Red Hat CodeReady Workspaces 2.8

Administration Guide

Administering Red Hat CodeReady Workspaces 2.8

Last Updated: 2021-06-21

Red Hat CodeReady Workspaces 2.8 Administration Guide

Administering Red Hat CodeReady Workspaces 2.8

Robert Kratky
rkratky@redhat.com

Michal Maléř
mmaler@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Yana Hontyk
yhontyk@redhat.com

Legal Notice

Abstract

Information for administrators operating Red Hat CodeReady Workspaces.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

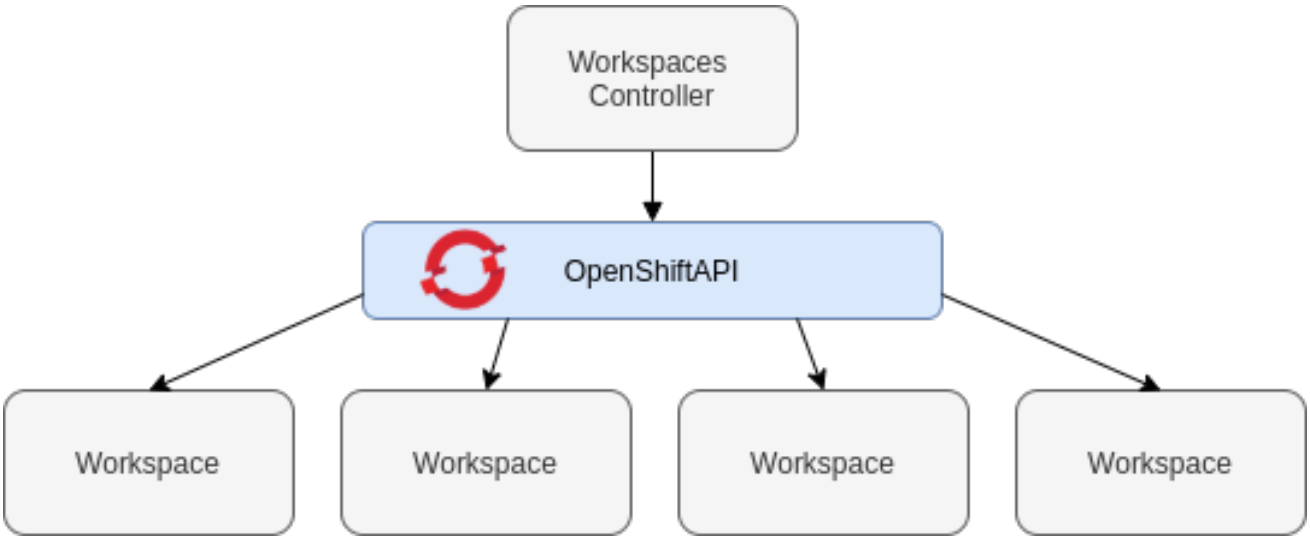
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases.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

CHAPTER 1. CODEREADY WORKSPACES ARCHITECTURE OVERVIEW

Red Hat CodeReady Workspaces components are:

- A central workspace controller: an always running service that manages users workspaces through the OpenShift API.
- Users workspaces: container-based IDEs that the controller stops when the user stops coding.

Figure 1.1. High-level CodeReady Workspaces architecture



When CodeReady Workspaces is installed on a OpenShift cluster, the workspace controller is the only component that is deployed. A CodeReady Workspaces workspace is created immediately after a user requests it.

Additional resources

- Section 1.1, “Understanding CodeReady Workspaces workspace controller”
- Section 1.2, “Understanding CodeReady Workspaces workspaces architecture”

1.1. UNDERSTANDING CODEREADY WORKSPACES WORKSPACE CONTROLLER

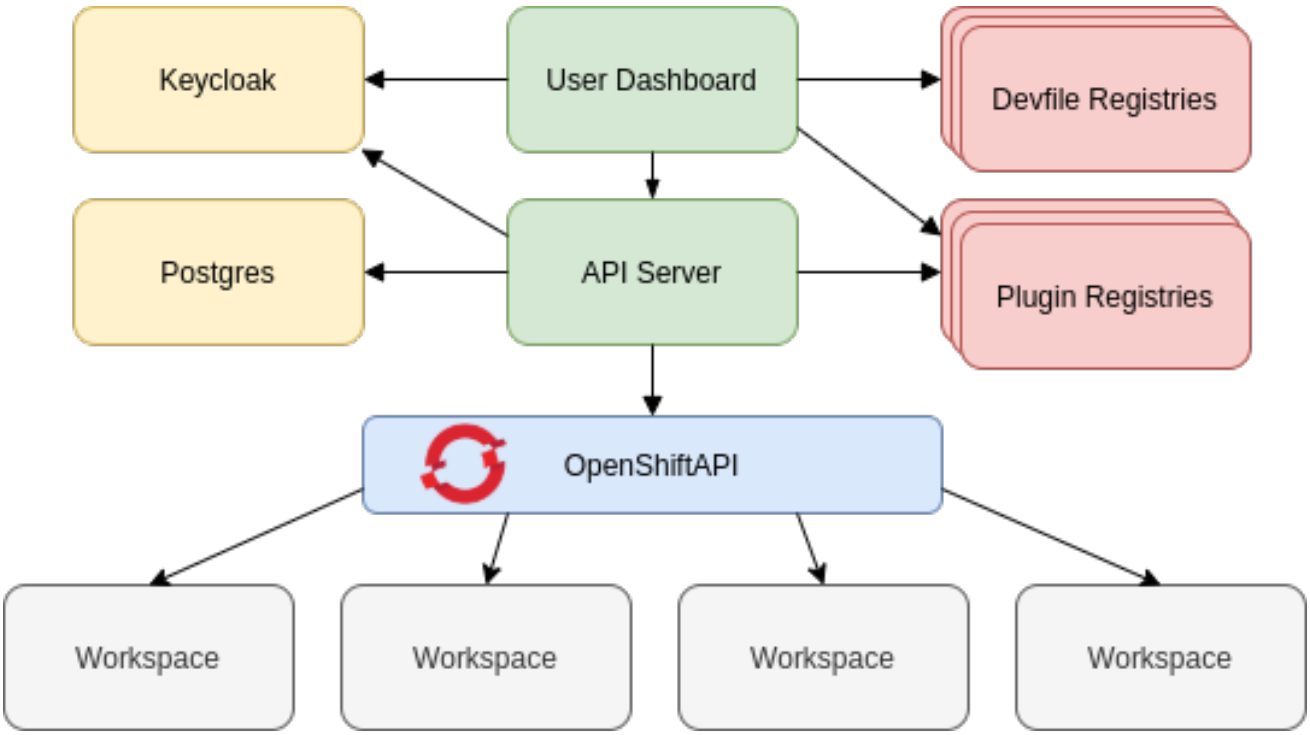
1.1.1. CodeReady Workspaces workspace controller

The workspaces controller manages the container-based development environments: CodeReady Workspaces workspaces. Following deployment scenarios are available:

- **Single-user**: The deployment contains no authentication service. Development environments are not secured. This configuration requires fewer resources. It is more adapted for local installations.
- **Multi-user**: This is a multi-tenant configuration. Development environments are secured, and this configuration requires more resources. Appropriate for cloud installations.

The following diagram shows the different services that are a part of the CodeReady Workspaces workspaces controller. Note that RH-SSO and PostgreSQL are only needed in the multi-user configuration.

Figure 1.2. CodeReady Workspaces workspaces controller



Additional resources

- Section 9.1, “Authenticating users”

1.1.2. CodeReady Workspaces server

The CodeReady Workspaces server is the central service of the workspaces controller. It is a Java web service that exposes an HTTP REST API to manage CodeReady Workspaces workspaces and, in multi-user mode, CodeReady Workspaces users.

Container image	**eclipse/che-server**

Additional resources

- Advanced configuration options for the CodeReady Workspaces server component

1.1.3. CodeReady Workspaces user dashboard

The user dashboard is the landing page of Red Hat CodeReady Workspaces. It is an Angular front-end application. CodeReady Workspaces users create, start, and manage CodeReady Workspaces workspaces from their browsers through the user dashboard.

Container image	**eclipse/che-server**

1.1.4. CodeReady Workspaces Devfile registry

The CodeReady Workspaces devfile registry is a service that provides a list of CodeReady Workspaces stacks to create ready-to-use workspaces. This list of stacks 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.

For more information about devfile registry customization, see the Customizing devfile registry section.

Container image	**registry.redhat.io/codeready-workspaces/devfileregistry-rhel8:2.8**

1.1.5. CodeReady Workspaces plug-in registry

The CodeReady Workspaces plug-in registry is a service that provides the list of plug-ins and editors for the 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.

Container image	**registry.redhat.io/codeready-workspaces/pluginregistry-rhel8:2.8**

1.1.6. CodeReady Workspaces and PostgreSQL

The PostgreSQL database is a prerequisite to configure CodeReady Workspaces in multi-user mode. The CodeReady Workspaces administrator can choose to connect CodeReady Workspaces to an existing PostgreSQL instance or let the CodeReady Workspaces deployment start a new dedicated PostgreSQL instance.

The CodeReady Workspaces server uses the database to persist user configurations (workspaces metadata, Git credentials). RH-SSO uses the database as its back end to persist user information.

Container image	**registry.redhat.io/rhel8/postgresql-96:1**

1.1.7. CodeReady Workspaces and RH-SSO

RH-SSO is a prerequisite to configure CodeReady Workspaces in multi-user mode. 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.

Container image	**registry.redhat.io/rh-sso-7/sso74-openshift-rhel8:7.4**

1.2. UNDERSTANDING CODEREADY WORKSPACES WORKSPACES ARCHITECTURE

1.2.1. 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 related components, such as:

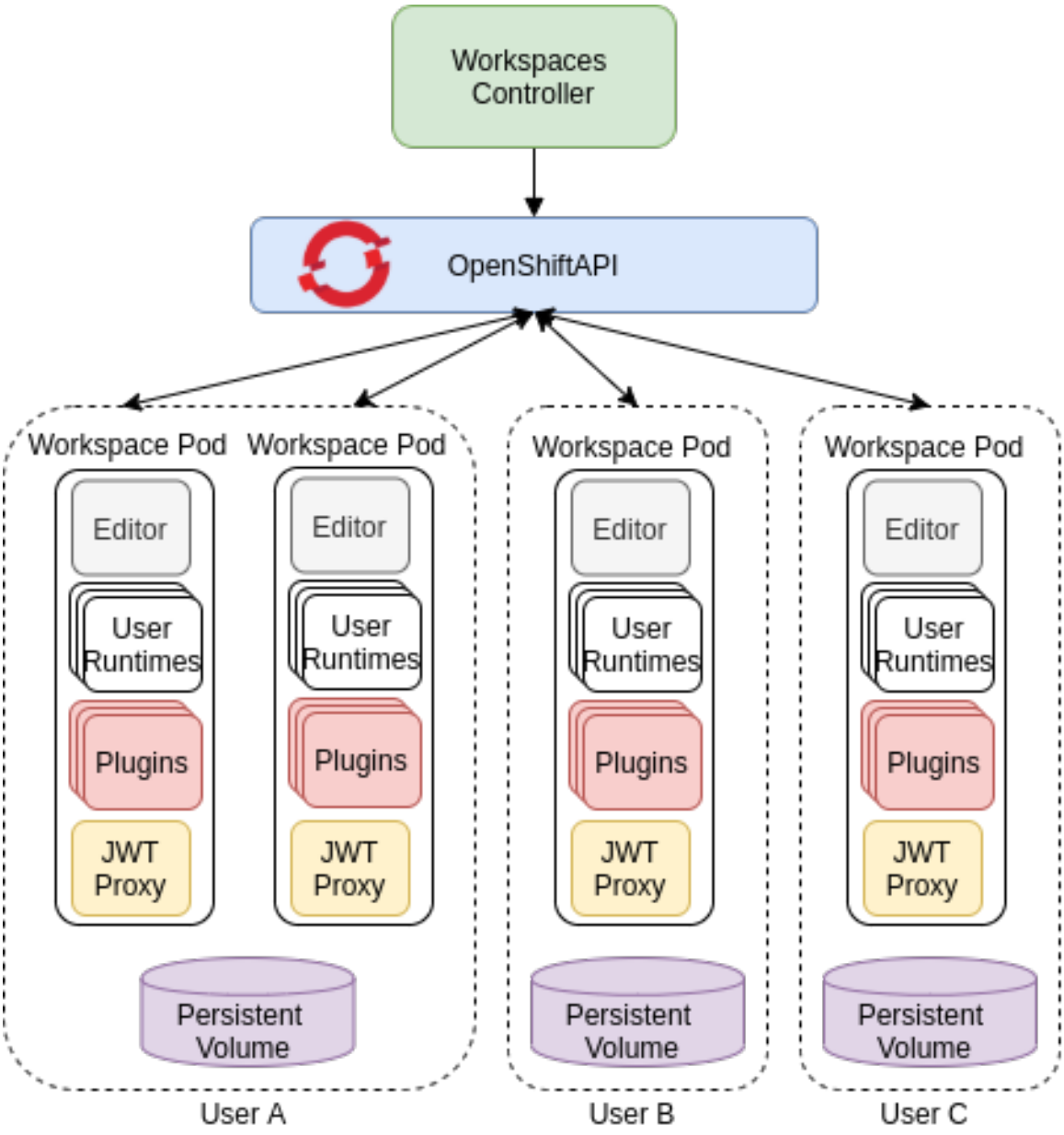
- ConfigMaps
- services
- endpoints
- ingresses/routes
- secrets
- PVs

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 1.3. CodeReady Workspaces workspace components



In the diagram, there are three running workspaces: two belonging to **User A** and one to **User C**. A fourth workspace is getting provisioned where the plug-in broker is verifying and completing the workspace configuration.

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

1.2.2. CodeReady Workspaces workspace components

This section describes the components of a CodeReady Workspaces workspace.

1.2.2.1. 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 (VS Code). It has a plug-in system that supports VS 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

1.2.2.2. 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.

1.2.2.3. CodeReady Workspaces workspace JWT proxy

The JWT proxy is responsible for securing the communication of the CodeReady Workspaces workspace services. The CodeReady Workspaces workspace JWT proxy is included in a CodeReady Workspaces workspace only if the CodeReady Workspaces server is configured in multi-user mode.

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**

1.2.2.4. 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).

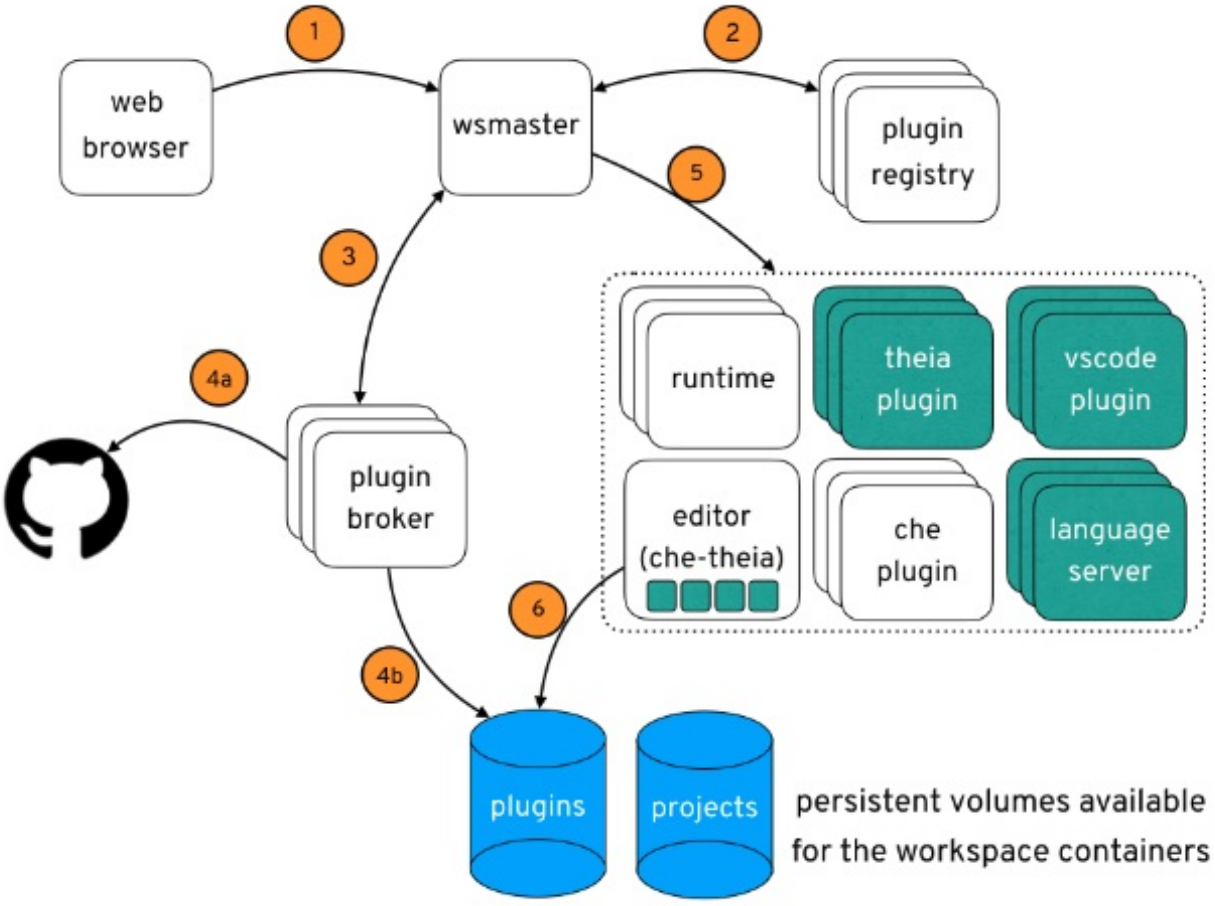
The main goal of the plug-in broker is to decouple the CodeReady Workspaces plug-ins definitions from the actual plug-ins that CodeReady Workspaces can support. With brokers, CodeReady Workspaces can support different plug-ins without updating the CodeReady Workspaces server.

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**

1.2.3. 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.

CHAPTER 2. CALCULATING CODEREADY WORKSPACES RESOURCE REQUIREMENTS

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 Advanced configuration options for the CodeReady Workspaces server component 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 1.1, “Understanding CodeReady Workspaces workspace controller” .

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 Configuring a workspace using a devfile .

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. When CodeReady Workspaces is running in multiuser mode, it loads the **JWT Proxy** component. The JWT Proxy 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 presents the default requirements for all workspace components. It also presents the corresponding CodeReady Workspaces server property 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
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:
If present, the **memoryLimit** and **memoryRequest** attributes of the **containers** section of the **meta.yaml** file define the memory limit of the **chePlugins** or **cheEditors** components. CodeReady Workspaces automatically sets the memory request to match the memory limit in case it is not specified explicitly.

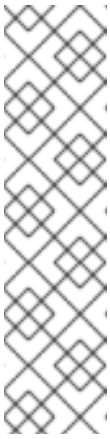
Example 2.1. The chePlugin che-incubator/typescript/latest

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 Systems (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 Systems (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 che-plugin-registry GitHub 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 Advanced configuration options for the CodeReady Workspaces server component .

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
   If present, the **memoryLimit** and **memoryRequest** attributes of the devfile define the memory limit of a **dockerimage** container. CodeReady Workspaces automatically sets the memory request to match the memory limit in case 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 1.2, “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: guestbook-nodejs-sample-
projects:
 - name: guestbook-nodejs-sample
   source:
     type: git
     location: "https://github.com/l0rd/nodejs-sample"
components:
 - type: chePlugin
   id: che-incubator/typescript/latest
 - type: kubernetes
   alias: guestbook-frontend
   reference: https://raw.githubusercontent.com/l0rd/nodejs-sample/master/kubernetes-
manifests/guestbook-frontend.deployment.yaml
   mountSources: true
   entrypoints:
     - command: ['sleep']
       args: ['infinity']
```

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	128 MiB
Workspace	vscode-typescript (**chePlugin**)	512 MiB	512 MiB
Workspace	frontend (**kubernetes**)	1 GiB	512 MiB
JWT Proxy	verifier	128 MiB	128 MiB
	Total	**2.25 GiB**	**1.75 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 VS 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 **kubernetes** component type: a memory limit of **1 GiB** and a memory request of **512 MiB**. This is because the **kubernetes** component references a **Deployment** manifest that has a container specification with no resource limits or requests.
- The JWT container requires **128 MiB** of memory.

Adding all together results in **1.75 GiB** of memory requests with a **2.25 GiB** limit.

Additional resources

- Chapter 1, *CodeReady Workspaces architecture overview*
- Configuring the CodeReady Workspaces installation
- Advanced configuration options for the CodeReady Workspaces server component
- Configuring a workspace using a devfile
- A minimal devfile
- Section 9.1, “Authenticating users”
- Eclipse Che plugin registry - GitHub 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: Configuring a workspace using a devfile .

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-2.8-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 Configuring a workspace using a devfile .

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.
globalMemoryLimit	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.
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.

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

- Configuring a workspace using a devfile .
- 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

- NodeJS 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.
- Valid content for the **meta.yaml** file describing the plug-in to add. See: Publishing metadata for a VS Code extension.

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-2.8-rhel-8
```

2. In the **./dependencies/che-plugin-registry/v3/plugins/** directory, create new directories ***<publisher>*/*<plugin-name>*/*<plugin-version>***/ and a **meta.yaml** file in the last directory.

Example 3.3. File organization for a plugin

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

3. Add valid content to the **meta.yaml** file. See: Publishing metadata for a VS Code extension .
4. Create a file named **latest.txt** with content the name of the latest ***<plugin-version>*** directory.

Example 3.4. Example plug-in files tree

```
$ tree che-plugin-registry/v3/plugins/redhat/java/
```

```
che-plugin-registry/v3/plugins/redhat/java/
├── 0.38.0
│   └── meta.yaml
├── 0.43.0
│   └── meta.yaml
├── 0.45.0
│   └── meta.yaml
├── 0.46.0
│   └── meta.yaml
├── 0.50.0
│   └── meta.yaml
└── latest.txt
$ cat che-plugin-registry/v3/plugins/redhat/java/latest.txt
0.50.0
```

5. 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.

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 3.5. 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 3.6. 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 3.7. 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 3.8. 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 **Get Started** and **Create Custom Workspace** tab of 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. 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 Firefox

This section describes how to view the RH-SSO IDE client diagnostics or error information in the 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 che metrics.
- Prometheus 2.9.1 or higher 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, there is a single job called **che**, which 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 in the Prometheus documentation.

Additional resources

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

CHAPTER 6. 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.

6.1. 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.

6.2. 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.

6.3. 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 6.3.1, “Installing Jaeger using OperatorHub on OpenShift 4”
- Section 6.3.2, “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.

6.3.1. 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.

6.3.2. 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.

6.4. ENABLING METRICS COLLECTION

Prerequisites

- Installed Jaeger v1.12.0 or above. See instructions at Section 6.3, “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 Advanced configuration options for the CodeReady Workspaces server component.
- For custom configuration of Jaeger, see the list of Jaeger client environment variables.

6.5. VIEWING CODEREADY WORKSPACES TRACES IN JAEGER UI

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

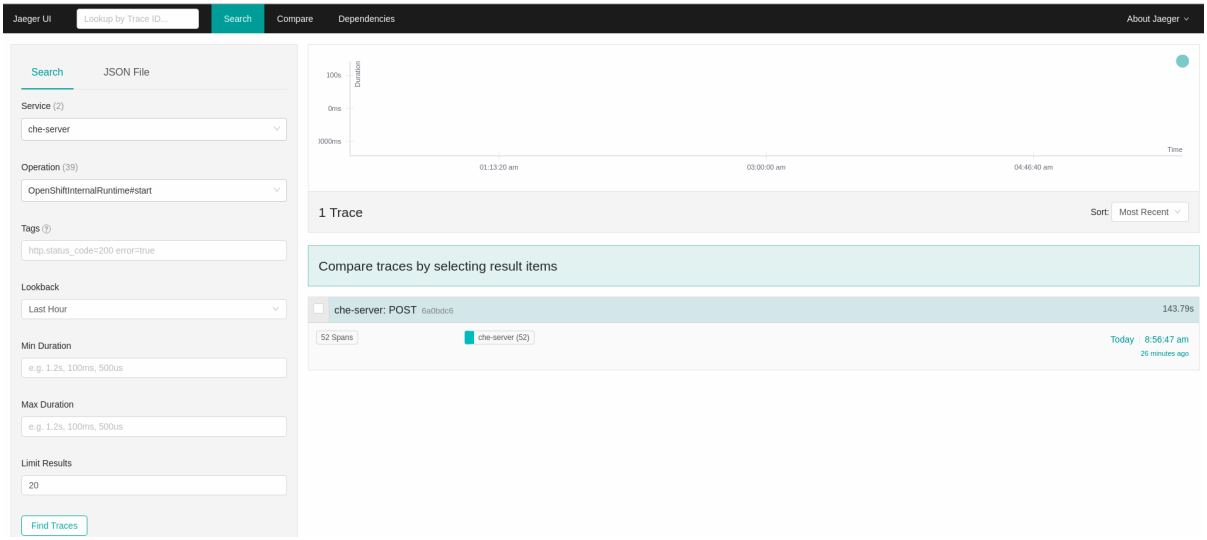
Procedure

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:

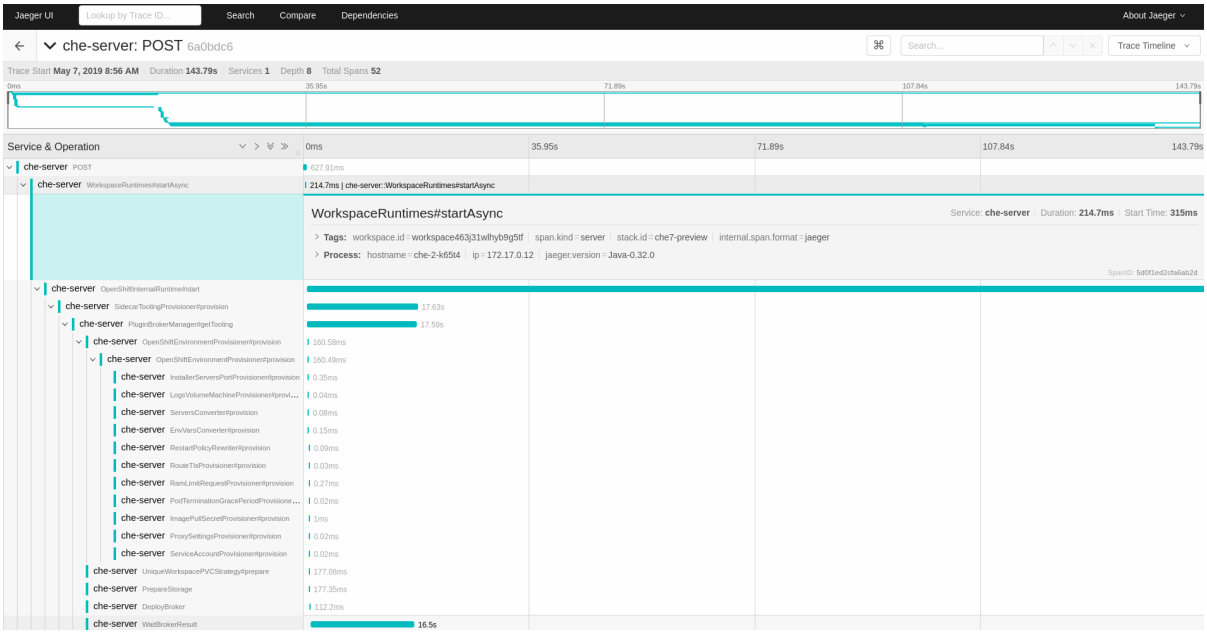
1. In the **Search** panel on the left, filter spans by the operation name (span name), tags, or time and duration.

Figure 6.1. 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 6.2. Expanded tracing tree



6.6. 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.

6.6.1. 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 7. BACKUP AND DISASTER RECOVERY

This section describes aspects of the CodeReady Workspaces backup and disaster recovery.

7.1. EXTERNAL DATABASE SETUP

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

By default, the CodeReady Workspaces Operator creates and manages the database deployment.

However, the CodeReady Workspaces Operator does not support full life-cycle capabilities, such as backups and recovery.

For a business-critical setup, configure an external database with the following recommended disaster-recovery options:

- High Availability (HA)
- Point In Time Recovery (PITR)

Configure an external PostgreSQL instance on-premises or use a cloud service, such as Amazon Relational Database Service (Amazon RDS). With Amazon RDS, it is possible to deploy production databases in a Multi-Availability Zone configuration for a resilient disaster recovery strategy with daily and on-demand snapshots.

The recommended configuration of the example database is:

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)

7.1.1. Configuring external PostgreSQL

Procedure

1. Use the following SQL script to create user and database for the CodeReady Workspaces server to persist workspaces metadata etc:

```
CREATE USER <database-user> WITH PASSWORD '<database-password>' (1) (2)
CREATE DATABASE <database>                                       (3)
GRANT ALL PRIVILEGES ON DATABASE <database> TO <database-user>
ALTER USER <database-user> WITH SUPERUSER
```

(1) CodeReady Workspaces server database username

(2) CodeReady Workspaces server database password

(3) CodeReady Workspaces server database name

2. Use the following SQL script to create database for RH-SSO back end to persist user information:

```
CREATE USER keycloak WITH PASSWORD '<identity-database-password>' (1)
CREATE DATABASE keycloak
GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak
```

(1) RH-SSO database password

7.1.2. Configuring CodeReady Workspaces to work with an external PostgreSQL

Prerequisites

- The **oc** tool is available.

Procedure

1. Pre-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 with applying a patch. For example:

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

patch.yaml should contain the following to make the Operator skip deploying 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

Additional resources

- PostgreSQL
- RDS

7.2. PERSISTENT VOLUMES BACKUPS

Persistent Volumes (PVs) store the CodeReady Workspaces workspace data similarly to how workspace data is stored for desktop IDEs on the local hard disk drive.

To prevent data loss, back up PVs periodically. The recommended approach is to use storage-agnostic tools for backing up and restoring OpenShift resources, including PVs.

7.2.1. Recommended backup tool: Velero

Velero is an open-source tool for backing up OpenShift applications and their PVs. Velero allows you to:

- Deploy in the cloud or on premises.
- Back up the cluster and restore 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 dependent on the underlying storage system. For example, solutions that are Gluster or Ceph-specific.

Additional resources

- Persistent Volumes documentation
- Gluster documentation
- Ceph documentation
- Velero on GitHub

CHAPTER 8. CACHING IMAGES FOR FASTER WORKSPACE START

To improve the start time performance of CodeReady Workspaces workspaces, use the Image Puller. The Image Puller is an additional OpenShift deployment. It creates a *DaemonSet* downloading and running the relevant container images on each node. These images are already available when a CodeReady Workspaces workspace starts.

The Image Puller provides the following parameters for configuration.

Table 8.1. 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 8.2, "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 8.2, "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**
IMAGES	Semicolon separated list of images to pull, in the format ***<name1>=<image1>;<name2>=<image2>*** See Section 8.1, "Defining the list of images to pull".	

Parameter	Usage	Default
NODE_SELECTOR	Node selector to apply to the Pods created by the DaemonSet	**'{}'**

Additional resources

- Section 8.1, "Defining the list of images to pull"
- Section 8.2, "Defining the memory parameters for the Image Puller".
- Section 8.3, "Installing Image Puller using the CodeReady Workspaces Operator"
- Section 8.4, "Installing Image Puller on OpenShift 4 using OperatorHub"
- Section 8.5, "Installing Image Puller on OpenShift using OpenShift templates"
- Kubernetes Image Puller source code repository

8.1. DEFINING THE LIST OF IMAGES TO PULL

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. Get the list of relevant container images.

   Example 8.1. Getting the list of all relevant images for CodeReady Workspaces

   ```
   $ curl -sSLo- https://raw.githubusercontent.com/redhat-developer/codeready-workspaces-operator/crw-2.8-rhel-8/manifests/codeready-workspaces.csv.yaml | \
     yq -r '.spec.relatedImages[]'
   ```

2. Exclude from the list the container images not containing the **sleep** command.

   Example 8.2. Images incompatibles with {image-puller-short}, missing the sleep command

   - **FROM scratch** images.
   - **che-machine-exec**

3. Exclude from the list the container images mounting volumes in Dockerfile.

Additional resources

- Section 8.2, “Defining the memory parameters for the Image Puller” .
- Section 8.4, “Installing Image Puller on OpenShift 4 using OperatorHub”
- Section 8.5, “Installing Image Puller on OpenShift using OpenShift templates”

8.2. 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 8.1, “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 8.4, “Installing Image Puller on OpenShift 4 using OperatorHub”
- Section 8.5, “Installing Image Puller on OpenShift using OpenShift templates”

8.3. INSTALLING IMAGE PULLER USING THE CODEREADY WORKSPACES OPERATOR

This section describes how to use the CodeReady Workspaces Operator to install the Image Puller. This is a Community-supported technology preview feature.

Prerequisites

- Section 8.1, “Defining the list of images to pull”
- Section 8.2, “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 Installing CodeReady Workspaces on OpenShift 4 using OperatorHub

Procedure

1. Edit the **CheCluster** Custom Resource and set **.spec.imagePuller.enable** to **true**

Example 8.3. Enabling Image Puller in the CheCluster Custom Resource

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

UNINSTALLING IMAGE PULLER USING CODEREADY WORKSPACES OPERATOR

- 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.

Example 8.4. Configuring 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: <{image-puller-images}>
```

Verification steps

1. OpenShift creates a **{image-puller-operator-id}** Subscription.

2. The **eclipse-che namespace** contains a **community supported Kubernetes Image Puller Operator ClusterServiceVersion**:

```
$ oc get clusterserviceversions
```

3. The **eclipse-che namespace** contains these deployments: **kubernetes-image-puller** and **{image-puller-deployment-id}**.

```
$ oc get deployments
```

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

```
$ oc get kubernetesimagepullers
```

8.4. 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 8.1, "Defining the list of images to pull"
- Section 8.2, "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.

8.5. 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 8.1, "Defining the list of images to pull" .
- Section 8.2, "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 8.2. 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-rhel8:2.8**
IMAGE_TAG	The image tag to pull	**latest**
SERVICEACCOUNT_NAME	The name of the ServiceAccount created and used by the deployment	**{image-puller-serviceaccount-name}**

Table 8.3. 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	**{image-puller-images}**
NODE_SELECTOR	The value of **NODE_SELECTOR** in the ConfigMap	**"{}"**

Table 8.4. Image Puller OpenShift templates parameters in serviceaccount.yaml

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

3. Create an OpenShift project to host the Image Puller:

```
$ oc new-project <kubernetes-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 <kubernetes-image-puller>
```

2. Verify the values of the *<kubernetes-image-puller>* **ConfigMap**.

```
$ oc get configmap <kubernetes-image-puller> --output yaml
```

CHAPTER 9. MANAGING IDENTITIES AND AUTHORIZATIONS

This section describes different aspects of managing identities and authorizations of Red Hat CodeReady Workspaces.

9.1. 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.

9.1.1. Authenticating to the CodeReady Workspaces server

9.1.1.1. 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.

9.1.1.2. 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.

9.1.1.3. 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.

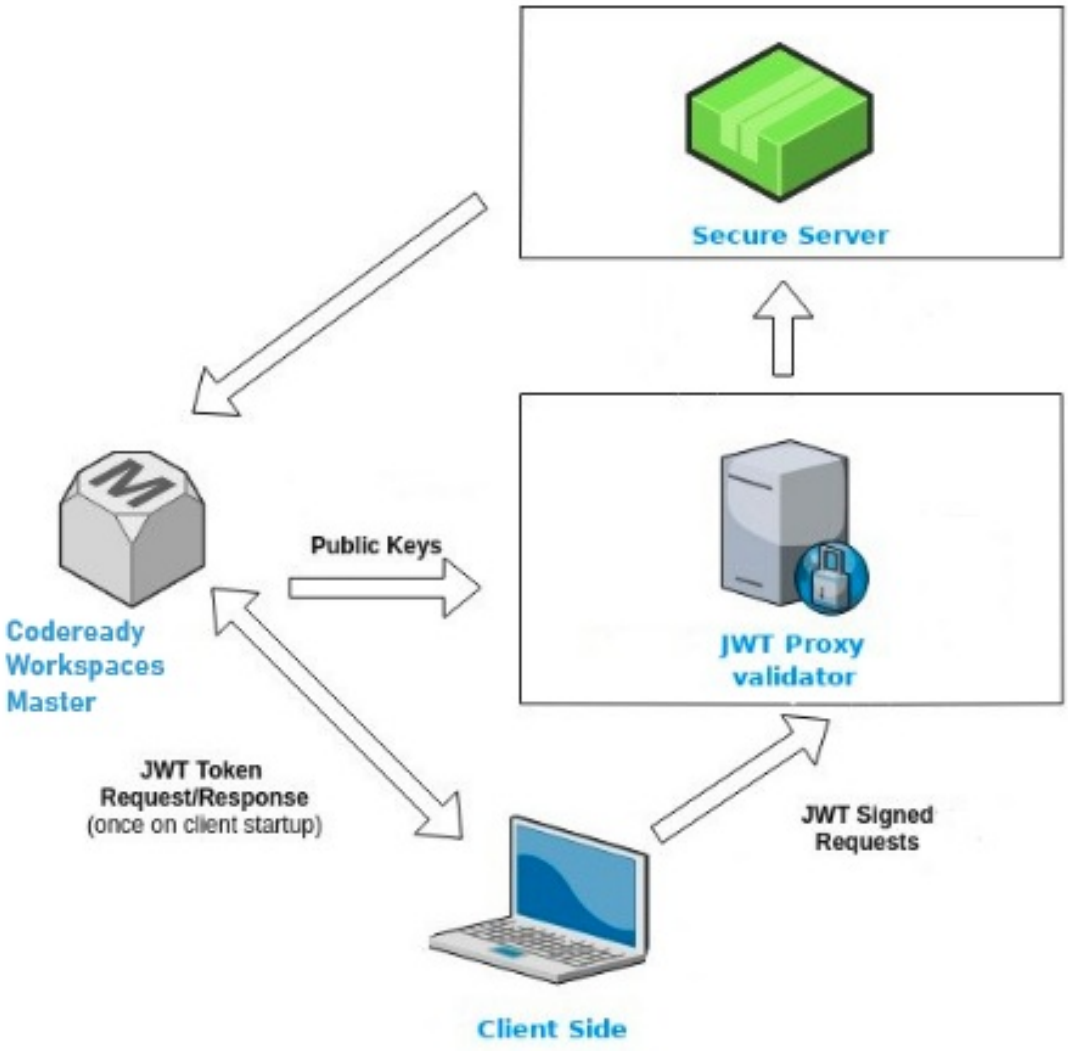
Access the CodeReady Workspaces Swagger interface at **https://codeready-<openshift_deployment_name>.<domain_name>/swagger**. The user must be signed in through RH-SSO, so that the access token is included in the **Request** header.

9.1.2. 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 9.1. Authentication inside a workspace



9.1.2.1. 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'
```

9.1.2.2. 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 than 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.

9.1.2.3. 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.

9.2. 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**.

9.2.1. 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 9.1. 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.

9.2.2. 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 9.2. 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 who is configured in the **CHE_SYSTEM_ADMIN__NAME** property (the default is **admin**). The system permissions are granted when the CodeReady Workspaces server starts. If the user is not present in the CodeReady Workspaces user database, it happens after the first user's login.

9.2.3. 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.

9.2.4. 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.

9.2.5. 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.

9.2.6. 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.

9.2.7. Sharing CodeReady Workspaces permissions

A user with **setPermissions** privileges can share a workspace and grant **read**, **use**, **run**, **configure**, or **setPermissions** privileges for other users.

Procedure

To share workspace permissions:

1. Select a workspace in the user dashboard.
2. Navigate to the **Share** tab and enter the email IDs of the users. Use commas or spaces as separators for multiple emails.

9.3. CONFIGURING AUTHORIZATION

9.3.1. 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.

9.3.2. 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.

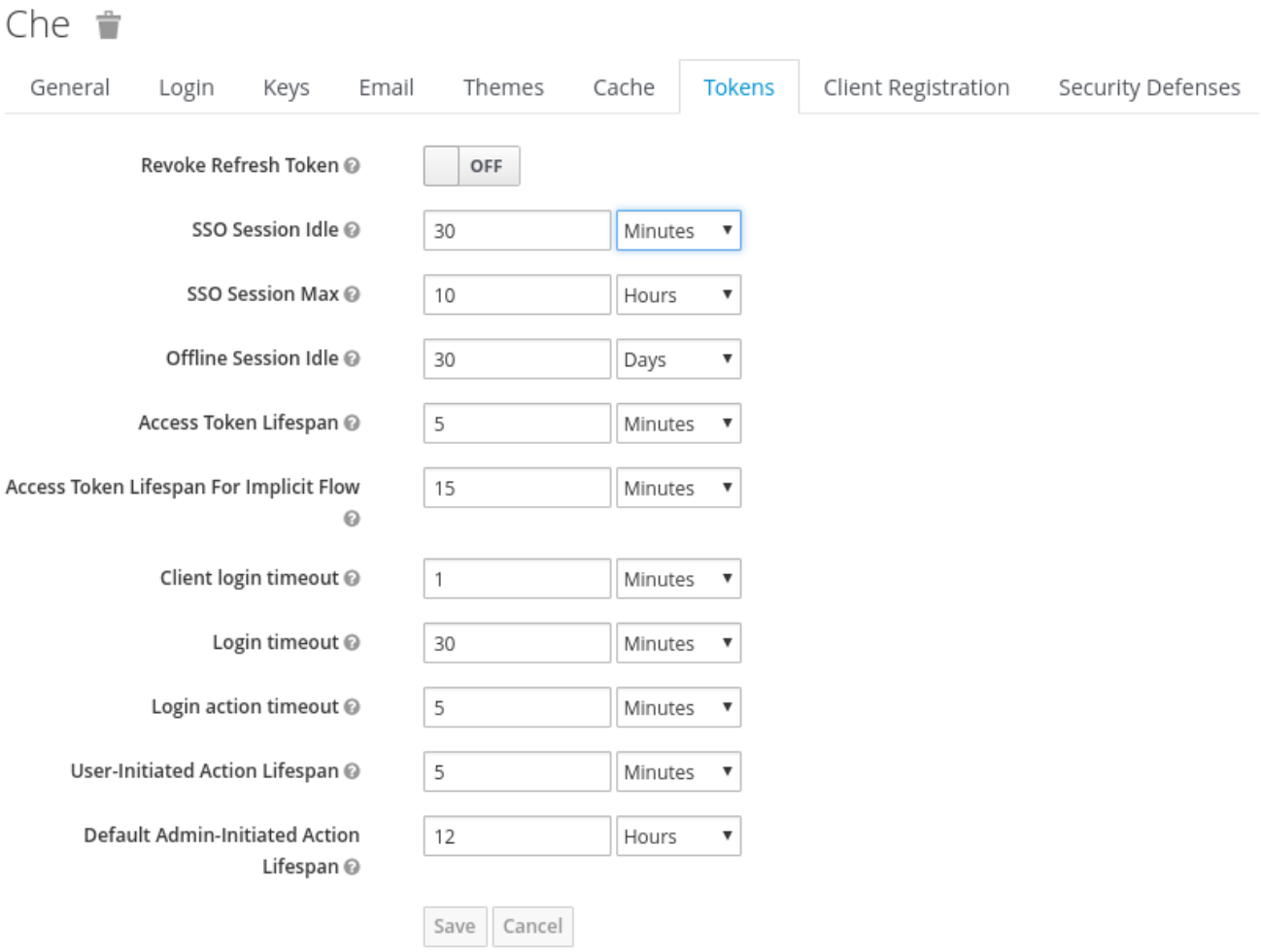
CORS error

Occurs when you have an invalid web origin.

9.3.3. Configuring RH-SSO tokens

A user token expires after 30 minutes by default.

You can change the following RH-SSO token settings:



9.3.4. 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.

9.3.5. 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.

9.3.5.1. Configuring GitHub OAuth

OAuth for GitHub allows for automatic SSH key upload to GitHub.

Prerequisites

- The **oc** tool is available.

Procedure

- Create a OAuth application in GitHub using CodeReady Workspaces URL as the value for the application **Homepage URL** and RH-SSO GitHub endpoint URL as the value for Authorization callback URL. The default values are **https://codeready-openshift-workspaces.*<DOMAIN>*/** and **https://keycloak-openshift-workspaces.*<DOMAIN>*/auth/realms/codeready/broker/github/endpoint** respectively, where ***<DOMAIN>*** is OpenShift cluster domain.

  1. 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

  2. If CodeReady Workspaces was already installed wait until rollout of RH-SSO component finishes.

9.3.5.2. Configuring Bitbucket Server OAuth 1

This procedure describes how to activate OAuth 1 for Bitbucket Server to:

- Use devfiles hosted on a Bitbucket Server.
- Authenticating users on private repositories of SCM servers.

It enables 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.

Procedure

1. Generate a 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. Create a Kubernetes Secret in CodeReady Workspaces namespace containing the consumer and private keys.

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-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: bitbucket
    che.eclipse.org/scm-server-endpoint: <...> 2
type: Opaque
data:
  private.key: <...> 3
  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.

4. 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**.

a. Enter the application URL: **https://codeready-<openshift_deployment_name>.<domain_name>** and click the **Create new link** button.

a. On the warning message stating "No response was received from the URL" click the **Continue** button.

a. Fill-in 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.

b. Fill-in 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.

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
- Authenticating users on private repositories of SCM servers.

9.3.6. Using protocol-based providers

RH-SSO supports SAML v2.0 and OpenID Connect v1.0 protocols.

9.3.7. Managing users using RH-SSO

You can add, delete, and edit users in the user interface. See RH-SSO User Management for more information.

9.3.8. Configuring CodeReady Workspaces to use an external RH-SSO installation

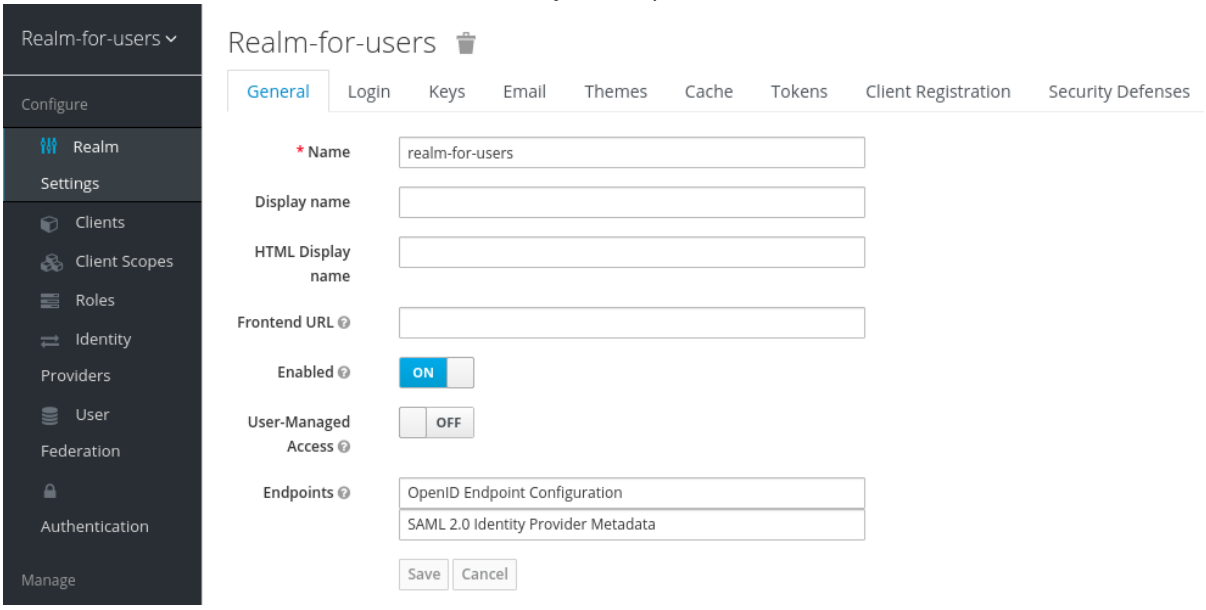
By default, CodeReady Workspaces installation includes the deployment of a dedicated RH-SSO instance. However, using an external RH-SSO is also possible. This option is useful when a user has an existing RH-SSO instance with already-defined users, for example, a company-wide RH-SSO server used by several applications.

Table 9.3. Placeholders used in examples

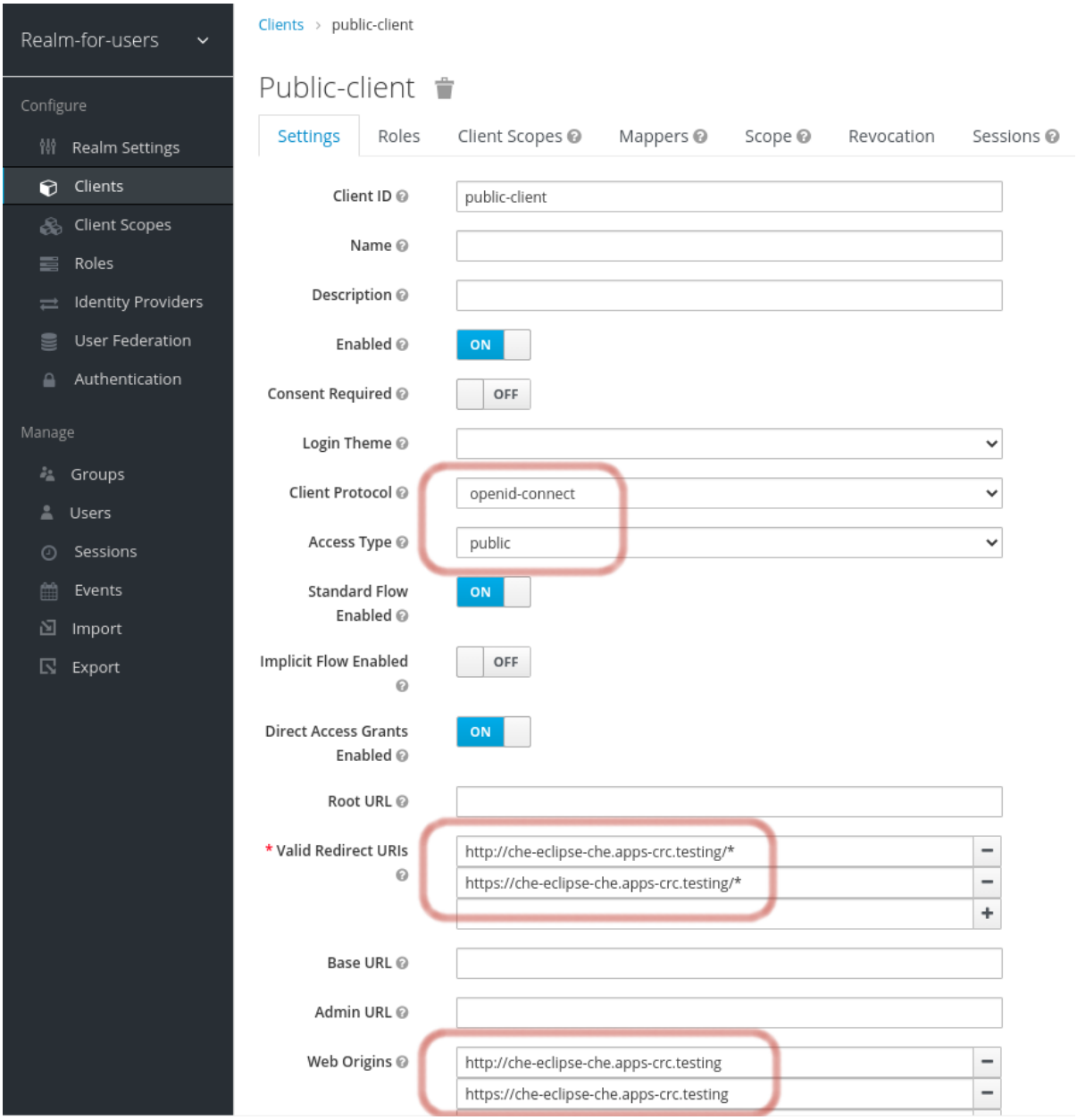
<provider-realm-name>	Identity provider realm name intended for use by CodeReady Workspaces
<oidc-client-name>	Name of the **oidc** client defined in ***<provider-realm-name>***
<auth-base-url>	Base URL of the external RH-SSO server

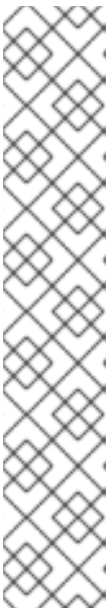
Prerequisites

- In the administration console of the external installation of RH-SSO, define a realm containing the users intended 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 identity provider ins RH-SSO. When using an external RH-SSO, configure the identity provider manually. For instructions, see the appropriate RH-SSO documentations for either link:OpenShift 3[OpenShift 3] or link:OpenShift 4[OpenShift 4].

- The configured identity provider 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 identity provider alias is different from 'openshift-v3' or
'openshift-v4'
server:
  customCheProperties:
    CHE_INFRA_OPENSHIFT_OAUTH__IDENTITY__PROVIDER: <OpenShift identity
provider alias>
```

9.3.9. 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.

9.3.10. 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**.

9.4. 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 OpenShift Connector overview 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.

9.4.1. Configuring OpenShift OAuth with initial user

Prerequisites

- The **oc** tool is available.
- **crwctl** management tool is available. See Using the crwctl management tool.

Procedure

- Configure OpenShift identity providers on the cluster. See the Understanding identity provider configuration.
  When a user skips the Configuring step of OpenShift identity providers, and the OpenShift cluster does not already contain configured identity providers, 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-workspaces 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-workspaces -o json | jq -r '.data.user' | base64 -d
     ```

  2. Obtain OpenShift user password:

     ```
     $ oc get secret openshift-oauth-user-credentials -n openshift-workspaces -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.

9.4.2. Configuring OpenShift OAuth without provisioning OpenShift initial OAuth user

The following procedure describes how to configure OpenShift OAuth without provisioning OpenShift initial OAuth user.

Prerequisites

- **crwctl** management tool is available. See Using the crwctl management tool.

Procedure

1. When OperatorHub is used to deploy CodeReady Workspaces then set the following values in codeready-workspaces Custom Resource (CR):

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

2. When **crwctl** tool is used to deploy CodeReady Workspaces then use **--che-operator-cr-patch-yaml** flag:

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

patch.yaml must contain the following:

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

9.4.3. 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}]'
```

9.5. REMOVING USER DATA

9.5.1. 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 who is configured in the **CHE_SYSTEM_ADMIN__NAME** property (the default is **admin**). The system permissions are granted when the CodeReady Workspaces server starts. If the user is not present in the CodeReady Workspaces user database, it happens after the first user's login.

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 Uninstalling CodeReady Workspaces.