOF
```

1. CodeReady Workspaces namespace. The default is openshift-workspaces
2. Bitbucket Server URL
3. base64 encoded content of the ***<privatepkcs8.pem>*** file without first and last lines.
4. base64 encoded content of the ***<bitbucket_server_consumer_key>*** file.

Example

```
#!/usr/bin/env bash

NS=${1:-eclipse-che}
CONSUMER_KEY=$(cat ./certs/bitbucket_server_consumer_key)
PRIVATE_KEY=$(cat ./certs/privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' |
sed 's/-----END PRIVATE KEY-----//g' | tr -d '\n')
BITBUCKET_HOST='<your-bitbucket-host-here>'
unameOut="$(uname -s)"

case "${unameOut}" in
    Linux*)     BASE64_FUNC='base64 -w 0';;
    Darwin*)    BASE64_FUNC='base64';;
    CYGWIN*)    BASE64_FUNC='base64 -w 0';;
    MINGW*)     BASE64_FUNC='base64 -w 0';;
    *)          BASE64_FUNC='base64 -w 0'
esac

cat <<EOF | oc apply -n $NS -f -
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
```

```
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: https://$BITBUCKET_HOST
type: Opaque
data:
  private.key: $(echo -n $PRIVATE_KEY | $BASE64_FUNC)
  consumer.key: $(echo -n $CONSUMER_KEY | $BASE64_FUNC)
EOF
```

- See the whole script in this GitHub example.

Additional resources

- Bitbucket Server overview
- Download Bitbucket Server
- Bitbucket Server Personal access tokens
- How to generate public key to application link 3rd party applications
- Using AppLinks to link to other applications
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#authenticating-on-scm-server-with-a-personal-access-token.adoc.

12.3.5.4. Configuring GitLab servers

To use a GitLab server as a project sources supplier, register the GitLab server URL with CodeReady Workspaces using the **CHE_INTEGRATION_GITLAB_SERVER__ENDPOINTS** property and specify the hostname of the server to register.

Example

```
https://gitlab.apps.cluster-2ab2.2ab2.example.opentlc.com/
```

For additional examples of configuring GitLab servers see Understanding CodeReady Workspaces server advanced configuration using the Operator

Additional resources

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

12.3.5.5. Configuring GitLab OAuth2 with the codeready-server engine

OAuth2 for GitLab allows accepting factories from private GitLab repositories.

Prerequisites

- The GitLab server is running and available from CodeReady Workspaces.

Procedure

- Create a Authorized OAuth2 application in GitLab using CodeReady Workspaces as the application **Name** and RH-SSO GitLab endpoint URL as the value for **Redirect URI**. The callback URL default value is **https://keycloak-openshift-workspaces.*<DOMAIN>*/auth/realms/codeready/broker/gitlab/endpoint**, where ***<DOMAIN>*** is OpenShift cluster domain. Store the **Application ID** and **Secret** values. All three types of GitLab OAuth 2 applications are supported: User owned, Group owned and Instance-wide.

  1. Create a custom OIDC provider link on RH-SSO pointing to GitLab server. Fill the following fields:

     Client ID

     a value from the **Application ID** field provided by GitLab server in previous step;

     Client Secret

     a value from **Secret** field provided by GitLab server in previous step;

     Authorization URL

     a URL which have a **https://*<GITLAB_DOMAIN>*/oauth/authorize** format;

     Token URL

     a URL which have a **https://*<GITLAB_DOMAIN>*/oauth/token** format;

     Scopes

     set of scopes which must contain (but not limited to) the following set: **api write_repository openid**;

     Store Tokens

     needs to be enabled;

     Store Tokens Readable

     needs to be enabled

     NOTE

     - Substitute ***<GITLAB_DOMAIN>*** with the URL and port of the GitLab installation.

  2. Register the GitLab instance URL with the enabled OAuth 2 support in CodeReady Workspaces using the **CHE_INTEGRATION_GITLAB_OAUTH__ENDPOINT** property.

     WARNING

     - The GitLab instance URL must be present in the list of configured GitLab integration endpoints, set by the **CHE_INTEGRATION_GITLAB_SERVER__ENDPOINTS** property.

Additional resources

In case of having issues CodeReady Workspaces accessing GitLab related to TLS keys, consult with the following docs:

- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc.
- https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#deploying-che-with-support-for-git-repositories-with-self-signed-certificates.adoc.

12.3.5.6. Configuring GitLab OAuth2 with the Dev Workspace engine

Prerequisites

- The GitLab server is running and available from CodeReady Workspaces.

Procedure

1. Create an OAuth2 Authorized application in Gitlab:
   - Enter Che as the application **Name**.
   - Enter the CodeReady Workspaces OAuth callback URL as the **Redirect URI**. The default value is **https://codeready-openshift-workspaces.*<domain>*/api/oauth/callback**.
   - Check the **Confidential** and **Expire access tokens** checkboxes.
   - Under **Scopes**, check the **api**, **write_repository**, and **openid** checkboxes.
2. Click **Save application** and store the **Application ID** and **Secret** values.
3. Create a new secret in the project where CodeReady Workspaces is deployed.

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: gitlab-oauth-config
  namespace: <namespace> (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: gitlab
    che.eclipse.org/scm-server-endpoint: <your_GitLab_service_URL>
type: Opaque
data:
  id: <base64_encoded_GitLab_Application_ID>
  secret: <base64_encoded_GitLab_Secret>
EOF
```

(1) The CodeReady Workspaces namespace. The default is openshift-workspaces.

Additional resources

- Configure GitLab as an OAuth 2.0 authentication identity provider

12.3.6. Using protocol-based providers

RH-SSO supports SAML v2.0 and OpenID Connect v1.0 protocols.

12.3.7. Managing users using RH-SSO

You can add, delete, and edit users in the user interface. See RH-SSO User Management for more information.

12.3.8. Configuring CodeReady Workspaces to use an external RH-SSO installation

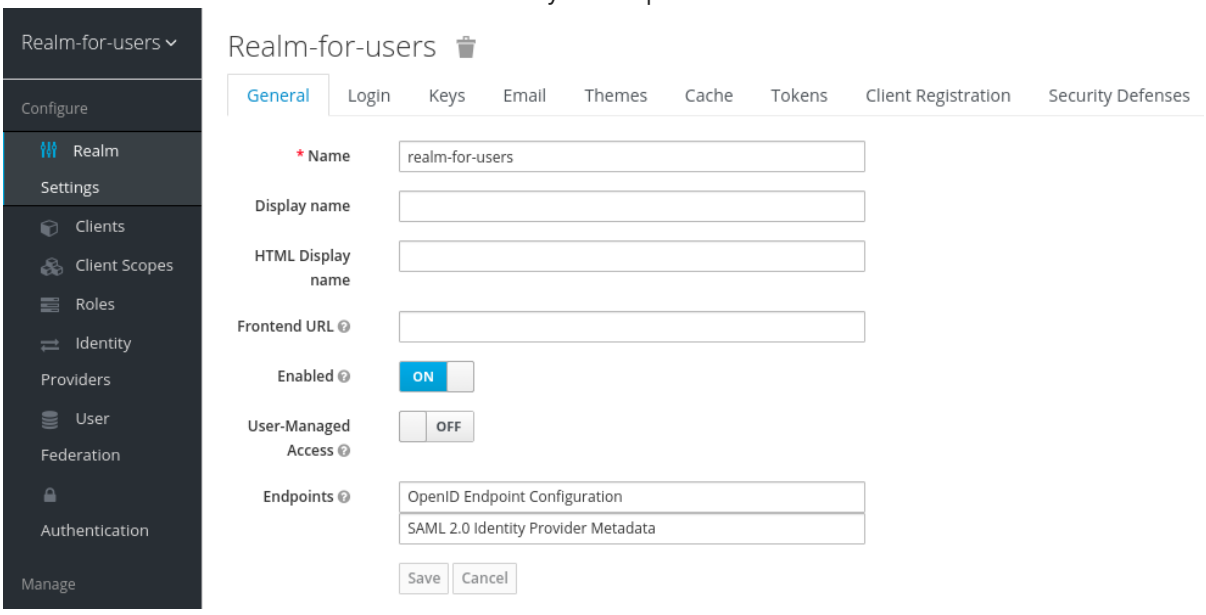
By default, CodeReady Workspaces installation includes the deployment of a dedicated RH-SSO instance. However, using an external RH-SSO is also possible. This option is useful when a user has an existing RH-SSO instance with already-defined users, for example, a company-wide RH-SSO server used by several applications.

Table 12.3. Placeholders used in examples

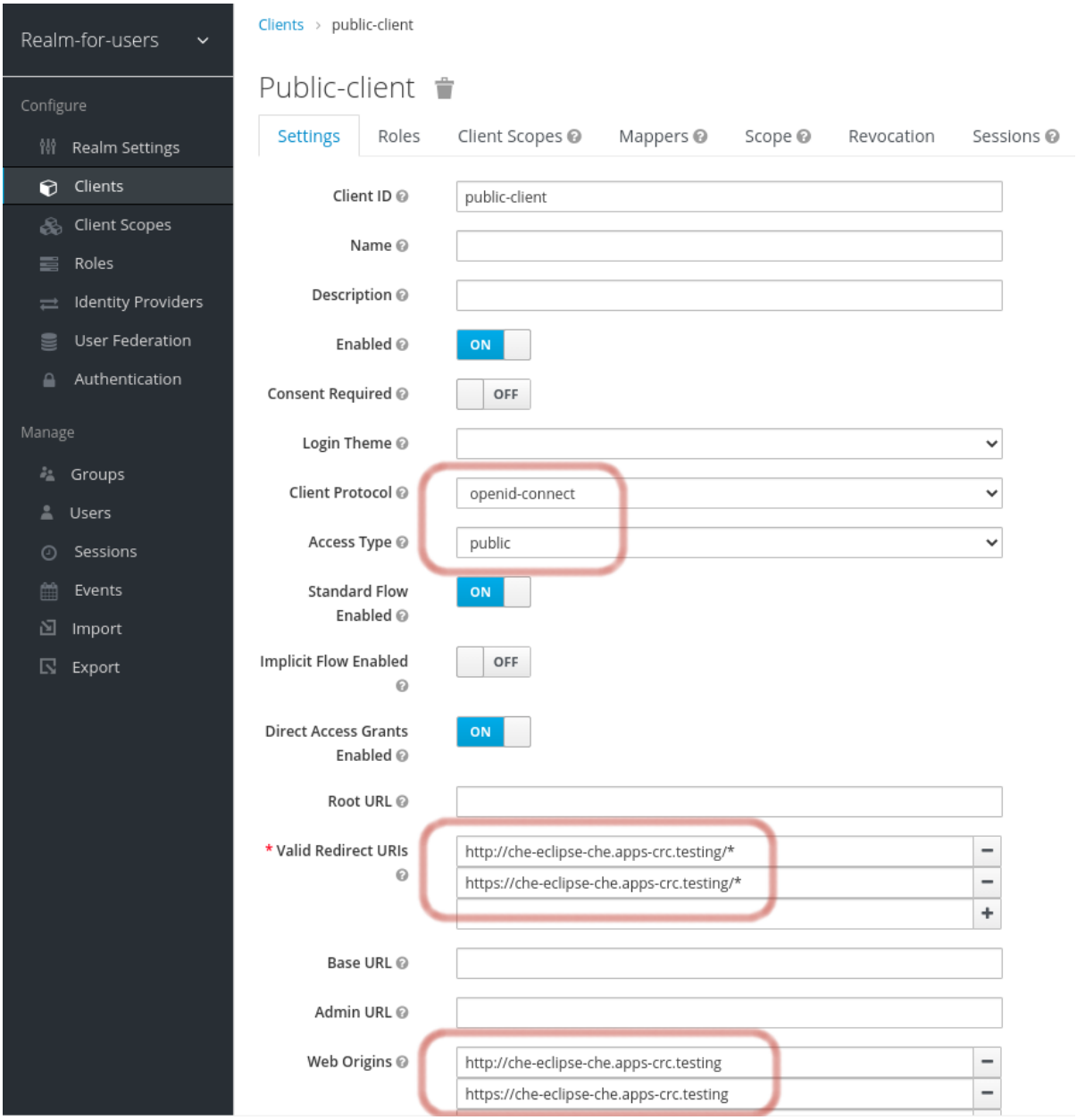
<provider-realm-name>	RH-SSO realm name intended for use by CodeReady Workspaces
<oidc-client-name>	Name of the **oidc** client defined in ***<provider-realm-name>***
<auth-base-url>	Base URL of the external RH-SSO server

Prerequisites

- In the administration console of the external installation of RH-SSO, define a realm that contains the users to connect to CodeReady Workspaces:



- In this **realm**, define an OIDC client that CodeReady Workspaces will use to authenticate the users. This is an example of such a client with the correct settings:



NOTE

- **Client Protocol** must be **openid-connect**.
- **Access Type** must be **public**. CodeReady Workspaces only supports the **public** access type.
- **Valid Redirect URIs** must contain at least two URIs related to the CodeReady Workspaces server, one using the **http** protocol and the other **https**. These URIs must contain the base URL of the CodeReady Workspaces server, followed by /* wildcards.
- **Web Origins** must contain at least two URIs related to the CodeReady Workspaces server, one using the **http** protocol and the other **https**. These URIs must contain the base URL of the CodeReady Workspaces server, without any path after the host.
  The number of URIs depends on the number of installed product tools.

- With CodeReady Workspaces that uses the default OpenShift OAuth support, user authentication relies on the integration of RH-SSO with OpenShift OAuth. This allows users to

log in to CodeReady Workspaces with their OpenShift login and have their workspaces created under personal OpenShift projects.
This requires setting up an OpenShift "RH-SSO Identity Provider". When using an external RH-SSO, configure the RH-SSO manually. For instructions, see the appropriate RH-SSO documentations for either OpenShift 3 or OpenShift 4.

- The configured RH-SSO has the options **Store Tokens** and **Stored Tokens Readable** enabled.

Procedure

1. Set the following properties in the **CheCluster** Custom Resource (CR):

```
spec:
  auth:
    externalIdentityProvider: true
    identityProviderURL: <auth-base-url>
    identityProviderRealm: <provider-realm-name>
    identityProviderClientId: <oidc-client-name>
```

2. When installing CodeReady Workspaces with OpenShift OAuth support enabled, set the following properties in the **CheCluster** Custom Resource (CR):

```
spec:
  auth:
    openShiftoAuth: true
# Note: only if the OpenShift "RH-SSO Identity Provider" alias is different from 'openshift-v3' or 'openshift-v4'
server:
  customCheProperties:
    CHE_INFRA_OPENSHIFT_OAUTH_IDENTITY_PROVIDER: <OpenShift "RH-SSO Identity Provider" alias>
```

12.3.9. Configuring SMTP and email notifications

Red Hat CodeReady Workspaces does not provide any pre-configured MTP servers.

To enable SMTP servers in RH-SSO:

1. Go to **che realm settings > Email**.

2. Specify the host, port, username, and password.

Red Hat CodeReady Workspaces uses the default theme for email templates for registration, email confirmation, password recovery, and failed login.

12.3.10. Enabling self-registration

Self-registration allows users to register themselves in a CodeReady Workspaces instance by accessing the CodeReady Workspaces server URL.

For CodeReady Workspaces installed without OpenShift OAuth support, self-registration is disabled by default, therefore the option to register a new user is not available on the login page.

Prerequisites

- You are logged in as an administrator.

Procedure

To enable self-registration of users:

1. Navigate to the **Realm Settings** menu on the left and open the **Login** tab.
2. Set **User registration** option to **On**.

12.4. CONFIGURING OPENSHIFT OAUTH

For users to interact with OpenShift, they must first authenticate to the OpenShift cluster. OpenShift OAuth is a process in which users prove themselves to a cluster through an API with obtained OAuth access tokens.

Authentication with the https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/end-user_guide/index#openshift-connector-overview.adoc is a possible way for CodeReady Workspaces users to authenticate with an OpenShift cluster.

The following section describes the OpenShift OAuth configuration options and its use with a CodeReady Workspaces.

12.4.1. Configuring OpenShift OAuth with initial user

Prerequisites

- The **oc** tool is available.
- **crwctl** management tool is available. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#using-the-chectl-management-tool.adoc.

Procedure

- Configure OpenShift identity providers on the cluster. See the Understanding identity provider configuration.
  When a user skips the Configuring step of OpenShift "RH-SSO Identity Provider", and the OpenShift cluster does not already contain a configured RH-SSO, CodeReady Workspaces creates an initial OpenShift user for the **HTPasswd** identity provider. Credentials of this user are stored in the **openshift-oauth-user-credentials** secret, located in the **openshift-config** namespace.

  Obtain the credentials for logging in to an OpenShift cluster and CodeReady Workspaces instance:

  1. Obtain OpenShift user name:

     ```
     $ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r
     '.data.user' | base64 -d
     ```

  2. Obtain OpenShift user password:

```
$ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r
'.data.password' | base64 -d
```

- Deploy CodeReady Workspaces using OperatorHub or the crwctl, see the **crwctl server:deploy** specification chapter. OpenShift OAuth will be enabled by default.

12.4.2. Configuring OpenShift OAuth without provisioning OpenShift initial OAuth user

The following procedure describes how to configure OpenShift OAuth without provisioning the initial OAuth user.

Prerequisites

- **crwctl** management tool is available. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#using-the-chectl-management-tool.adoc.

Procedure

1. If you have installed CodeReady Workspaces by using the Operator, configure the following values in the codeready-workspaces Custom Resource:

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

2. If you have installed CodeReady Workspaces by using the **crwctl** tool, use the **--che-operator-cr-patch-yaml** flag:

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

The **patch.yaml** file must contain the following:

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

12.4.3. Removing OpenShift initial OAuth user

The following procedure describes how to remove OpenShift initial OAuth user provisioned by Red Hat CodeReady Workspaces.

Prerequisites

- The **oc** tool installed.
- An instance of Red Hat CodeReady Workspaces running on OpenShift.
- Logged in to OpenShift cluster using the **oc** tool.

Procedure

1. Update codeready-workspaces custom resource:

```
$ oc patch checluster/codeready-workspaces -n openshift-workspaces --type=json -p \
'[{"op": "replace", "path": "/spec/auth/initialOpenShiftOAuthUser", "value": false}]'
```

12.5. CONFIGURING MINIKUBE WITH GITHUB AUTHENTICATION

On Minikube, crwctl provides a default OpenID Connect (OIDC) issuer, which can serve as a bridge to third party RH-SSO, such as GitHub. Dex is the default OIDC issuer, preconfigured with static users. Configure Dex to use GitHub authentication.

Prerequisites

- CodeReady Workspaces is installed on Minikube. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#installing-che-on-minikube.adoc.

Procedure

1. Get Minikube IP and remember it as ***<minikube_ip>***:

```
$ minikube ip
```

2. Create an OAuth App for your Minikube instance in GitHub. See GitHub documentation.

```
Application name: CodeReady Workspaces (1)
Homepage URL: https://<minikube_ip>.nip.io (2)
Authorization callback URL: https://dex.<minikube_ip>.nip.io/callback (3)
```

(1) Name is only displayed on GitHub. It is not used internally so it can be any name.

(2) Main URL to CodeReady Workspaces instance.

(3) Callback URL to Dex. crwctl deploys Dex on **dex.** subdomain.

3. In the GitHub OAuth application page, click **Generate a new client secret** and remember the value of the generated client secret as ***<client_secret>***.

4. Edit the Dex config map:

```
$ oc edit configmap dex -n dex
```

```
connectors:
- type: github
  id: github
  name: GitHub
  config:
    clientID: <client_id> (1)
    clientSecret: <client_secret> (2)
    redirectURI: https://dex.<minikube_ip>.nip.io/callback (3)
```

1. OAuth client id copied from GitHub OAuth application
2. OAuth client secret, generated at GitHub in previous step
3. Callback URL to Dex. This must match configuration in GitHub OAuth application from step 1.

Note: To remove Dex static users, delete all **enablePasswordDB** and **staticPasswords** sections.

1. Restart the Dex pod:

```
$ oc delete pod dex -n dex
```

Verification steps

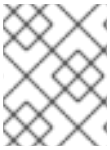
- Open CodeReady Workspaces URL. The dashboard displays GitHub login prompt.

12.6. REMOVING USER DATA

12.6.1. Removing user data according to GDPR

The General Data Protection Regulation (GDPR) law enforces the right for individuals to have personal data erased.

The following procedure describes how to remove a user's data from a cluster and the RH-SSO database.



NOTE

The following commands use the default OpenShift project, **openshift-workspaces**, as a user's example for the **-n** option.

Prerequisites

- A user or an administrator authorization token. To delete any other data except the data bound to a user account, **admin** privileges are required. The **admin** is a special CodeReady Workspaces administrator account pre-created and enabled using the **CHE_SYSTEM_ADMIN__NAME** and **CHE_SYSTEM_SUPER__PRIVILEGED__MODE = true** Custom Resource definitions.

```
spec:
 server:
   customCheProperties:
     CHE_SYSTEM_SUPER__PRIVILEGED__MODE: 'true'
     CHE_SYSTEM_ADMIN__NAME: '<admin-name>'
```

If needed, use commands below for creating the **admin** user:

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
  -p '{ "spec": { "server": {"customCheProperties":
{"CHE_SYSTEM_SUPER__PRIVILEGED__MODE": "true"} } }}' \
  -n openshift-workspaces
```

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
  -p '{ "spec": { "server": {"customCheProperties": {"CHE_SYSTEM_ADMIN__NAME":
"<admin-name>"} } }}' \
  -n openshift-workspaces
```

NOTE

All system permissions are granted to the administrative user. To configure the administrative user, use the **CHE_SYSTEM_ADMIN__NAME** property. The default value is **admin**. The system permissions are granted when the CodeReady Workspaces server starts. If the user record is not in the CodeReady Workspaces user database, the permissions are granted after the first login of the user.

Authorization token privileges:

- **admin** - Can delete all personal data of all users
- **user** - Can delete only the data related to the user

- A user or an administrator is logged in the OpenShift cluster with deployed CodeReady Workspaces.
- A user ID is obtained. Get the user ID using the commands below:
  - For the current user:

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user'
```

  - To find a user by name:

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/find?
name=<username>'
```

  - To find a user by email:

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/find?
email=<email>'
```

Example of obtaining a user ID

This example uses **vparfono** as a local user name.

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://che-vp-che.apps.che-dev.x6e0.p1.openshiftapps.com/api/user/find?
name=vparfono'
```

The user ID is at the bottom of the curl command output.

```
{
 "name": "vparfono",
 "links": [
   {
.
.
.
   }
 ],
 "email": "vparfono@redhat.com",
 "id": "921b6f33-2657-407e-93a6-fb14cf2329ce"
}
```

Procedure

1. Update the **codeready-workspaces CheCluster Custom** Resource (CR) definition to permit the removal of a user's data from the RH-SSO database:

```
$ oc patch checluster/codeready-workspaces \
  --patch "{\"spec\":{\"server\":{\"customCheProperties\":
{\"CHE_KEYCLOAK_CASCADE__USER__REMOVAL__ENABLED\": \"true\"}}}}" \
  --type=merge -n openshift-workspaces
```

2. Remove the data using the API:

```
$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-id>
```

Verification

Running the following command returns code **204** as the API response:

```
$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-id>
```

Additional resources

To remove the data of all users, follow the instructions for https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#uninstalling-che.adoc.