Red Hat CodeReady Workspaces 2.2

Administration Guide

Administering Red Hat CodeReady Workspaces 2.2

Last Updated: 2020-11-19

Red Hat CodeReady Workspaces 2.2 Administration Guide

Administering Red Hat CodeReady Workspaces 2.2

Supriya Takkhi

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

CHAPTER 1. CUSTOMIZING THE DEVFILE AND PLUG-IN REGISTRIES

CodeReady Workspaces 2.2 introduces two registries: the plug-ins registry and the devfile registry. They are static websites where the metadata of CodeReady Workspaces plug-ins and CodeReady Workspaces devfiles is published.

The plug-in registry makes it possible to share a plug-in definition across all the users of the same instance of CodeReady Workspaces. Only plug-ins that are published in a registry can be used in a devfile.

The devfile registry holds the definitions of the CodeReady Workspaces stacks. These 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.

The devfile and plug-in registries run in two separate pods and are deployed when the CodeReady Workspaces server is deployed (that is the default behavior of the CodeReady Workspaces Operator). The metadata of the plug-ins and devfiles are versioned on GitHub and follow the CodeReady Workspaces server life cycle.

In this document, the following two ways to customize the default registries that are deployed with CodeReady Workspaces (to modify the plug-ins or devfile metadata) are described:

1. Building a custom image of the registries
2. Running the default images but modifying them at runtime
   - Building and running a custom registry image
   - Including the plug-in binaries in the registry image
   - Editing a devfile and plug-in at runtime

1.1. BUILDING AND RUNNING A CUSTOM REGISTRY IMAGE

This section describes the building of registries and updating a running CodeReady Workspaces server to point to the registries.

1.1.1. Building a custom devfile registry

This section describes how to build a custom devfiles registry. Following operations are covered:

- Getting a copy of the source code necessary to build a devfiles registry.
- Adding a new devfile.
- Building the devfiles registry.

Procedure

1. Clone the devfile registry repository:

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces/dependencies/che-devfile-registry
```

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

   File organization for a devfile

   ```
   ./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 Making a workspace portable using a Devfile .

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

   Table 1.1. 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 usually include the tools included in the stack.

   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 the containers for the custom devfile registry:

   ```
   $ docker build -t my-devfile-registry .
   ```

1.1.2. Building a custom plug-in registry

This section describes how to build a custom plug-in registry. Following operations are covered:

- Getting a copy of the source code necessary to build a custom plug-in registry.
- Adding a new plug-in.

- Building the custom plug-in registry.

Procedure

1. Clone the plug-in registry repository:

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces/dependencies/che-plugin-registry
```

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

File organization for a plugin

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

3. Add valid content to the **meta.yaml** file. See the "Using a Visual Studio Code extension in CodeReady Workspaces" section or the README.md file in the **eclipse/che-plugin-registry** repository for a detailed description of the **meta.yaml** file format.

4. Create a file named **latest.txt** with content the name of the latest ***<plugin-version>*** directory.

EXAMPLE

```
$ 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 the containers for the custom plug-in registry:

```
$ docker build -t my-devfile-registry .
```

1.1.3. Deploying the 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.

This is true on Minikube, for example, if before running the **docker build** commands, the user executed the **eval $\{minikube docker-env}** command (or, the **eval $\{minishift docker-env}** command for Minishift).

Otherwise, these images can be pushed to a container registry (public, such as **quay.io**, or the DockerHub, or a private registry).

1.1.3.1. Deploying registries in OpenShift

Procedure

An OpenShift template to deploy the plug-in registry is available in the **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="IfNotPresent"
```

(1) If installed using crwctl, the default CodeReady Workspaces project is **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="IfNotPresent"
```

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

3. Check if the registries are deployed successfully on OpenShift.

   a. To verify that the new plug-in is correctly published to the plug-in registry, make a request to the registry path **/v3/plugins/index.json** (or **/devfiles/index.json** for the devfile registry).

```
$ URL=$(oc get -o 'custom-columns=URL:.spec.rules[0].host' \
  -l app=che-plugin-registry route --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | grep -A 4 -B 5 "\"name\":\"my-plug-in\""
,\{
 "id": "my-org/my-plug-in/1.0.0",
 "displayName":"This is my first plug-in for CodeReady Workspaces",
 "version":"1.0.0",
 "type":"VS Code extension",
 "name":"my-plug-in",
 "description":"This plugin shows that we are able to add plugins to the registry",
 "publisher":"my-org",
 "links": \{"self":"/v3/plugins/my-org/my-plug-in/1.0.0" }
}
--
--
,\{
 "id": "my-org/my-plug-in/latest",
 "displayName":"This is my first plug-in for CodeReady Workspaces",
 "version":"latest",
 "type":"VS Code extension",
 "name":"my-plug-in",
 "description":"This plugin shows that we are able to add plugins to the registry",
 "publisher":"my-org",
 "links": \{"self":"/v3/plugins/my-org/my-plug-in/latest" }
}
```

b. Verify that the CodeReady Workspaces server points to the URL of the registry. To do this, compare the value of the **CHE_WORKSPACE_PLUGIN__REGISTRY__URL** parameter in the **codeready** ConfigMap (or **CHE_WORKSPACE_DEVFILE__REGISTRY__URL** for the devfile registry):

```
$ oc get \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_PLUGIN_REGISTRY_URL']" \
  --no-headers cm/che
URL
http://che-plugin-registry-che.192.168.99.100.mycluster.mycompany.com/v3
```

with the URL of the route:

```
$ oc get -o 'custom-columns=URL:.spec.rules[0].host' \
  -l app=che-plugin-registry route --no-headers
che-plugin-registry-che.192.168.99.100.mycluster.mycompany.com
```

c. If they do not match, update the ConfigMap and restart the CodeReady Workspaces server.

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

When the new registries are deployed and the CodeReady Workspaces server is configured to use them, the new plug-ins are available in the **Plugin** view of a workspace and the new stacks are displayed in the **New Workspace** tab of the user dashboard.

1.2. INCLUDING THE PLUG-IN BINARIES IN THE REGISTRY IMAGE

The plug-in registry only hosts CodeReady Workspaces plug-in metadata. The binaries are usually referred through a link in the **meta.yaml** file. Sometimes, such as offline environments, it may be necessary to make the binaries available inside the registry image.

This section describes how to modify a plug-in **meta.yaml** file to point to a local file inside the container and rebuild a new registry that contains the modified plug-in **meta.yaml** file and the binary file. In the following example, the Java plug-in that refers to two remote VS Code extensions binaries is considered.

Prerequisites

- CodeReady Workspaces is installed.
- The OpenShift command-line tool, **oc**, is installed.

Procedure

1. Download the binaries locally:

```
$ ORG=redhat
$ NAME=java11
$ VERSION=latest
$ URL_VS_CODE_EXT1="https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-
0.19.0.vsix[_https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-0.19.0.vsix_]"
$ URL_VS_CODE_EXT2="https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-
0.46.0-1549.vsix[_https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-0.46.0-
1549.vsix_]"
$ VS_CODE_EXT1=https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-0.19.0.vsix[_vscode-java-debug-
0.19.0.vsix_]
$ VS_CODE_EXT2=https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-0.46.0-
1549.vsix[_java-0.46.0-1549.vsix_]
$ curl -sSL -o ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/$\
{VS_CODE_EXT1} \
  $\{URL_VS_CODE_EXT1}
$ curl -sSL -o ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/$\
{VS_CODE_EXT2} \
  $\{URL_VS_CODE_EXT2}
```

2. Get the plug-in registry URL:

   - For an Operator installation:

```
$ oc get checluster ${CHECLUSTER_NAME} \
  -o jsonpath='{.status.pluginRegistryURL}' -n ${CODEREADY_NAMESPACE}
```

     Note that the obtained URL is without the **http** or **https** prefix.

3. Update the URLs in the **meta.yaml** file, so that they point to the VS Code extension binaries that are saved in the registry container:

```
$ NEW_URL_VS_CODE_EXT1=http://$\{PLUGIN_REG_URL}/v3/plugins/$\{ORG}/$\
{NAME}/$\{VERSION}/$\{VS_CODE_EXT1}
$ NEW_URL_VS_CODE_EXT2=http://$\{PLUGIN_REG_URL}/v3/plugins/$\{ORG}/$\
{NAME}/$\{VERSION}/$\{VS_CODE_EXT2}
$ sed -i -e 's/$\{URL_PLUGIN1}/$\{NEW_URL_VS_CODE_EXT1}/g' \
  ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/meta.yaml
$ sed -i -e 's/$\{URL_PLUGIN2}/$\{NEW_URL_VS_CODE_EXT2}/g' \
  ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/meta.yaml
```

4. Build and deploy the plug-in registry using the instructions in the Building and running a custom registry image section.

1.3. EDITING A DEVFILE AND PLUG-IN AT RUNTIME

An alternative to building a custom registry image is to:

1. Start a registry
2. Modify its content at runtime

This approach is simpler and faster. But the modifications are lost as soon as the container is deleted.

1.3.1. Adding a plug-in at runtime

Procedure

To add a plug-in:

1. Check out the plugin registry sources.

```
$ git clone https://github.com/eclipse/che-plugin-registry; \
  cd che-plugin-registry
```

2. Create a **meta.yaml** in some local folder. This can be done from scratch or by copying from an existing plug-in's **meta.yaml** file.

```
$ PLUGIN="v3/plugins/new-org/new-plugin/0.0.1"; \
  mkdir -p ${PLUGIN}; cp v3/plugins/che-incubator/cpptools/0.1/* ${PLUGIN}/
  echo "${PLUGIN##*/}" > ${PLUGIN}/../latest.txt
```

3. If copying from an existing plug-in, make changes to the **meta.yaml** file to suit your needs. Make sure your new plug-in has a unique **title**, **displayName** and **description**. Update the **firstPublicationDate** to today's date.

4. These fields in **meta.yaml** must match the path defined in **PLUGIN** above.

```
publisher: new-org
name: new-plugin
version: 0.0.1
```

5. Get the name of the Pod that hosts the plug-in registry container. To do this, filter the **component=plugin-registry** label:

```
$ PLUGIN_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
  --no-headers pod -l component=plugin-registry)
```

6. Regenerate the registry's **index.json** file to include your new plug-in.

```
$ cd che-plugin-registry; \
    "$(pwd)/build/scripts/generate_latest_metas.sh" v3 && \
    "$(pwd)/build/scripts/check_plugins_location.sh" v3 && \
    "$(pwd)/build/scripts/set_plugin_dates.sh" v3 && \
    "$(pwd)/build/scripts/check_plugins_viewer_mandatory_fields.sh" v3 && \
    "$(pwd)/build/scripts/index.sh" v3 > v3/plugins/index.json
```

7. Copy the new **index.json** and **meta.yaml** files from your new local plug-in folder to the container.

```
$ cd che-plugin-registry; \
  LOCAL_FILES="$(pwd)/${PLUGIN}/meta.yaml $(pwd)/v3/plugins/index.json"; \
  oc exec ${PLUGIN_REG_POD} -i -t -- mkdir -p /var/www/html/${PLUGIN}; \
  for f in $LOCAL_FILES; do e=${f/$(pwd)\/}; echo "Upload ${f} -> /var/www/html/${e}"; \
    oc cp "${f}" ${PLUGIN_REG_POD}:/var/www/html/${e}; done
```

8. The new plug-in can now be used from the existing CodeReady Workspaces instance's plug-in registry. To discover it, go to the CodeReady Workspaces dashboard, then click the **Workspaces** link. From there, click the gear icon to configure one of your workspaces. Select the **Plugins** tab to see the updated list of available plug-ins.

1.3.2. Adding a devfile at runtime

Procedure

To add a devfile:

1. Check out the devfile registry sources.

```
$ git clone https://github.com/eclipse/che-devfile-registry; \
  cd che-devfile-registry
```

2. Create a **devfile.yaml** and **meta.yaml** in some local folder. This can be done from scratch or by copying from an existing devfile.

```
$ STACK="new-stack"; \
  mkdir -p devfiles/${STACK}; cp devfiles/nodejs/* devfiles/${STACK}/
```

3. If copying from an existing devfile, make changes to the devfile to suit your needs. Make sure your new devfile has a unique **displayName** and **description**.

4. Get the name of the Pod that hosts the devfile registry container. To do this, filter the **component=devfile-registry** label:

```
$ DEVFILE_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
  --no-headers pod -l component=devfile-registry)
```

5. Regenerate the registry's **index.json** file to include your new devfile.

```
$ cd che-devfile-registry; \
  "$(pwd)/build/scripts/check_mandatory_fields.sh" devfiles; \
  "$(pwd)/build/scripts/index.sh" > index.json
```

6. Copy the new **index.json**, **devfile.yaml** and **meta.yaml** files from your new local devfile folder to the container.

```
$ cd che-devfile-registry; \
  LOCAL_FILES="$(pwd)/${STACK}/meta.yaml $(pwd)/${STACK}/devfile.yaml
$(pwd)/index.json"; \
  oc exec ${DEVFILE_REG_POD} -i -t -- mkdir -p /var/www/html/devfiles/${STACK}; \
  for f in $LOCAL_FILES; do e=${f/$(pwd)\/}; echo "Upload ${f} ->
/var/www/html/devfiles/${e}"
    oc cp "${f}" ${DEVFILE_REG_POD}:/var/www/html/devfiles/${e}; done
```

7. The new devfile can now be used from the existing CodeReady Workspaces instance's devfile registry. To discover it, go to the CodeReady Workspaces dashboard, then click the **Workspaces** link. From there, click **Add Workspace** to see the updated list of available devfiles.

CHAPTER 2. RETRIEVING CODEREADY WORKSPACES LOGS

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

- Viewing OpenShift events
- Viewing CodeReady Workspaces server logs
- Viewing external service logs
- Viewing CodeReady Workspaces workspaces logs
- Viewing the plug-in broker logs
- Collecting Red Hat CodeReady Workspaces logs using crwctl

2.1. 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 .

2.2. 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 2.1. 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 2.2. Logs of the Operator:

2.3. VIEWING CODEREADY WORKSPACES SERVER LOGS

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

2.3.1. 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
```

2.4. VIEWING EXTERNAL SERVICE LOGS

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

2.4.1. 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.

2.4.1.1. 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.
3. In the **Deployment Configs** section, click the **keycloak** link to open it.
4. In the **History** tab, click the **View log** link for the active RH-SSO deployment.
5. The RH-SSO logs are displayed.

Additional resources

- See the active CodeReady Workspaces deployment log for diagnostics and error messages related to the RH-SSO IDE Server.

2.4.1.2. 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**.

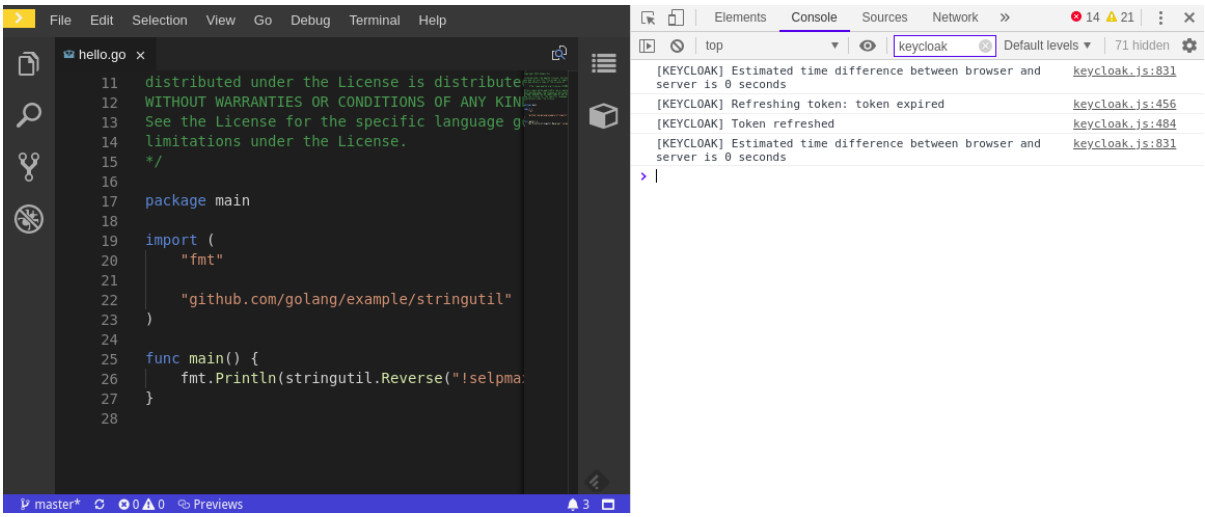
2.4.1.3. Viewing the RH-SSO client logs on Google Chrome

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

Procedure

1. Click on **Menu** > **More Tools** > **Developer Tools**.

2. Click on the **Console** tab.



2.4.2. 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 **Enter**
     The OpenShift Web Console now searches 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 Viewing the CodeReady Workspaces server logs section.

2.5. VIEWING CODEREADY WORKSPACES WORKSPACES LOGS

This section describes how to view CodeReady Workspaces workspaces logs.

2.5.1. Viewing Che-Theia IDE logs

This section describes how to view Che-Theia IDE logs.

2.5.1.1. Viewing Che-Theia editor logs using the OpenShift CLI

Observing Che-Theia editor logs helps to get a better understanding and insight over the plug-ins loaded by the editor. This section describes how to access the Che-Theia editor logs using the OpenShift CLI (command-line interface).

Prerequisites

- CodeReady Workspaces is deployed in an OpenShift cluster.
- A workspace is created.
- User is located in a CodeReady Workspaces installation project.

Procedure

1. Obtain the list of the available Pods:

```
$ oc get pods
```

Example

```
$ oc get pods
NAME                                          READY  STATUS   RESTARTS  AGE
codeready-9-xz6g8                               1/1    Running  1         15h
workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2  4/4    Running  0         1h
```

2. Obtain the list of the available containers in the particular Pod:

```
$ oc get pods <name-of-pod> --output jsonpath='\{.spec.containers[*].name}'
```

Example:

```
$ oc get pods workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -o
jsonpath='\{.spec.containers[*].name}'
> go-cli che-machine-exechr7 theia-idexzb vscode-gox3r
```

3. Get logs from the **theia**/**ide** container:

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

Example:

```
$ oc logs --follow workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -container
theia-idexzb
>root INFO unzipping the plug-in 'task_plugin.theia' to directory: /tmp/theia-
unpacked/task_plugin.theia
root INFO unzipping the plug-in 'theia_yeoman_plugin.theia' to directory: /tmp/theia-
unpacked/theia_yeoman_plugin.theia
root WARN A handler with prefix term  is already registered.
root INFO [nsfw-watcher: 75] Started watching: /home/theia/.theia
root WARN e.onStart is slow, took: 367.4600000013015 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [nsfw-watcher: 75] Started watching: /projects/.theia/tasks.json
```

```
root INFO [4f9590c5-e1c5-40d1-b9f8-ec31ec3bdac5] Sync of 9 plugins took:
62.26000000242493 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [hosted-plugin: 88] PLUGIN_HOST(88) starting instance
```

2.5.2. Viewing logs from language servers and debug adapters

2.5.2.1. Checking important logs

This section describes how to check important logs.

Procedure

1. In the OpenShift web console, click **Applications → Pods** to see a list of all the active workspaces.
2. Click on the name of the running Pod where the workspace is running. The Pod screen contains the list of all containers with additional information.
3. Choose a container and click the container name.

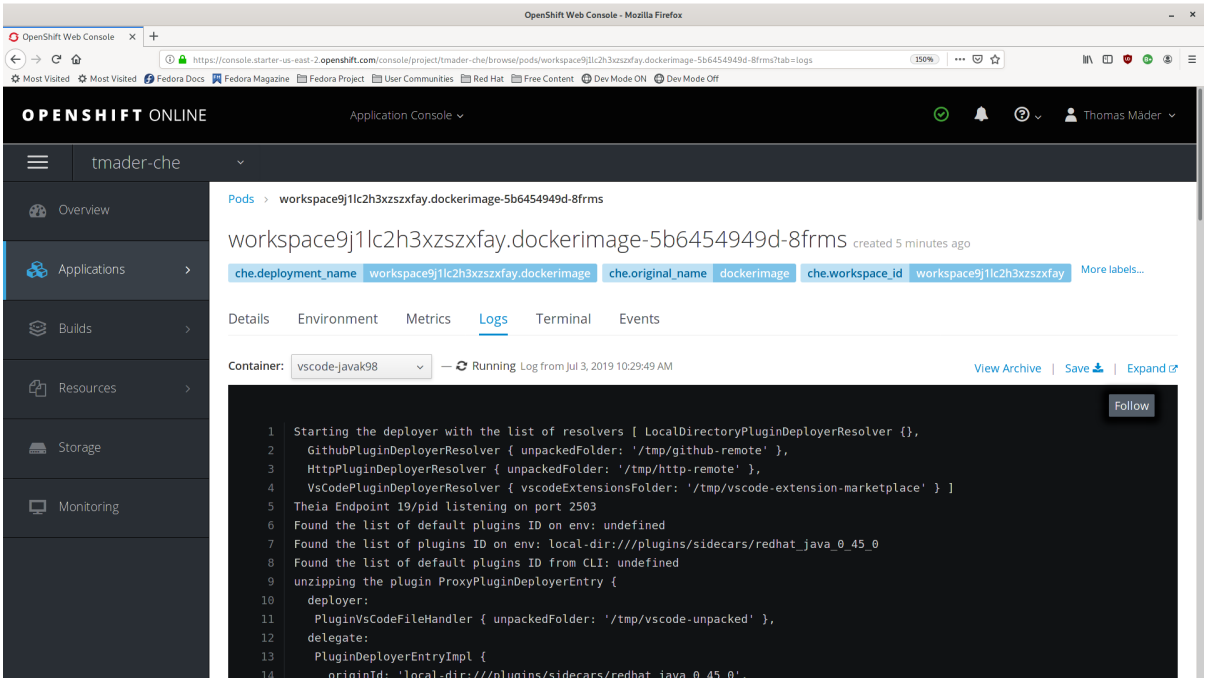
   TIP

   The most important logs are the **theia-ide** container and the plug-ins container logs.

4. On the container screen, navigate to the **Logs** section.

   EXAMPLE

   The following is an output log of the sidecar container running the Java plug-in.



2.5.2.2. Detecting memory problems

This section describes how to detect memory problems related to a plug-in running out of memory. The following are the two most common problems related to a plug-in running out of memory:

The plug-in container runs out of memory

This can happen during plug-in initialization when the container does not have enough RAM to execute the entrypoint of the image. The user can detect this in the logs of the plug-in container. In this case, the logs contain **OOMKilled**, which implies that the processes in the container requested more memory than is available in the container.

A process inside the container runs out of memory without the container noticing this

For example, the Java language server (Eclipse JDT Language Server, started by the vscode-java extension) throws an **OutOfMemoryException**. This can happen any time after the container is initialized, for example, when a plug-in starts a language server or when a process runs out of memory because of the size of the project it has to handle.

To detect this problem, check the logs of the primary process running in the container. For example, to check the log file of Eclipse JDT Language Server for details, see the relevant plug-in-specific sections.

2.5.2.3. Logging the client-server traffic for debug adapters

This section describes how to log the exchange between Che-Theia and a debug adapter into the **Output** view.

Prerequisites

- A debug session must be started for the **Debug adapters** option to appear in the list.

Procedure

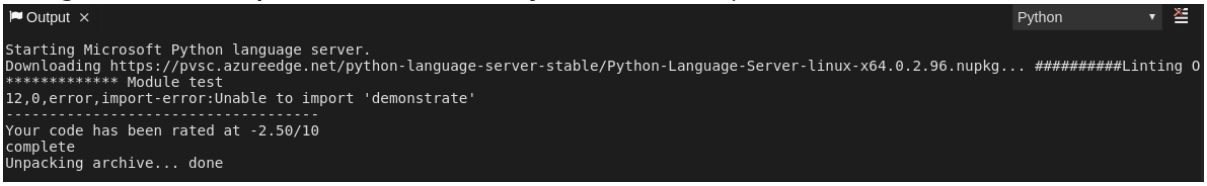
1. Click **File → Settings** and then **open Preferences**.
2. Expand the **Debug** section in the **Preferences** view.
3. Set the **trace** preference value to **true** (default is **false**).
4. All the communication events are now logged.
5. To watch these events, click **View → Output** and select **Debug adapters** from the drop-down list at the upper right corner of the **Output** view.

2.5.2.4. Viewing logs for Python

This section describes how to view logs for the Python language server.

Procedure

- Navigate to the **Output** view and select **Python** in the drop-down list.



2.5.2.5. Viewing logs for Go

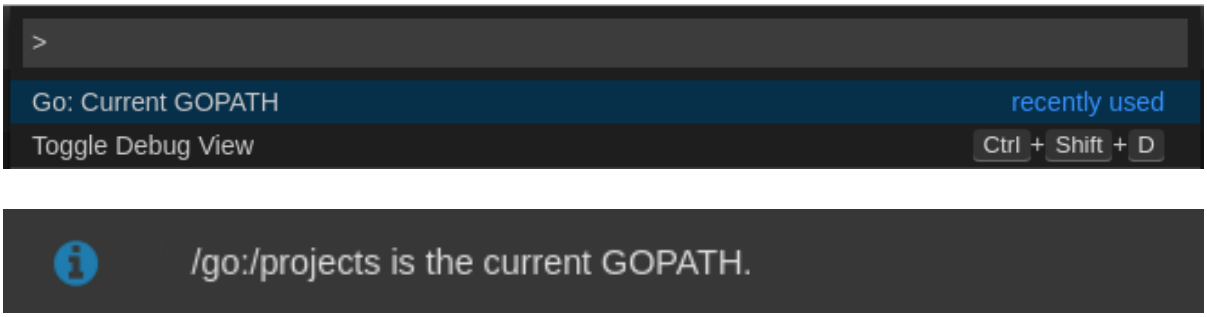
This section describes how to view logs for the Go language server.

2.5.2.5.1. Finding the gopath

This section describes how to find where the **GOPATH** variable points to.

Procedure

- Execute the **Go: Current GOPATH** command.



2.5.2.5.2. Viewing the Debug Console log for Go

This section describes how to view the log output from the Go debugger.

Procedure

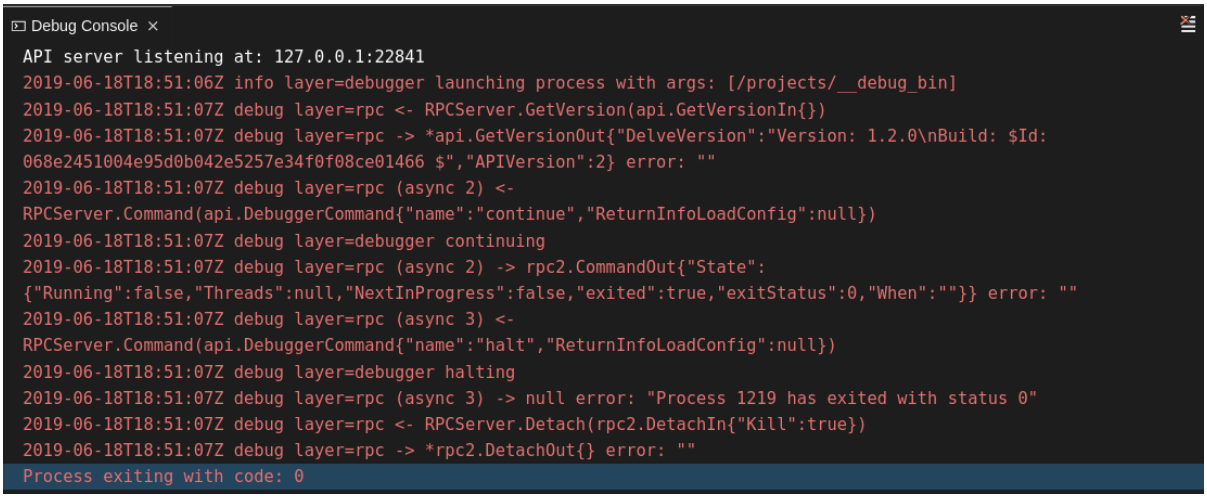
1. Set the **showLog** attribute to **true** in the debug configuration.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true
      ....
    }
  ]
}
```

2. To enable debugging output for a component, add the package to the comma-separated list value of the **logOutput** attribute:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true,
      "logOutput": "debugger,rpc,gdbwire,lldbout,debuglineerr"
      ....
    }
  ]
}
```

3. The debug console prints the additional information in the debug console.



2.5.2.5.3. Viewing the Go logs output in the Output panel

This section describes how to view the Go logs output in the **Output** panel.

Procedure

1. Navigate to the **Output** view.
2. Select **Go** in the drop-down list.

2.5.2.6. Viewing logs for the NodeDebug NodeDebug2 adapter

NOTE

There are no specific diagnostics other than the general ones.

2.5.2.7. Viewing logs for Typescript

2.5.2.7.1. Enabling the label switched protocol (LSP) tracing

Procedure

1. To enable the tracing of messages sent to the Typescript (TS) server, in the **Preferences** view, set the **typescript.tsserver.trace** attribute to **verbose**. Use this to diagnose the TS server issues.
2. To enable logging of the TS server to a file, set the **typescript.tsserver.log** attribute to **verbose**. Use this log to diagnose the TS server issues. The log contains the file paths.

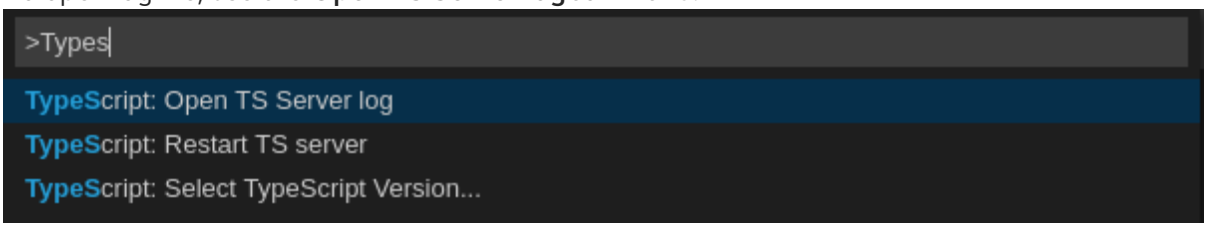
2.5.2.7.2. Viewing the Typescript language server log

This section describes how to view the Typescript language server log.

Procedure

1. To get the path to the log file, see the Typescript **Output** console:

2. To open log file, use the **Open TS Server log**command.

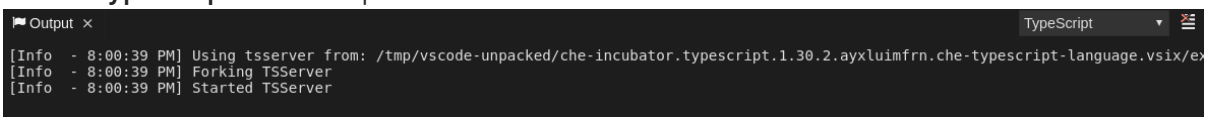


2.5.2.7.3. Viewing the Typescript logs output in the Output panel

This section describes how to view the Typescript logs output in the **Output** panel.

Procedure

1. Navigate to the **Output** view
2. Select **TypeScript** in the drop-down list.



2.5.2.8. Viewing logs for Java

Other than the general diagnostics, there are Language Support for Java (Eclipse JDT Language Server) plug-in actions that the user can perform.

2.5.2.8.1. Verifying the state of the Eclipse JDT Language Server

Procedure

Check if the container that is running the Eclipse JDT Language Server plug-in is running the Eclipse JDT Language Server main process.

1. Open a terminal in the container that is running the Eclipse JDT Language Server plug-in (an example name for the container: **vscode-javaxxx**).
2. Inside the terminal, run the **ps aux | grep jdt** command to check if the Eclipse JDT Language Server process is running in the container. If the process is running, the output is:

```
usr/lib/jvm/default-jvm/bin/java --add-modules=ALL-SYSTEM --add-opens java.base/java.util
```

   This message also shows the VSCode Java extension used. If it is not running, the language server has not been started inside the container.

3. Check all logs described in Checking important logs

2.5.2.8.2. Verifying the Eclipse JDT Language Server features

Procedure

If the Eclipse JDT Language Server process is running, check if the language server features are working:

1. Open a Java file and use the hover or autocomplete functionality. In case of an erroneous file, the user sees Java in the **Outline** view or in the **Problems** view.

2.5.2.8.3. Viewing the Java language server log

Procedure

The Eclipse JDT Language Server has its own workspace where it logs errors, information about executed commands, and events.

1. To open this log file, open a terminal in the container that is running the Eclipse JDT Language Server plug-in. You can also view the log file by running the **Java: Open Java Language Server log file** command.

2. Run **cat *<PATH_TO_LOG_FILE>*** where **PATH_TO_LOG_FILE** is **/home/theia/.theia/workspace-storage/*<workspace_name>*/redhat.java/jdt_ws/.metadata/.log**.

2.5.2.8.4. Logging the Java language server protocol (LSP) messages

Procedure

To log the LSP messages to the VS Code **Output** view, enable tracing by setting the **java.trace.server** attribute to **verbose**.

Additional resources

For troubleshooting instructions, see the VS Code Java Github repository .

2.5.2.9. Viewing logs for Intelephense

2.5.2.9.1. Logging the Intelephense client-server communication

Procedure

To configure the PHP Intelephense language support to log the client-server interexchange in the **Output** view:

1. Click **File → Settings**.

2. Open the **Preferences** view.

3. Expand the **Intelephense** section and set the **trace.server.verbose** preference value to **verbose** to see all the communication events (the default value is **off**).

2.5.2.9.2. Viewing Intelephense events in the Output panel

This procedure describes how to view Intelephense events in the **Output** panel.

Procedure

1. Click **View → Output**
2. Select **Intelephense** in the drop-down list for the **Output** view.

2.5.2.10. Viewing logs for PHP-Debug

This procedure describes how to configure the PHP Debug plug-in to log the PHP Debug plug-in diagnostic messages into the **Debug Console** view. Configure this before the start of the debug session.

Procedure

1. In the **launch.json** file, add the **"log": true** attribute to the selected launch configuration.
2. Start the debug session.
3. The diagnostic messages are printed into the **Debug Console** view along with the application output.

2.5.2.11. Viewing logs for XML

Other than the general diagnostics, there are XML plug-in specific actions that the user can perform.

2.5.2.11.1. Verifying the state of the XML language server

Procedure

1. Open a terminal in the container named **vscode-xml-*<xxx>***.
2. Run **ps aux | grep java** to verify that the XML language server has started. If the process is running, the output is:

   ```
   java ***/org.eclipse.ls4xml-uber.jar`
   ```

   If is not, see the Checking important logs chapter.

2.5.2.11.2. Checking XML language server feature flags

Procedure

1. Check if the features are enabled. The XML plug-in provides multiple settings that can enable and disable features:
   - **xml.format.enabled**: Enable the formatter
   - **xml.validation.enabled**: Enable the validation
   - **xml.documentSymbols.enabled**: Enable the document symbols
2. To diagnose whether the XML language server is working, create a simple XML element, such as **<hello></hello>**, and confirm that it appears in the **Outline** panel on the right.
3. If the document symbols do not show, ensure that the **xml.documentSymbols.enabled** attribute is set to **true**. If it is **true**, and there are no symbols, the language server may not be hooked to the editor. If there are document symbols, then the language server is connected to

the editor.

4. Ensure that the features that the user needs, are set to **true** in the settings (they are set to **true** by default). If any of the features are not working, or not working as expected, file an issue against the Language Server.

2.5.2.11.3. Enabling XML Language Server Protocol (LSP) tracing

Procedure

To log LSP messages to the VS Code **Output** view, enable tracing by setting the **xml.trace.server** attribute to **verbose**.

2.5.2.11.4. Viewing the XML language server log

Procedure

The log from the language server can be found in the plug-in sidecar at **/home/theia/.theia/workspace-storage/<workspace_name>/redhat.vscode-xml/lsp4xml.log**.

2.5.2.12. Viewing logs for YAML

This section describes the YAML plug-in specific actions that the user can perform, in addition to the general diagnostics ones.

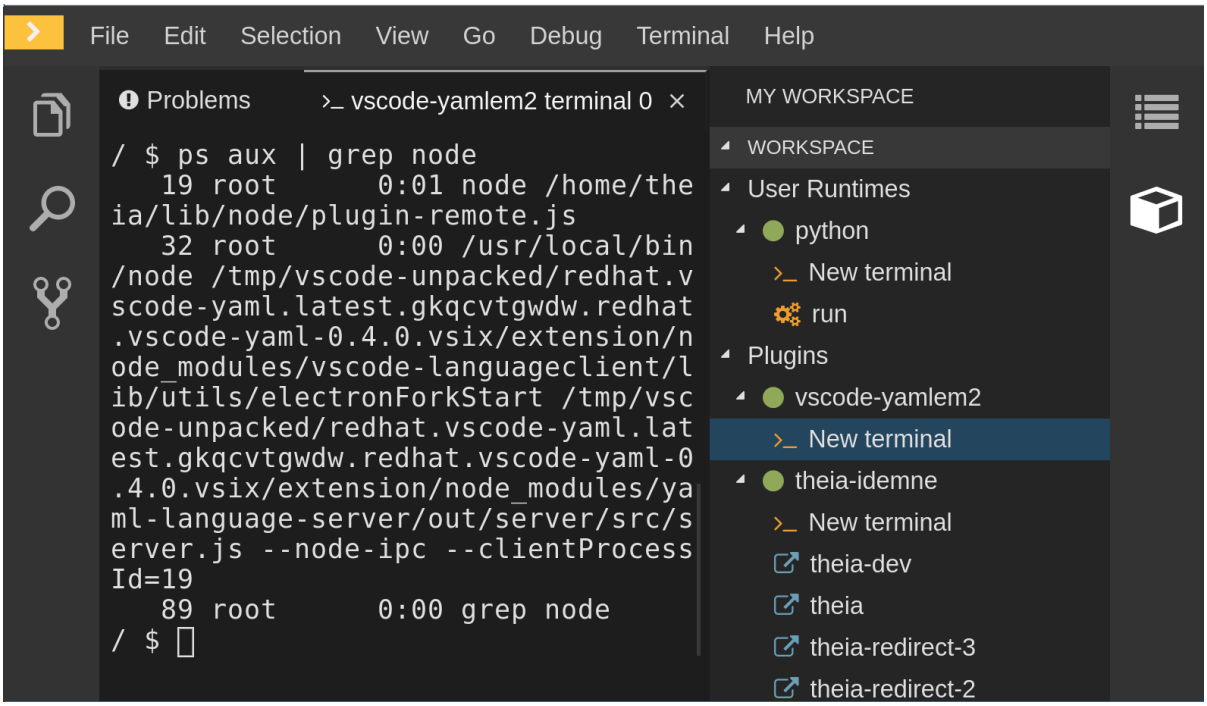
2.5.2.12.1. Verifying the state of the YAML language server

This section describes how to verify the state of the YAML language server.

Procedure

Check if the container running the YAML plug-in is running the YAML language server.

1. In the editor, open a terminal in the container that is running the YAML plug-in (an example name of the container: **vscode-yaml-*<xxx>***).
2. In the terminal, run the **ps aux | grep node** command. This command searches all the node processes running in the current container.
3. Verify that a command **node **/server.js is running.**



The **node **/server.js** running in the container indicates that the language server is running. If it is not running, the language server has not started inside the container. In this case, see Checking important logs.

2.5.2.12.2. Checking the YAML language server feature flags

Procedure

To check the feature flags:

1. Check if the features are enabled. The YAML plug-in provides multiple settings that can enable and disable features, such as:
   - **yaml.format.enable**: Enables the formatter
   - **yaml.validate**: Enables validation
   - **yaml.hover**: Enables the hover function
   - **yaml.completion**: Enables the completion function
2. To check if the plug-in is working, type the simplest YAML, such as **hello: world**, and then open the Outline panel on the right side of the editor.
3. Verify if there are any document symbols. If yes, the language server is connected to the editor.
4. If any feature is not working, make sure that the settings listed above are set to **true** (they are set to **true** by default). If a feature is not working, file an issue against the Language Server.

2.5.2.12.3. Enabling YAML Language Server Protocol (LSP) tracing

Procedure

To log LSP messages to the VS Code Output view, enable tracing by setting the **yaml.trace.server** attribute to **verbose**.

2.5.2.13. Viewing logs for Dotnet with Omnisharp-Theia plug-in

2.5.2.13.1. Omnisharp-Theia plug-in

CodeReady Workspaces uses the Omnisharp-Theia plug-in as a remote plug-in. It is located at github.com/redhat-developer/omnisharp-theia-plugin. In case of an issue, report it, or contribute your fix in the repository.

This plug-in registers omnisharp-roslyn as a language server and provides project dependencies and language syntax for C# applications.

The language server runs on .NET SDK 2.2.105.

2.5.2.13.2. Verifying the state of the Omnisharp-Theia plug-in language server

Procedure

To check if the container running the Omnisharp-Theia plug-in is running OmniSharp, execute the **ps aux | grep OmniSharp.exe** command. If the process is running, the following is an example output:

```
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/bin/mono
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/omnisharp/OmniSharp.exe
```

If the output is different, the language server has not started inside the container. Check the logs described in Checking important logs.

2.5.2.13.3. Checking Omnisharp Che-Theia plug-in language server features

Procedure

- If the OmniSharp.exe process is running, check if the language server features are working by opening a **.cs** file and trying the hover or completion features, or opening the **Problems** or Outline view.

2.5.2.13.4. Viewing Omnisharp-Theia plug-in logs in the Output panel

Procedure

If **Omnisharp.exe** is running, it logs all information in the **Output** panel. To view the logs, open the Output view and select **C#** from the drop-down list.

2.5.2.14. Viewing logs for Dotnet with NetcoredebugOutput plug-in

2.5.2.14.1. NetcoredebugOutput plug-in

The NetcoredebugOutput plug-in provides netcoredbg, which implements the VS Code Debug Adapter protocol and allows users to debug .NET applications under the .NET Core runtime.

Dotnet SDK v.2.2.105 is installed in the container where the Netcoredbg plug-in is running.

2.5.2.14.2. Verifying the state of the NetcoredebugOutput plug-in

Procedure

To test the plug-in initialization:

1. Check if there is a netcoredbg debug configuration in the **launch.json** file. The following is an example debug configuration:

```
{
 "type": "netcoredbg",
 "request": "launch",
 "program": "$\{workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>",
 "args": [],
 "name": ".NET Core Launch (console)",
 "stopAtEntry": false,
 "console": "internalConsole"
}
```

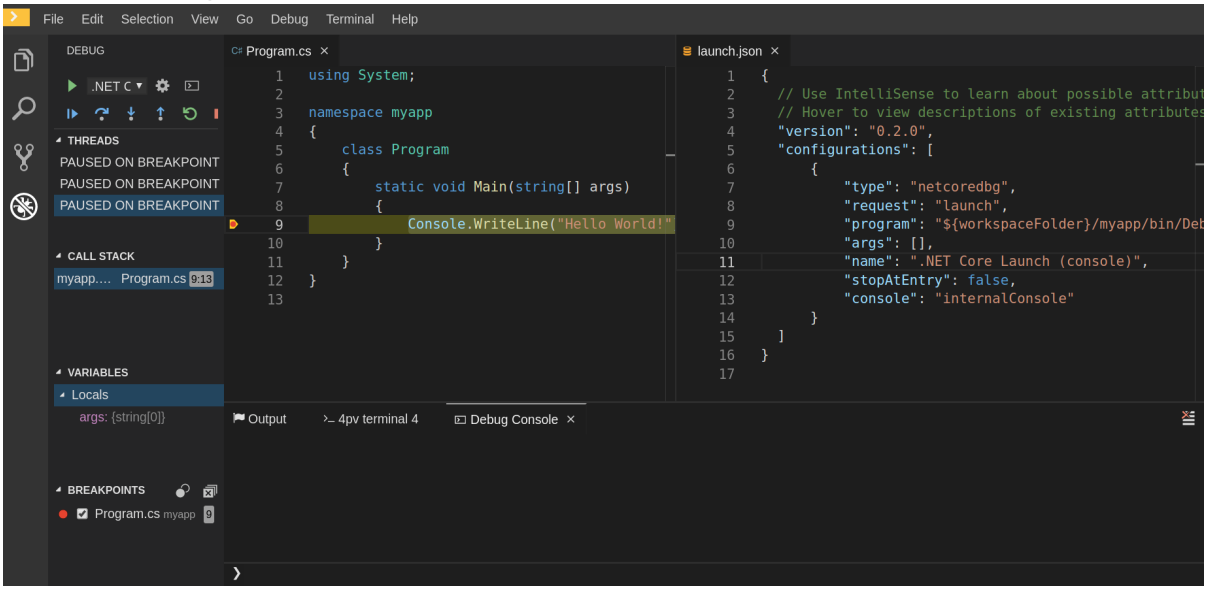
2. To test if it exists, test the autocompletion feature within the braces of the **configuration** section of the **launch.json** file. If you can find**netcoredbg**, the Che-Theia plug-in is correctly initialized. If not, see Checking important logs.

2.5.2.14.3. Viewing NetcoredebugOutput plug-in logs in the Output panel

This section describes how to view NetcoredebugOutput plug-in logs in the Output panel.

Procedure

- Open the Debug console.



2.5.2.15. Viewing logs for Camel

2.5.2.15.1. Verifying the state of the Camel language server

Procedure

The user can inspect the log output of the sidecar container using the Camel language tools that are stored in the **vscode-apache-camel<*xxx*>** Camel container.

To verify the state of the language server:

1. Open a terminal inside the **vscode-apache-camel<*xxx*>** container.
2. Run the **ps aux | grep java** command. The following is an example language server process:

```
java -jar /tmp/vscode-unpacked/camel-tooling.vscode-apache-
camel.latest.euqhbmepxd.camel-tooling.vscode-apache-camel-
0.0.14.vsix/extension/jars/language-server.jar
```

3. If you cannot find it, see Checking important logs.

2.5.2.15.2. Viewing Camel logs in the Output panel

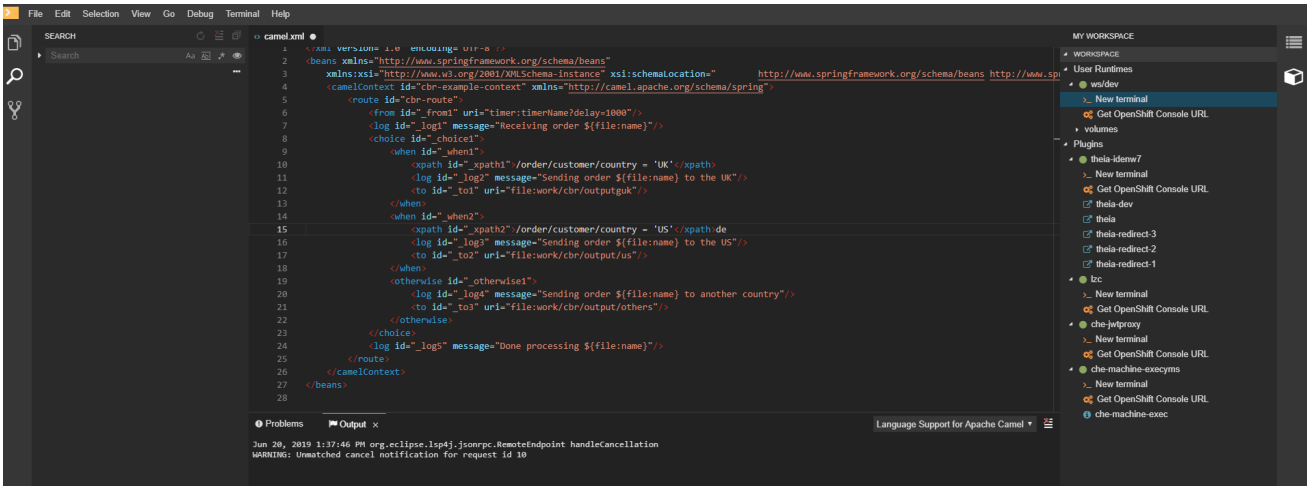
The Camel language server is a SpringBoot application that writes its log to the **$\{java.io.tmpdir}/log-camel-lsp.out** file. Typically,**$\{java.io.tmpdir}** points to the/**tmp** directory, so the filename is /**tmp/log-camel-lsp.out**.

Procedure

The Camel language server logs are printed in the Output channel named Language Support for Apache Camel.

NOTE

The output channel is created only at the first created log entry on the client side. It may be absent when everything is going well.



2.6. 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

2.7. 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:start** 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 running workspace logs
- **crwctl workspace:logs --follow** starts collecting logs of all new workspaces

CHAPTER 3. 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.

3.1. 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
```

3.2. COLLECTING CODEREADY WORKSPACES METRICS WITH PROMETHEUS

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

Prerequisites

- CodeReady Workspaces is exposing metrics on port **8087**. See Enabling and exposing codeready-workspaces metrics.
- Prometheus 2.9.1 or 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:

  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-host**, 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.

3.3. EXTENDING CODEREADY WORKSPACES MONITORING METRICS

This section describes how to create a metric or a group of metrics to extend the monitoring metrics that CodeReady Workspaces is exposing.

There are two major modules for metrics:

- **che-core-metrics-core** – contains core metrics module
- **che-core-api-metrics** – contains metrics that are dependent on core CodeReady Workspaces components, such as workspace or user managers

Procedure

- Create a class that extends the **MeterBinder** class. This allows to register the created metric in the overridden **bindTo(MeterRegistry registry)** method.
  The following is an example of a metric that has a function that supplies the value for it:

  Example metric

```
public class UserMeterBinder implements MeterBinder {

  private final UserManager userManager;

  @Inject
  public UserMeterBinder(UserManager userManager) {
    this.userManager = userManager;
  }

  @Override
  public void bindTo(MeterRegistry registry) {
    Gauge.builder("che.user.total", this::count)
        .description("Total amount of users")
        .register(registry);
  }

  private double count() {
    try {
      return userManager.getTotalCount();
    } catch (ServerException e) {
      return Double.NaN;
    }
  }
```

  Alternatively, the metric can be stored with a reference and updated manually in some other place in the code.

Additional resources

- Metric and label naming for Prometheus
- Metric types for Prometheus

CHAPTER 4. 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 transparent 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.

4.1. 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 instead of repeating the whole instrumentation process for the new distributed tracing system, the developer can simply change the configuration of the tracer back end.

4.2. 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.

4.3. 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 4.3.1, “Installing the Jaeger tracing tool for CodeReady Workspaces on OpenShift 4”
- Section 4.3.2, “Installing Jaeger using OperatorHub 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.

4.3.1. Installing the Jaeger tracing tool for CodeReady Workspaces 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.
- CodeReady Workspaces is installed in a 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 CodeReady Workspaces metrics collections to finish the procedure.

4.3.2. 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 deployed 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. The Jaeger cluster **jaeger-all-in-one-inmemory** is created.
20. Follow the steps in Enabling CodeReady Workspaces metrics collections to finish the procedure.

4.4. ENABLING CODEREADY WORKSPACES METRICS COLLECTIONS

Prerequisites

- Installed Jaeger v1.12.0 or above. See instructions at Section 4.3, “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 custom configuration of Jaeger, see the list of Jaeger client environment variables.

4.5. VIEWING CODEREADY WORKSPACES TRACES IN JAEGER UI

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

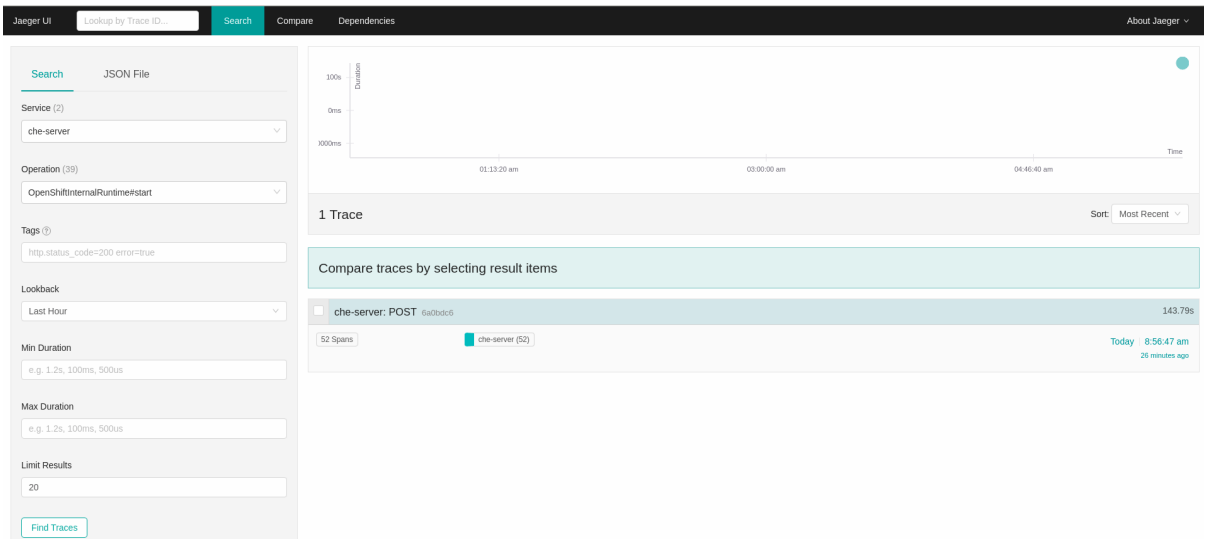
Procedure

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:

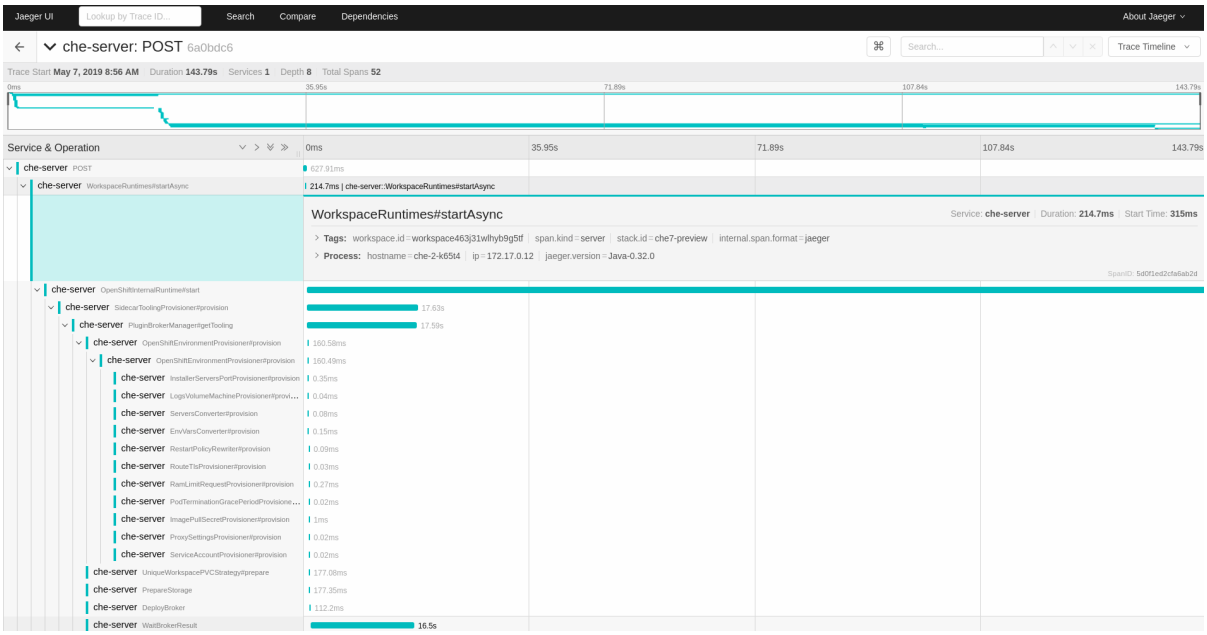
1. In the Search panel on the left, filter spans by the operation name (span name), tags, or time and duration.

   Figure 4.1. 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 4.2. Expanded tracing tree



4.6. 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.

4.6.1. 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 5. MANAGING USERS

This section describes how to configure authorization and authentication in Red Hat CodeReady Workspaces and how to administer user groups and users.

- Configuring authorization
- Removing user data

5.1. CONFIGURING AUTHORIZATION

5.1.1. 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.

Procedure

To find your RH-SSO URL:

- Go to the OpenShift web console and navigate to the RH-SSO project.

5.1.2. Configuring CodeReady Workspaces to work with RH-SSO

The deployment script configures RH-SSO. It creates a **che-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:

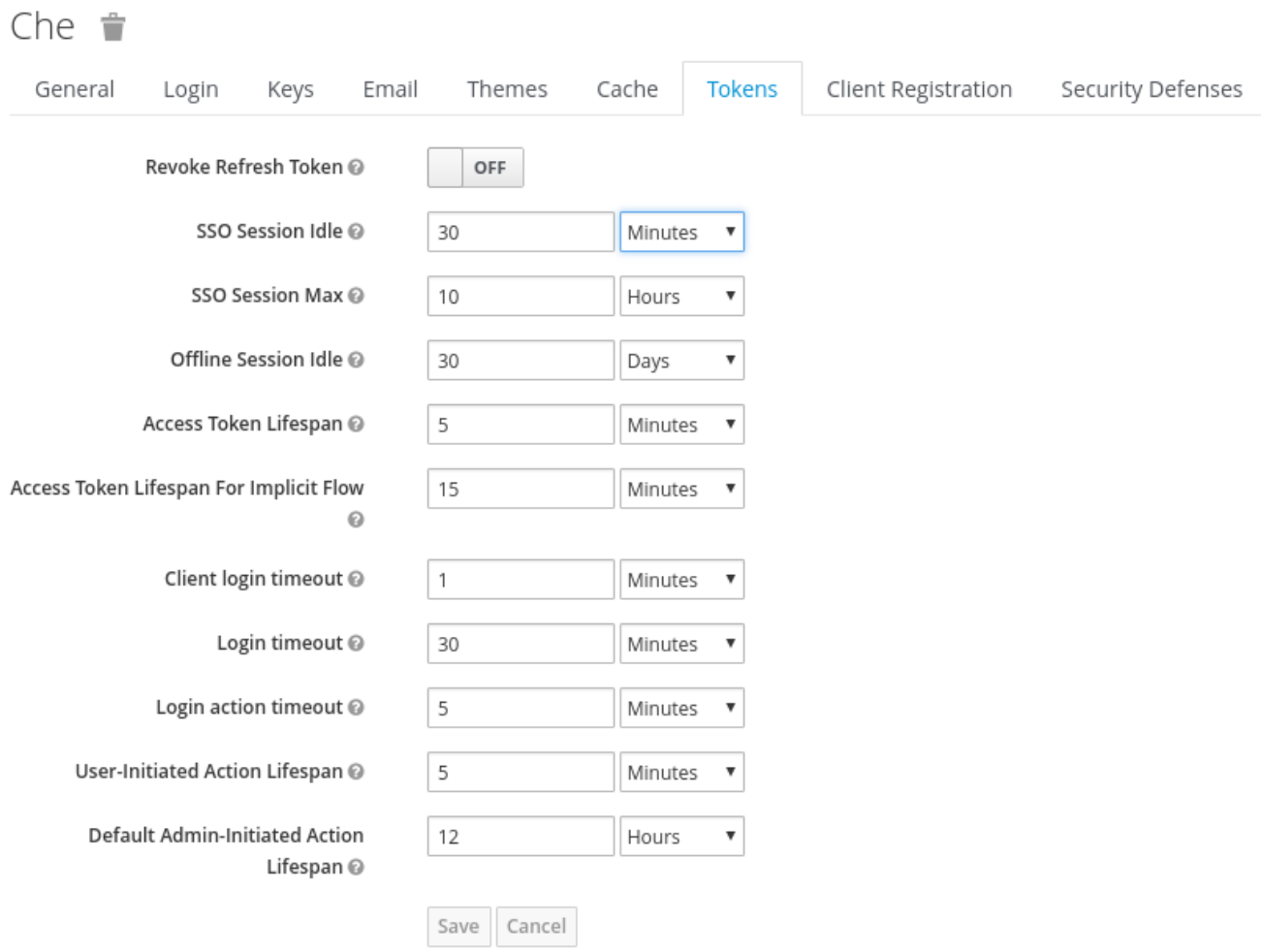
Invalid **redirectURI** error: occurs when you access CodeReady Workspaces at **myhost**, which is an alias, and your original **CODEREADY_HOST** is **1.1.1.1**. If this error occurs, go to the RH-SSO administration console and ensure that the valid redirect URIs are configured.

CORS error: occurs when you have an invalid web origin

5.1.3. Configuring RH-SSO tokens

A user token expires after 30 minutes by default.

You can change the following RH-SSO token settings:



5.1.4. 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.

5.1.5. Enabling authentication with social accounts and brokering

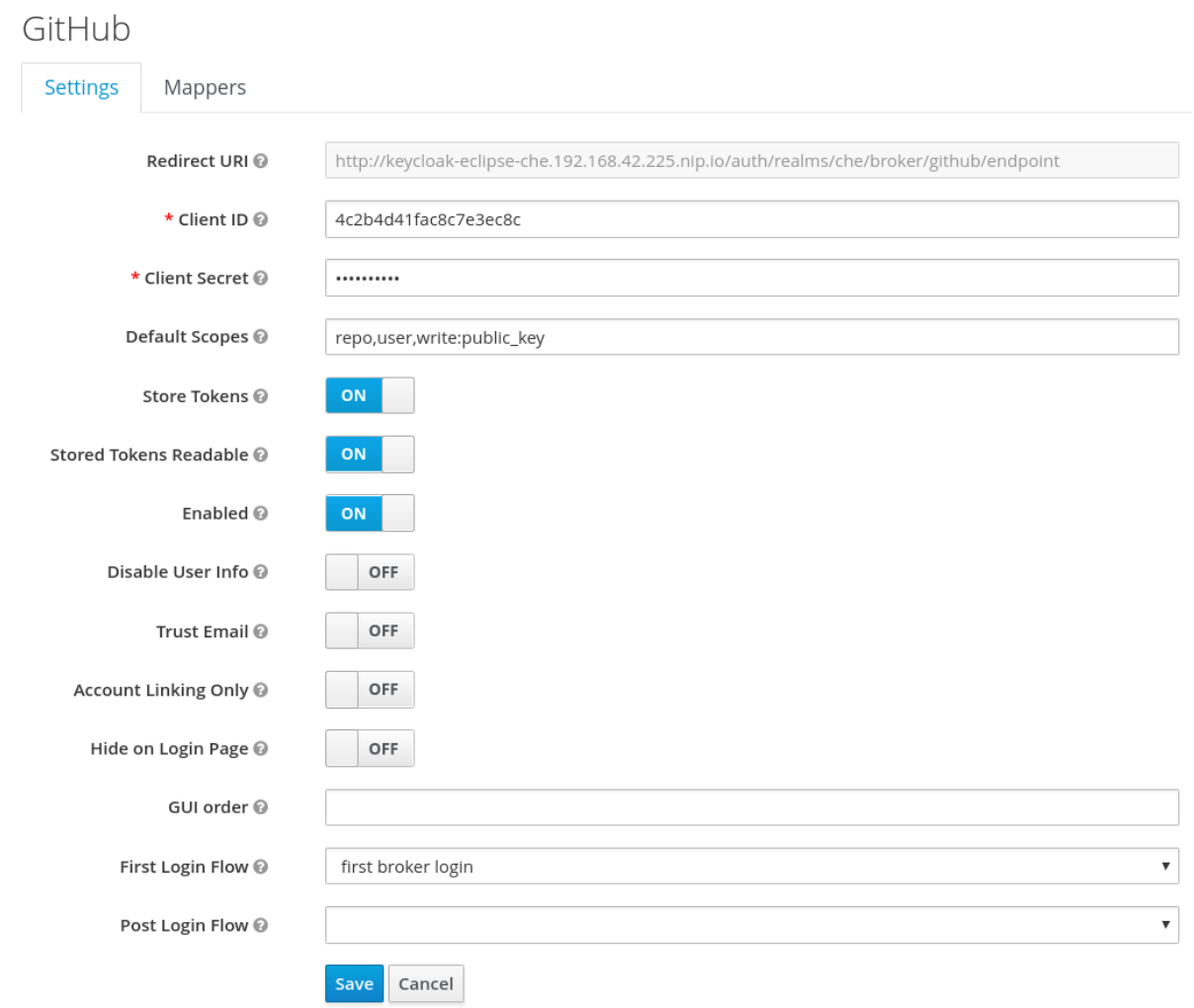
RH-SSO provides built-in support for GitHub, OpenShift, and most common social networks such as Facebook and Twitter.

See Instructions to enable Login with GitHub.

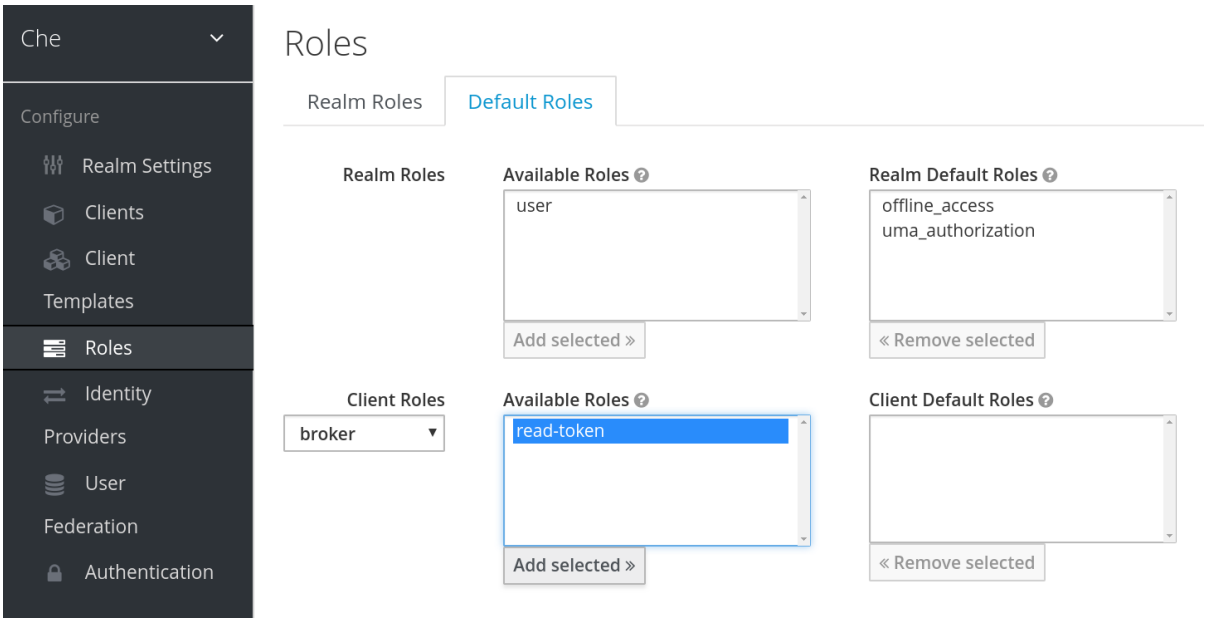
You can also enable the SSH key and upload it to the CodeReady Workspaces users' GitHub accounts.

To enable this feature when you register a GitHub identity provider:

1. Set scope to **repo,user,write:public_key**.
2. Set store tokens and stored tokens readable to ON.



3. **Add a default read-token role.**



This is the default **delegated** OAuth service mode for multiuser CodeReady Workspaces. You can configure the OAuth service mode with the property **che.oauth.service_mode**.

5.1.6. Using protocol-based providers

RH-SSO supports SAML v2.0 and OpenID Connect v1.0 protocols. You can connect your identity provider systems if they support these protocols.

5.1.7. Managing users using RH-SSO

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

5.1.8. 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.

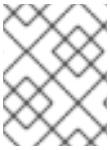
Red Hat CodeReady Workspaces uses the default theme for email templates for registration, email confirmation, password recovery, and failed login.

5.2. REMOVING USER DATA

5.2.1. GDPR

In case user data needs to be deleted, the following API should be used with the **user** or the **admin** authorization token:

```
curl -X DELETE `http(s)://{che-host}/api/user/{id}`
```



NOTE

All the user's workspaces should be stopped beforehand. Otherwise, the API request will fail with **500** Error.

To remove the data of all the users, follow instructions for Uninstalling Red Hat CodeReady Workspaces.

CHAPTER 6. SECURING CODEREADY WORKSPACES

This section describes all aspects of user authentication, types of authentication, and permissions models on the CodeReady Workspaces server and its workspaces.

- Authenticating users
- Authorizing users

6.1. 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 byKeycloak.

Authentication in workspaces implies the issuance of self-signed per-workspace JWT tokens and their verification on a dedicated service based on JWTProxy.

6.1.1. Authenticating to the CodeReady Workspaces server

6.1.1.1. Authenticating to the CodeReady Workspaces server using OpenID

OpenID authentication on the CodeReady Workspaces server implies the presence of an external OpenID Connect provider and has the following main steps:

- Authenticate the user through a JWT token that is retrieved from an HTTP request or, in case of a missing or invalid token, redirect the user to the RH-SSO login page.
- Send authentication tokens in an Authorization header. In limited cases, when it is impossible to use the Authorization header, the token can be sent in the token query parameter. Example: OAuth authentication initialization.
- Compose an internal **subject** object that represents the current user inside the CodeReady Workspaces server code.

NOTE

The only supported and tested OpenID provider is RH-SSO.

Procedure

To authenticate to the CodeReady Workspaces server using OpenID authentication:

1. Request the OpenID settings service where clients can find all the necessary URLs and properties of the OpenId provider, such as **jwks.endpoint**, **token.endpoint**, **logout.endpoint**, **realm.name**, or **client_id** returned in the JSON format.
2. The service URL is **https://codeready-<openshift_deployment_name>.<domain_name>/api/keycloak/settings**, and it is only available in the CodeReady Workspaces multiuser mode. The presence of the service in the URL confirms that the authentication is enabled in the current deployment.

Example output:

```
{
   "che.keycloak.token.endpoint":
"http://172.19.20.9:5050/auth/realms/che/protocol/openid-connect/token",
   "che.keycloak.profile.endpoint": "http://172.19.20.9:5050/auth/realms/che/account",
   "che.keycloak.client_id": "che-public",
   "che.keycloak.auth_server_url": "http://172.19.20.9:5050/auth",
   "che.keycloak.password.endpoint":
"http://172.19.20.9:5050/auth/realms/che/account/password",
   "che.keycloak.logout.endpoint":
"http://172.19.20.9:5050/auth/realms/che/protocol/openid-connect/logout",
   "che.keycloak.realm": "che"
}
```

The service allows downloading the JavaScript client library to interact with the provider using the **https://codeready-<openshift_deployment_name>.<domain_name>/api/keycloak/OIDCKeycloak.js** URL.

3. Redirect the user to the appropriate provider's login page with all the necessary parameters, including **client_id** and the return redirection path. This can be done with any client library (JS or Java).

4. When the user is logged in to the provider, the client side-code is obtained, and the JWT token has validated the token, the creation of the **subject** begins.

The verification of the token signature occurs in two main steps:

1. Authentication: The token is extracted from the Authorization header or from the**token** query parameter and is parsed using the public key retrieved from the provider. In case of expired, invalid, or malformed tokens, a **403** error is sent to the user. The minimal use of the query parameter is recommended, due to its support limitations or complete removal in upcoming versions.
   If the validation is successful, the parsed form of the token is passed to the environment initialization step:

2. Environment initialization: The filter extracts data from the JWT token claims, creates the user in the local database if it is not yet present, and constructs the **subject** object and sets it into the per-request EnvironmentContext object, which is statically accessible everywhere.
   If the request was made using only a machine token, the following single authentication filter is used:

   org.eclipse.che.multiuser.machine.authentication.server.MachineLoginFilter: The filter finds the user that the **userId** token belongs to, retrieves the user instance, and sets the principal to the session. The CodeReady Workspaces server-to-server requests are performed using a dedicated request factory that signs every request with the current subject token obtained from the **EnvironmentContext** object.

NOTE

Providing user-specific data

Since RH-SSO may store user-specific information (first and last name, phone number, job title), there is a special implementation of the ProfileDao that can provide this data to consumers. The implementation is read-only, so users cannot perform create and update operations.

6.1.1.1.1. Obtaining the token from credentials through RH-SSO

Clients that cannot run JavaScript or other clients (such as command-line clients or Selenium tests) must request the authorization token directly from RH-SSO.

To obtain the token, send a request to the token endpoint with the username and password credentials. This request can be schematically described as the following cURL request:

```
$ curl --data
"grant_type=password&client_id=<client_name>&username=<username>&password=<password>" \
  http://<keyckloak_host>:5050/auth/realms/<realm_name>/protocol/openid-connect/token
```

The CodeReady Workspaces dashboard uses a customized RH-SSO login page and an authentication mechanism based on **grant_type=authorization_code**. It is a two-step authentication process:

1. Logging in and obtaining the authorization code.
2. Obtaining the token using this authorization code.

6.1.1.1.2. Obtaining the token from the OpenShift token through RH-SSO

When CodeReady Workspaces is installed on OpenShift using the Operator, and the OpenShift OAuth integration is enabled, as it is by default, the user's CodeReady Workspaces authentication token can be retrieved from the user's OpenShift token.

To retrieve the authentication token from the OpenShift token, send a schematically described cURL request to the OpenShift token endpoint:

```
$ curl -X POST -d "client_id=<client_name>" \
--data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
-d "subject_token=<user_openshift_token>" \
 -d "subject_issuer=<openshift_identity_provider_name>" \
 --data-urlencode "subject_token_type=urn:ietf:params:oauth:token-type:access_token" \
 http://<keyckloak_host>:5050/auth/realms/<realm_name>/protocol/openid-connect/token
```

The default values for ***<openshift_identity_provider_name>*** are:

- On OpenShift 3.11: **openshift-v3**
- On OpenShift 4.x: **openshift-v4**

<user_openshift_token> is the token retrieved by the end-user with the command:

```
$ oc whoami --show-token
```

WARNING

Before using this token exchange feature, it is required for an end user to be interactively logged in at least once to the CodeReady Workspaces Dashboard using the OpenShift login page. This step is needed to link the OpenShift and RH-SSO user accounts properly and set the required user profile information.

6.1.1.2. 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.

6.1.1.3. 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 based on the RH-SSO brokering mechanism. The following are the two main OAuth API implementations:

internal

org.eclipse.che.security.oauth.EmbeddedOAuthAPI

external

org.eclipse.che.multiuser.keycloak.server.oauth2.DelegatedOAuthAPI

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 **org.eclipse.che.security.oauth.OAuthAuthenticationService**, which contains:

- An authentication method that the OAuth authentication flow can start with.

- A callback method to process callbacks from the provider.
- A token to retrieve the current user's OAuth token.

These methods apply to the currently activated, embedded or delegated, OAuthAPI. The OAuthAPI then provides the following underlying operations:

- Finding the appropriate authenticator.
- Initializing the login process.
- Forwarding the user.

6.1.1.4. 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.

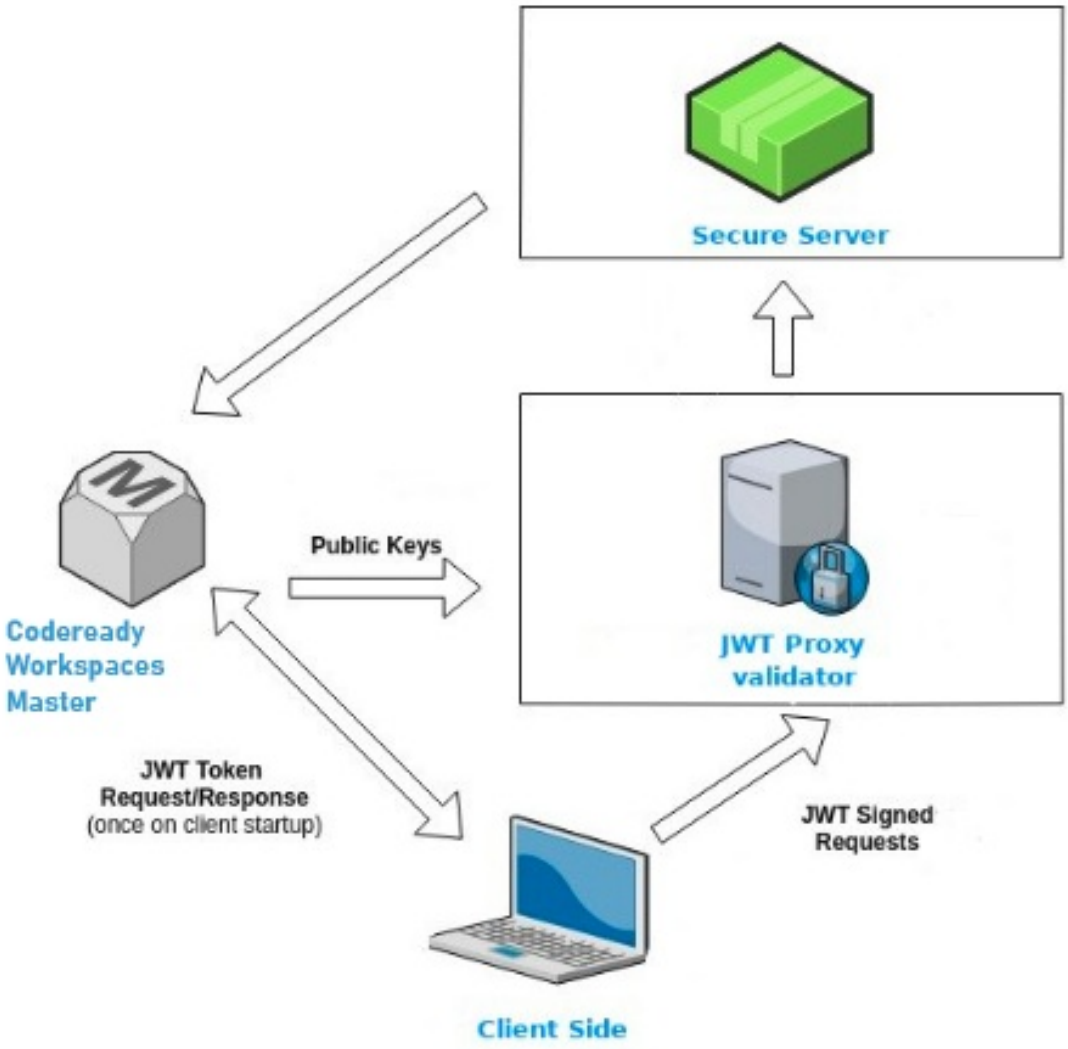
Access the CodeReady Workspaces Swagger interface at **https://codeready-<openshift_deployment_name>.<domain_name>/swagger**. The user must be signed in through RH-SSO, so that the access token is included in the Request header.

6.1.2. 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.

Machine 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 6.1. Authentication inside a workspace



6.1.2.1. 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'
```

6.1.2.2. 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 machine 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.

6.1.2.3. Machine token validation

The validation of machine 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.

6.2. 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**.

6.2.1. 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 6.1. 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.

6.2.2. 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 6.2. 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.

6.2.3. 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.

6.2.4. 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.

6.2.5. Listing CodeReady Workspaces permissions

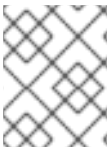
To list CodeReady Workspaces permissions that apply to a specific resource, perform the **GET /permissions** request.

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.

6.2.6. 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.

6.2.7. 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.

CHAPTER 7. BACKUP AND DISASTER RECOVERY

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

- External database setup
- Persistent Volumes backups

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)

When using an external PostgreSQL database, it is also necessary to use an external RH-SSO.

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)

To make the Operator skip deploying a database and pass connection details of an existing

database to a CodeReady Workspaces server using **chePostgresHostName**, **chePostgresPort**, **chePostgresUser**, **chePostgresPassword**, **chePostgresDb** parameters, specify the**externalDb: true** property in the CodeReady Workspaces Custom Resource.

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

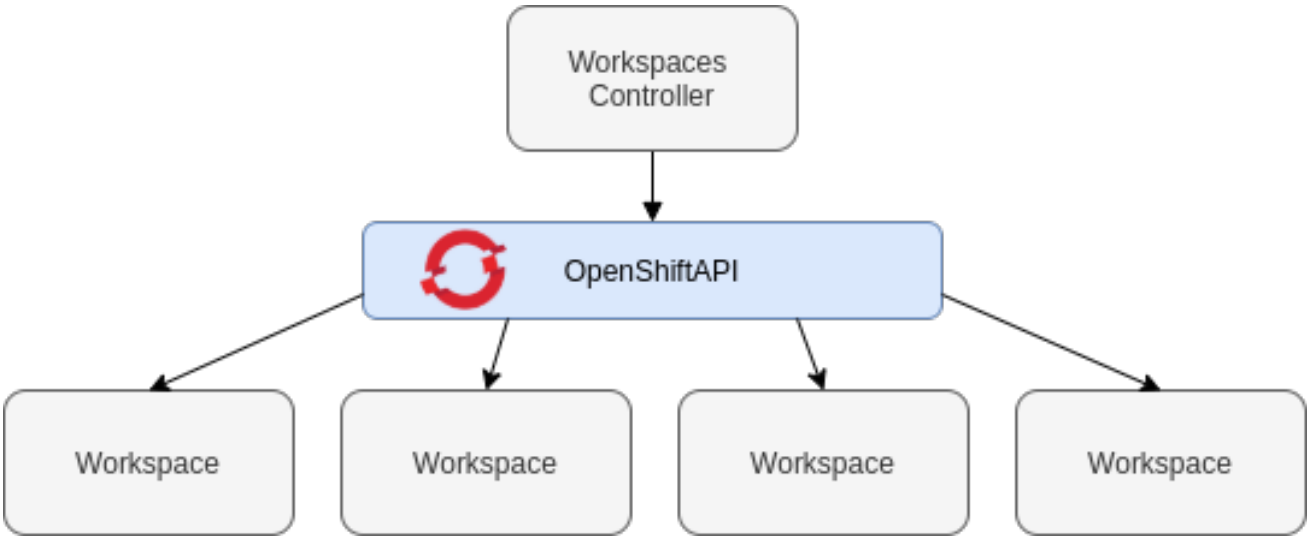
CHAPTER 8. CALCULATING CODEREADY WORKSPACES RESOURCE REQUIREMENTS

This section describes how to calculate resources (memory and CPU) required to run Red Hat CodeReady Workspaces.

8.1. CODEREADY WORKSPACES ARCHITECTURAL COMPONENTS

As illustrated in the High-level CodeReady Workspaces architecturearticle, Red Hat CodeReady Workspaces components are:

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



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. For a detailed explanation of how OpenShift manages container-resource requests and limits, see OpenShift documentation.

8.2. 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 8.1. 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

Pod	Container name	Default memory limit	Default memory request
Red Hat Single Sign-On	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 configurations options article for instructions on how to override the default requests and limits. For example, the hosted version of CodeReady Workspaces that runs on che.openshift.io uses 4 GiB of memory.

8.3. 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 10 active plug-ins requires more resources then 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 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 8.2. 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	128 MiB
cheEditor	**che.workspace.sidecar.default_memory_limit_mb**	128 MiB	128 MiB
kubernetes, **openshift**, **dockerimage**	**che.workspace.default_memory_limit_mb**, **che.workspace.default_memory_request_mb**	1 Gi	512 MiB
JWT Proxy	**che.server.secure_exposer.jwtproxy.memory_limit**	128 MiB	128 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 8.1. 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
> ```
>
> It results in a container with the following memory limit and request:
>
> | | |
> |---|---|
> | Memory limit | 512 MiB |
> | Memory request | 256 MiB |

TIP

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**.

For example, on a local Minikube cluster, the URL for the **che-incubator/typescript/latest meta.yaml** is **http://plugin-registry-che.192.168.64.78.mycluster.mycompany.com/v3/plugins/che-incubator/typescript/latest/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 8.2. 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 requirements previously calculated.

8.4. 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 8.3. Total workspace memory requirement and limit

Pod	Container name	Default memory limit	Default memory request
Workspace	theia-ide (default **cheEditor**)	512 MiB	512 MiB

Pod	Container name	Default memory limit	Default memory request
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.

8.5. ADDITIONAL RESOURCES

- High-level CodeReady Workspaces architecture
- OpenShift compute resources management documentation
- Advanced CodeReady Workspaces configuration options
- Devfile documentation
- A devfile that has no components
- JWT Proxy
- che-plugin-registry GitHub repository

CHAPTER 9. CACHING IMAGES FOR FASTER WORKSPACE START

This section describes installing the Image Puller on a CodeReady Workspaces cluster to cache images on cluster nodes.

9.1. IMAGE PULLER OVERVIEW

Slow starts of Red Hat CodeReady Workspaces workspaces may be caused by waiting for the underlying cluster to pull images used in workspaces from remote registries. As such, pre-pulling images can improve start times significantly. The *Image Puller* can be used to pre-pull images and shorten workspace start times.

The Image Puller is an additional deployment that runs alongside Red Hat CodeReady Workspaces. Given a list of images to pre-pull, the application runs inside a cluster and creates a *DaemonSet* that pulls the images on each node.

NOTE

The minimal requirement for an image to be pre-pulled is the availability of the **sleep** command, which means that **FROM scratch** images (for example, 'che-machine-exec') are currently not supported. Also, images that mount volumes in the dockerfile are not supported for pre-pulling on OpenShift.

The application can be deployed:

- using OperatorHub or installing the kubernetes image puller operator
- by processing and applying OpenShift templates.

The Image Puller pulls its configuration from a **ConfigMap** with the following available parameters:

Table 9.1. Image Puller default parameters

Parameter	Usage	Default
CACHING_INTERVAL_HOURS	Interval, in hours, between checking health of DaemonSets	**"1"**
CACHING_MEMORY_REQUEST	The memory request for each cached image when the puller is running	**10Mi**
CACHING_MEMORY_LIMIT	The memory limit for each cached image when the puller is running	**20Mi**
CACHING_CPU_REQUEST	The CPU request for each cached image when the puller is running	**.05**
CACHING_CPU_LIMIT	The CPU limit for each cached image when the puller is running	**.2**

Parameter	Usage	Default
DAEMONSET_NAME	Name of DaemonSet to be created	**kubernetes-image-puller**
NAMESPACE	Namespace where DaemonSet is to be created	**k8s-image-puller**
IMAGES	List of images to be cached, in the format **<name>=<image>; ...**	Contains a default list of images. Before deploying, fill this with the images that fit the current requirements
NODE_SELECTOR	Node selector applied to the Pods created by the DaemonSet	**'{}'**

The default memory requests and limits ensure that the container has enough memory to start. When changing **CACHING_MEMORY_REQUEST** or **CACHING_MEMORY_LIMIT**, you will need to consider the total memory allocated to the DaemonSet Pods in the cluster:

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

For example, running the image puller that caches 5 images on 20 nodes, with a container memory limit of **20Mi** requires **2000Mi** of memory.

9.2. DEPLOYING IMAGE PULLER USING THE OPERATOR

The recommended way to deploy the Image Puller is through the Operator.

9.2.1. Installing the Image Puller on OpenShift using OperatorHub

First, create a project in your cluster to host the image puller. Our example will use the project "image-puller".

Navigate to your OpenShift cluster's console, and select Operators. Select OperatorHub and type "image puller" into the "Filter by keyword.." search bar. Click the OpenShift Image Puller Operator, click Continue and click Install. At the Installation Mode selection, choose A specific project on the cluster, and use the drop-down to find the project you created to install the image puller. Click Subscribe.

Wait for the OpenShift Image Puller Operator to install, and click the installation. Click the OpenShiftImagePuller tab, and then click Create instance. You will be taken to a screen with a YAML editor with a **OpenShiftImagePuller** Custom Resource. Make any modifications to the Custom resource and click Create.

Navigate to the Workloads and Pods menu in the project that the image puller was installed, and you should see pods being created.

9.2.2. Installing the Image Puller on OpenShift using the Operator

Create a project to host the kubernetes image puller, and apply the following manifests from the GitHub repository:

```
export NAMESPACE=<namespace you created to host the image puller>
oc apply -f https://raw.githubusercontent.com/che-incubator/kubernetes-image-puller-operator/master/deploy/crds/che.eclipse.org_kubernetesimagepullers_crd.yaml -n $NAMESPACE
oc apply -f https://raw.githubusercontent.com/che-incubator/kubernetes-image-puller-operator/master/deploy/role.yaml -n $NAMESPACE
oc apply -f https://raw.githubusercontent.com/che-incubator/kubernetes-image-puller-operator/master/deploy/role_binding.yaml -n $NAMESPACE
oc apply -f https://raw.githubusercontent.com/che-incubator/kubernetes-image-puller-operator/master/deploy/service_account.yaml -n $NAMESPACE
oc apply -f https://raw.githubusercontent.com/che-incubator/kubernetes-image-puller-operator/master/deploy/operator.yaml -n $NAMESPACE
```

Then create a **OpenShiftImagePuller** Custom Resource:

```
apiVersion: che.eclipse.org/v1alpha1
kind: KubernetesImagePuller
metadata:
  name: image-puller
  namespace: <namespace you installed the image puller in>
spec:
  configMapName: k8s-image-puller
  daemonsetName: k8s-image-puller
  deploymentName: kubernetes-image-puller
  images: >-
    java11-maven=quay.io/eclipse/che-java11-maven:nightly;che-theia=quay.io/eclipse/che-theia:next
```

9.3. DEPLOYING IMAGE PULLER USING OPENSHIFT TEMPLATES

The Image Puller repository contains OpenShift templates for deploying on OpenShift.

Prerequisites

- A running OpenShift cluster.
- The **oc** binary file.

The following parameters are available to further configure the OpenShift templates:

Table 9.2. Parameters for installing with OpenShift templates

Value	Usage	Default
DAEMONSET_NAME	The value of **DAEMONSET_NAME** to set in the ConfigMap	**kubernetes-image-puller**

Value	Usage	Default
IMAGE	Image used for the **kubernetes-image-puller** deployment	**registry.redhat.io/codeready-workspaces/imagepuller-rhel8:2.2**
IMAGE_TAG	The image tag to pull	**2.2**
SERVICEACCOUNT_NAME	The name of the ServiceAccount used by the deployment (created as part of installation)	**k8s-image-puller**
CACHING_INTERVAL_HOURS	The value of **CACHING_INTERVAL_HOURS** to set in the ConfigMap	**"1"**
CACHING_INTERVAL_REQUEST	The value of **CACHING_MEMORY_REQUEST** to set in the ConfigMap	**"10Mi"**
CACHING_INTERVAL_LIMIT	The value of **CACHING_MEMORY_LIMIT** to set in the ConfigMap	**"20Mi"`**
NODE_SELECTOR	The value of **NODE_SELECTOR** to set in the ConfigMap	**"{}"**

See Table 9.1, "Image Puller default parameters"for more information about configuration values, such as **DAEMONSET_NAME**, **CACHING_INTERVAL_HOURS**, and **CACHING_MEMORY_REQUEST**.

Table 9.3. List of recommended images to pre-pull

Image	URL	Tag
stacks-java-rhel8	registry.access.redhat.com/codeready-workspaces/stacks-java-rhel8	2.2
theia-rhel8	registry.access.redhat.com/codeready-workspaces/theia-rhel8	2.2
stacks-golang-rhel8	registry.access.redhat.com/codeready-workspaces/stacks-golang-rhel8	2.2
stacks-node-rhel8	registry.access.redhat.com/codeready-workspaces/stacks-node-rhel8	2.2

Image	URL	Tag
theia-endpoint-rhel8	registry.access.redhat.com/codeready-workspaces/theia-rhel8	2.2
pluginbroker-metadata-rhel8	registry.access.redhat.com/codeready-workspaces/pluginbroker-metadata-rhel8	2.2
pluginbroker-artifacts-rhel8	registry.access.redhat.com/codeready-workspaces/pluginbroker-artifacts-rhel8	2.2

See Table 9.1, "Image Puller default parameters" for more information about configuration values, such as **DAEMONSET_NAME**, **CACHING_INTERVAL_HOURS**, and **CACHING_MEMORY_REQUEST**.

Procedure

Installing

1. **Clone the kubernetes-image-puller repository:**

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

2. **Create a new OpenShift project to deploy the puller into:**

```
$ oc new-project k8s-image-puller
```

3. **Process and apply the templates to deploy the puller:**
In CodeReady Workspaces you must use custom values to deploy the image puller. To set custom values, add to the oc process an option:-p *<parameterName>=<value>*:

```
$ oc process -f deploy/serviceaccount.yaml \
    | oc apply -f -
$ oc process -f deploy/configmap.yaml \
    -p IMAGES='stacks-java-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-java-rhel8:2.2;\
    theia-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:2.2;\
    stacks-golang-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-golang-rhel8:2.2;\
    stacks-node-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-node-rhel8:2.2;\
    theia-endpoint-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:2.2;\
    pluginbroker-metadata-rhel8=registry.access.redhat.com/codeready-workspaces/pluginbroker-metadata-rhel8:2.2;\
    pluginbroker-artifacts-rhel8=registry.access.redhat.com/codeready-workspaces/pluginbroker-artifacts-rhel8:2.2;' \
    | oc apply -f -
$ oc process -f deploy/app.yaml \
```

```
    -p IMAGE=registry.redhat.io/codeready-workspaces/imagepuller-rhel8 \
    -p IMAGE_TAG='2.2' \
    | oc apply -f -
```

Verifying the installation

1. Confirm that a new deployment, **kubernetes-image-puller**, and a DaemonSet (named based on the value of the **DAEMONSET_NAME** parameter) exist. The DaemonSet needs to have a Pod for each node in the cluster:

```
$ oc get deployment,daemonset,pod --namespace k8s-image-puller
deployment.extensions/kubernetes-image-puller   1/1     1         1         2m19s

NAME                                    DESIRED   CURRENT   READY     UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.extensions/kubernetes-image-puller   1         1         1         1
1           <none>          2m10s

NAME                                    READY     STATUS    RESTARTS   AGE
pod/kubernetes-image-puller-5495f46497-mkd4p   1/1       Running   0          2m18s
pod/kubernetes-image-puller-n8bmf              3/3       Running   0          2m10s
```

2. Check that the **ConfigMap** named **k8s-image-puller** has the values you specified in your parameter substitution, or that they contain the default values:

```
$ oc get configmap k8s-image-puller --output yaml
apiVersion: v1
data:
  CACHING_INTERVAL_HOURS: "1"
  CACHING_MEMORY_LIMIT: 20Mi
  CACHING_MEMORY_REQUEST: 10Mi
  DAEMONSET_NAME: kubernetes-image-puller
  IMAGES: |
    stacks-java-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-java-rhel8:
{prod-ver};
    theia-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:{prod-ver};
    stacks-golang-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-golang-
rhel8:{prod-ver};
    stacks-node-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-node-rhel8:
{prod-ver};
    theia-endpoint-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:{prod-
ver};
    pluginbroker-metadata-rhel8=registry.access.redhat.com/codeready-
workspaces/pluginbroker-metadata-rhel8:{prod-ver};
    pluginbroker-artifacts-rhel8=registry.access.redhat.com/codeready-
workspaces/pluginbroker-artifacts-rhel8:{prod-ver};
  NAMESPACE: k8s-image-puller
  NODE_SELECTOR: '{}'
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":
{"CACHING_INTERVAL_HOURS":"1","CACHING_MEMORY_LIMIT":"20Mi","CACHING_ME
MORY_REQUEST":"10Mi","DAEMONSET_NAME":"kubernetes-image-
```

puller","IMAGES":"stacks-java-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-java-rhel8:{prod-ver}; theia-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:{prod-ver}; stacks-golang-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-golang-rhel8:{prod-ver}; stacks-node-rhel8=registry.access.redhat.com/codeready-workspaces/stacks-node-rhel8:{prod-ver}; theia-endpoint-rhel8=registry.access.redhat.com/codeready-workspaces/theia-rhel8:{prod-ver}; pluginbroker-metadata-rhel8=registry.access.redhat.com/codeready-workspaces/pluginbroker-metadata-rhel8:{prod-ver}; pluginbroker-artifacts-rhel8=registry.access.redhat.com/codeready-workspaces/pluginbroker-artifacts-rhel8:{prod-ver};\n","NAMESPACE":"k8s-image-puller","NODE_SELECTOR":"{}"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"k8s-image-puller","namespace":"k8s-image-puller"},"type":"Opaque"}
  creationTimestamp: 2020-02-17T22:40:13Z
  name: k8s-image-puller
  namespace: k8s-image-puller
  resourceVersion: "72250"
  selfLink: /api/v1/namespaces/k8s-image-puller/configmaps/k8s-image-puller
  uid: 76430ed6-51d6-11ea-9c19-52fdfc072182

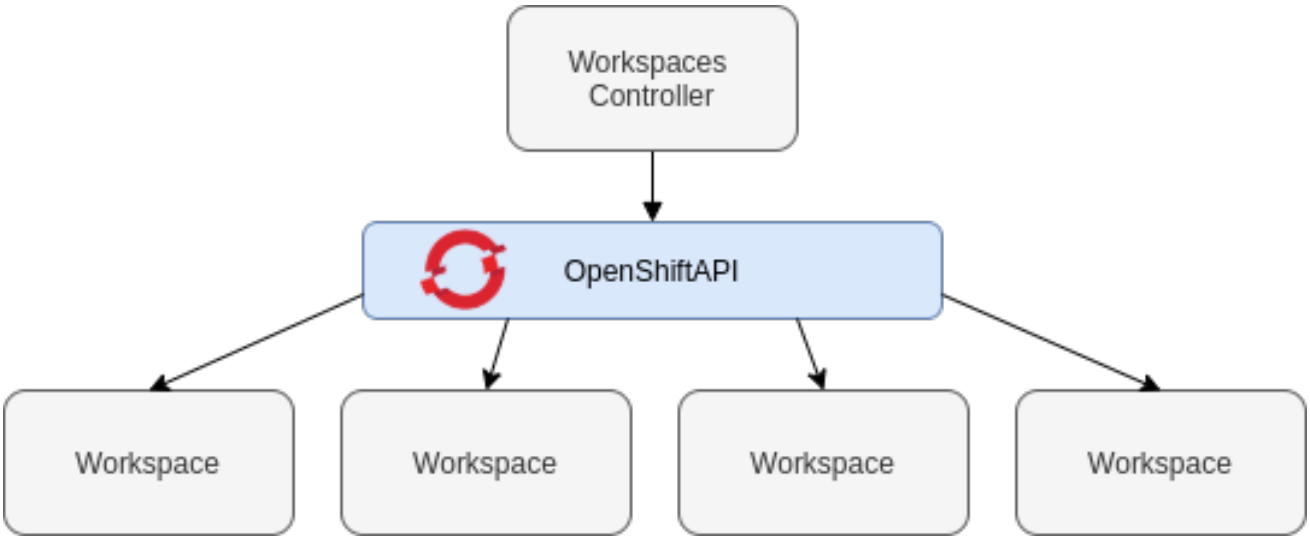
CHAPTER 10. CODEREADY WORKSPACES ARCHITECTURAL ELEMENTS

This section contains information about the CodeReady Workspaces architecture. Section starts with a High-level CodeReady Workspaces architecture overview and continues with providing information about the CodeReady Workspaces workspace controller and CodeReady Workspaces workspace architecture.

- High-level CodeReady Workspaces architecture
- CodeReady Workspaces workspace controller
- CodeReady Workspaces workspaces architecture

10.1. HIGH-LEVEL CODEREADY WORKSPACES ARCHITECTURE

Figure 10.1. High-level CodeReady Workspaces architecture



At a high-level, CodeReady Workspaces is composed of one central workspace controller that manages the CodeReady Workspaces workspaces through the OpenShift API.

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.

This section describes the different services that create the workspaces controller and the CodeReady Workspaces workspaces.

- CodeReady Workspaces workspace controller
- CodeReady Workspaces workspaces architecture

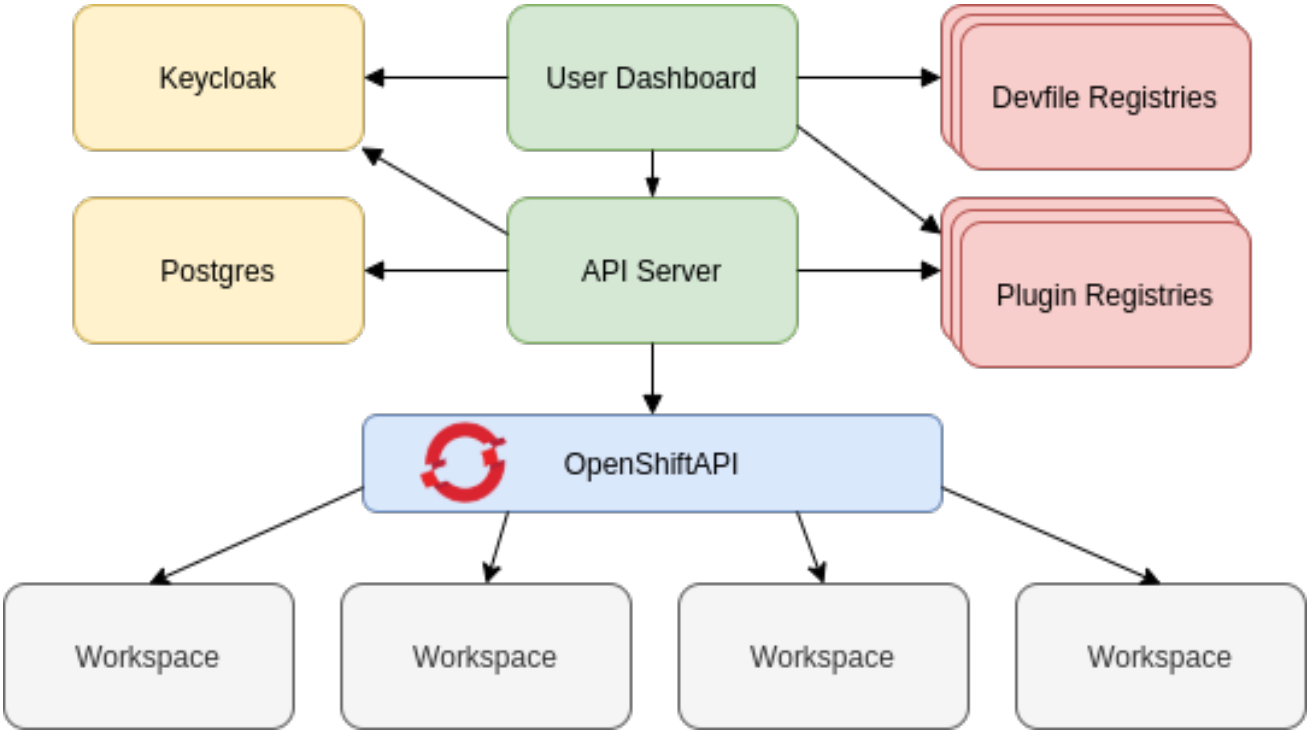
10.2. CODEREADY WORKSPACES WORKSPACE CONTROLLER

The workspaces controller manages the container-based development environments: CodeReady Workspaces workspaces. It can be deployed in the following distinct configurations:

- Single-user: No authentication service is set up. Development environments are not secured. This configuration requires fewer resources. It is more adapted for local installations, such as when using Minikube.
- Multi-user: This is a multi-tenant configuration. Development environments are secured, and this configuration requires more resources. Appropriate for cloud installations.

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

Figure 10.2. CodeReady Workspaces workspaces controller



10.2.1. CodeReady Workspaces server

The CodeReady Workspaces server, also known as wsmaster, 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.

Source code	Red Hat CodeReady Workspaces GitHub
Container image	**eclipse/che-server**
Environment variables	Advanced configuration options for the Che server component

10.2.2. 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.

Source code	CodeReady Workspaces Dashboard
Container image	**eclipse/che-server**

10.2.3. 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.

Source code	CodeReady Workspaces Devfile registry
Container image	**quay.io/crw/che-devfile-registry**

10.2.4. 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 wsmaster connects.

For more information about plug-in registry customization, see the Chapter 1, *Customizing the devfile and plug-in registries* section.

Source code	CodeReady Workspaces plug-in registry
Container image	**quay.io/crw/che-plugin-registry**

10.2.5. 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.

Source code	CodeReady Workspaces Postgres
Container image	**eclipse/che-postgres**

10.2.6. 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.

Source code	CodeReady Workspaces Keycloak
Container image	**eclipse/che-keycloak**

10.3. 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 a number of 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:

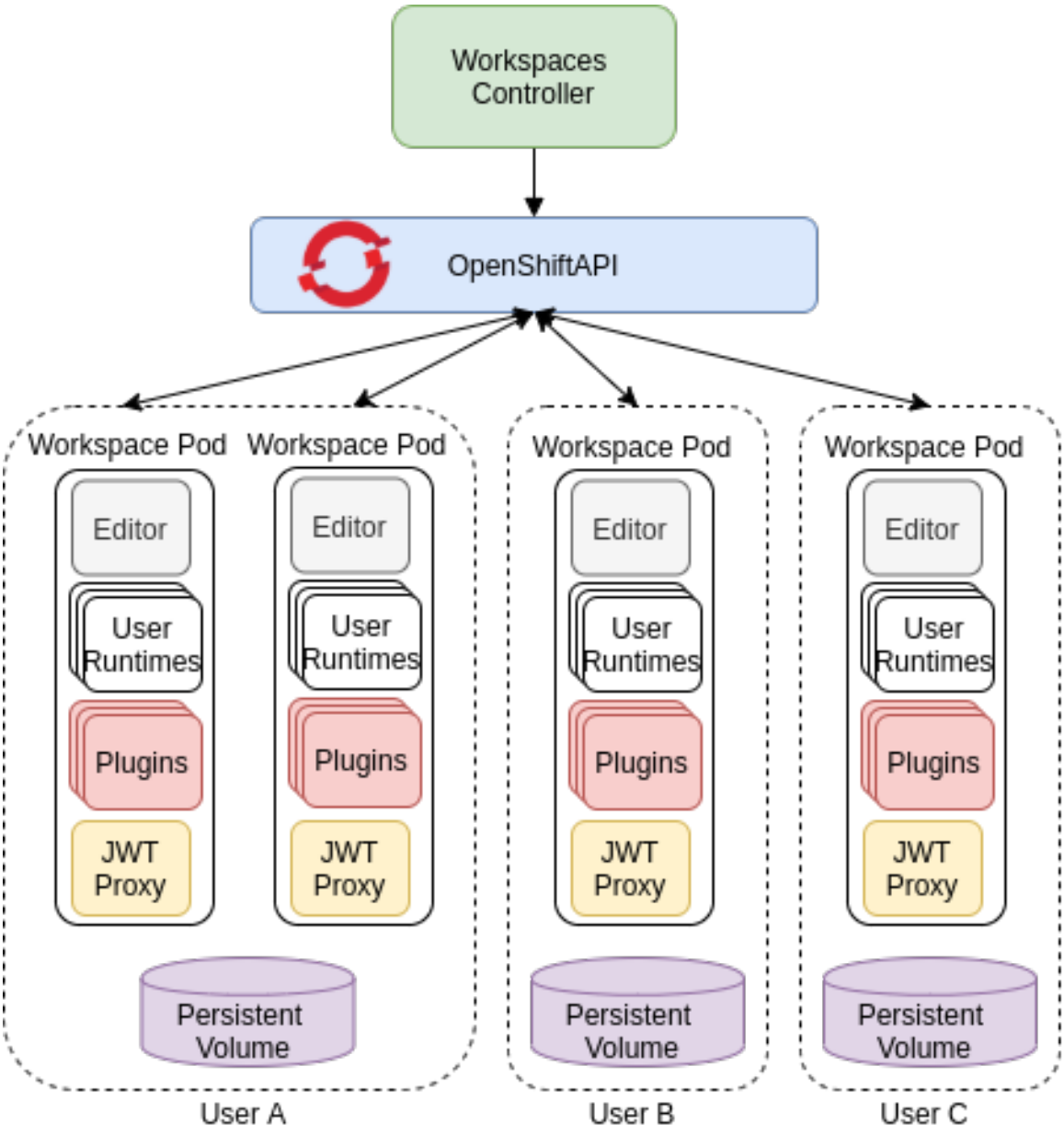
- configmaps
- services
- endpoints
- ingresses/routes
- secrets
- PVs

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 (an editor, language auto-completion, 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 10.3. CodeReady Workspaces workspace components



In the diagram, there are three running workspaces: two belonging to User A and one toUser 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.

10.3.1. CodeReady Workspaces workspace components

This section describes the components of a CodeReady Workspaces workspace.

10.3.1.1. **Che Plugin** plug-ins

Che Plugin plug-ins are special services that extend CodeReady Workspaces workspace capabilities. **Che Plugin** plug-ins are packaged as containers. Packaging plug-ins into a container has the following benefits:

- It isolates the plug-ins from the main IDE, therefore limiting the resources that a plug-in has access to.

- It uses the consolidated standard of container registries to publish and distribute plug-ins (as with any container image).

The containers that plug-ins are packaged into run as sidecars of the CodeReady Workspaces workspace editor and augment its capabilities.

Visual Studio Code extensions packaged in containers are CodeReady Workspaces plug-ins for the Che-Theia editor.

Multiple CodeReady Workspaces plug-ins can run in the same container (for better resource use), or a Che Plugin can run in its dedicated container (for better isolation).

10.3.1.2. **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.

The Che-Theia source-code repository is at Che-Theia Github. It is based on theEclipse Theia open-source project.

Che-Theia is written in TypeScript and is built on the Microsoft Monaco editor. 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**

10.3.1.3. 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 app dev/prod parity.

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

10.3.1.4. 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**

10.3.1.5. 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	**eclipse/che-init-plugin-broker** **eclipse/che-unified-plugin-broker**

10.3.2. CodeReady Workspaces workspace configuration

This section describes the properties of the CodeReady Workspaces server that affect the provisioning of a CodeReady Workspaces workspace.

10.3.2.1. Storage strategies for codeready-workspaces workspaces

Workspace Pods use Persistent Volume Claims (PVCs), which are bound to the physical Persistent Volumes (PVs) with ReadWriteOnce access mode. It is possible to configure how the CodeReady Workspaces server uses PVCs for workspaces. The individual methods for this configuration are called PVC strategies:

strategy	details	pros	cons
unique	One PVC per workspace volume or user-defined PVC	Storage isolation	An undefined number of PVs is required

strategy	details	pros	cons
per-workspace (default)	One PVC for one workspace	Easier to manage and control storage compared to unique strategy	PV count still is not known and depends on workspaces number
common	One PVC for all workspaces in one OpenShift namespace	Easy to manage and control storage	If PV does not support ReadWriteMany (RWX) access mode then workspaces must be in a separate OpenShift namespaces Or there must not be more than 1 running workspace per namespace at the same time See how to configure namespace strategy

Red Hat CodeReady Workspaces uses the **common** PVC strategy in combination with the "one project per user" project strategy when all CodeReady Workspaces workspaces operate in the user's project, sharing one PVC.

10.3.2.1.1. The **common** PVC strategy

All workspaces inside a OpenShift-native project use the same Persistent Volume Claim (PVC) as the default data storage when storing data such as the following in their declared volumes:

- projects
- workspace logs
- additional Volumes defined by a use

When the **common** PVC strategy is in use, user-defined PVCs are ignored and volumes that refer to these user-defined PVCs are replaced with a volume that refers to the common PVC. In this strategy, all {prod-short) workspaces use the same PVC. When the user runs one workspace, it only binds to one node in the cluster at a time.

The corresponding containers volume mounts link to a common volume, and sub-paths are prefixed with ***<workspace-ID>*** or ***<original-PVC-name>***. For more details, seeSection 10.3.2.1.4, "How subpaths are used in PVCs".

The CodeReady Workspaces Volume name is identical to the name of the user-defined PVC. It means that if a machine is configured to use a CodeReady Workspaces volume with the same name as the user-defined PVC has, they will use the same shared folder in the common PVC.

When a workspace is deleted, a corresponding subdirectory (**${ws-id}**) is deleted in the PV directory.

Restrictions on using the **common** PVC strategy

When the **common** strategy is used and a workspace PVC access mode is ReadWriteOnce (RWO), only one {admin-context} node can simultaneously use the PVC.

If there are several nodes, you can use the **common** strategy, but:

- The workspace PVC access mode must be reconfigured to **ReadWriteMany** (RWM), so multiple nodes can use this PVC simultaneously.
- Only one workspace in the same project may be running. See Configuring project strategies.

The **common** PVC strategy is not suitable for large multi-node clusters. Therefore, it is best to use it in single-node clusters. However, in combination with **per-workspace** project strategy, the **common** PVC strategy is usable for clusters with around 75 nodes. The PVC used in this strategy must be large enough to accommodate all projects since there is a risk of the event, in which one project depletes the resources of others.

10.3.2.1.2. The **per-workspace** PVC strategy

The **per-workspace** strategy is similar to the**common** PVC strategy. The only difference is that all workspace Volumes, but not all the workspaces, use the same PVC as the default data storage for:

- projects
- workspace logs
- additional Volumes defined by a user

It's a strategy when CodeReady Workspaces keeps its workspace data in assigned PVs that are allocated by a single PVC.

The **per-workspace** PVC strategy is the most universal strategy out of the PVC strategies available and acts as a proper option for large multi-node clusters with a higher amount of users. Using the **per-workspace** PVC strategy, users can run multiple workspaces simultaneously, results in more PVCs being created.

10.3.2.1.3. The **unique** PVC strategy

When using the `unique `PVC strategy, every CodeReady Workspaces Volume of a workspace has its own PVC. This means that workspace PVCs are:

Created when a workspace starts for the first time. Deleted when a corresponding workspace is deleted.

User-defined PVCs are created with the following specifics:

- They are provisioned with generated names to prevent naming conflicts with other PVCs in a project.
- Subpaths of the mounted Physical persistent volumes that reference user-defined PVCs are prefixed with ***<workspace-ID>*** or ***<PVC-name>***. This ensures that the same PV data structure is set up with different PVC strategies. For details, see Section 10.3.2.1.4, "How subpaths are used in PVCs".

The **unique** PVC strategy is suitable for larger multi-node clusters with a lesser amount of users

The **unique** PVC strategy is suitable for larger multi-node clusters with a lesser amount of users. Since this strategy operates with separate PVCs for each volume in a workspace, vastly more PVCs are created.

10.3.2.1.4. How subpaths are used in PVCs

Subpaths illustrate the folder hierarchy in the Persistent Volumes (PV).

```
/pv0001
  /workspaceID1
  /workspaceID2
  /workspaceIDn
    /che-logs
    /projects
    /<volume1>
    /<volume2>
    /<User-defined PVC name 1 | volume 3>
    ...
```

When a user defines volumes for components in the devfile, all components that define the volume of the same name will be backed by the same directory in the PV as ***<PV-name>***, ***<workspace-ID>***, **or `*<original-PVC-name>*. Each component can have this location mounted on a different path in its containers.**

Example

Using the **common** PVC strategy, user-defined PVCs are replaced with subpaths on the common PVC. When the user references a volume as **my-volume**, it is mounted in the common-pvc with the /**workspace-id**/**my-volume** subpath.

10.3.2.2. Configuring a CodeReady Workspaces workspace with a persistent volume strategy

A persistent volume (PV) acts as a virtual storage instance that adds a volume to a cluster.

A persistent volume claim (PVC) is a request to provision persistent storage of a specific type and configuration, available in the following CodeReady Workspaces storage configuration strategies:

- Common
- Per-workspace
- Unique

The mounted PVC is displayed as a folder in a container file system.

10.3.2.2.1. Configuring a PVC strategy using the Operator

The following section describes how to configure workspace persistent volume claim (PVC) strategies of a CodeReady Workspaces server using the Operator.

WARNING

It is not recommended to reconfigure PVC strategies on an existing CodeReady Workspaces cluster with existing workspaces. Doing so causes data loss.

Operators are software extensions to OpenShift that use Custom Resources to manage applications and their components.

When deploying CodeReady Workspaces using the Operator, configure the intended strategy by modifying the **spec.storage.pvcStrategy** property of the CheCluster Custom Resource object YAML file.

Prerequisites

- A OpenShift orchestration tool, the OpenShift command-line tool, **oc**, is installed.

Procedure

The following procedure steps are available for:

- OpenShift command-line tool, **oc**

To do changes to the CheCluster YAML file, choose one of the following:

- Create a new cluster by executing the **oc apply** command. For example:

```
$ oc apply -f <my-cluster.yaml>
```

- Update the YAML file properties of an already running cluster by executing the **oc patch** command. For example:

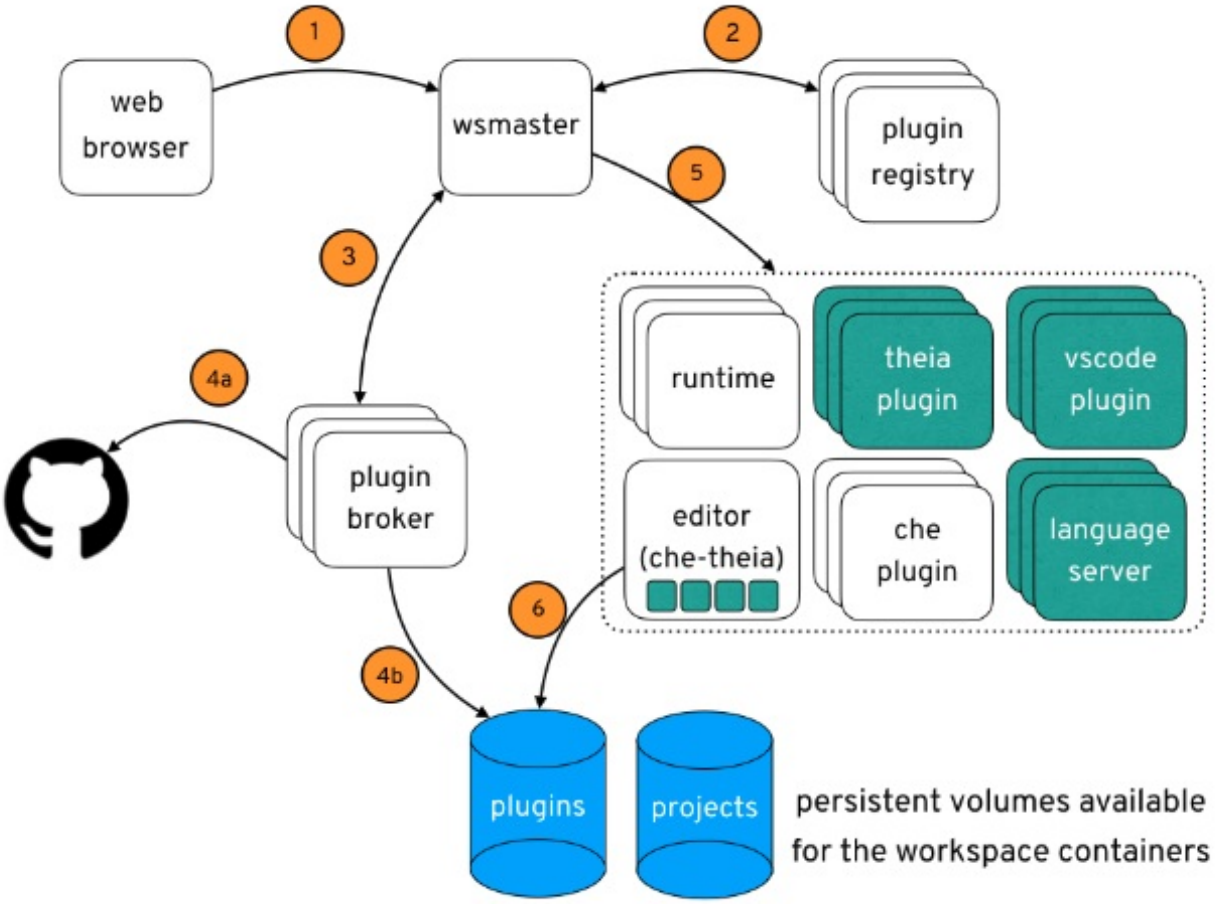
```
$ oc patch checluster codeready-workspaces --type=json \
  -p '[{"op": "replace", "path": "/spec/storage/pvcStrategy", "value": "<per-workspace>"}]'
```

Depending on the strategy used, replace the ***<per-workspace>*** option in the above example with **unique** or **common**.

10.3.2.3. Workspace projects configuration

The OpenShift project where a new workspace Pod is deployed depends on the CodeReady Workspaces server configuration. By default, every workspace is deployed in a distinct project, but the user can configure the CodeReady Workspaces server to deploy all workspaces in one specific project. The name of a project must be provided as a CodeReady Workspaces server configuration property and cannot be changed at runtime.

10.3.3. CodeReady Workspaces workspace creation flow



The following is a CodeReady Workspaces workspace creation flow:

1. A user starts a CodeReady Workspaces workspace defined by:
   - An editor (the default is Che-Theia)
   - A list of plug-ins (for example, Java and OpenShift tools)
   - A list of runtime applications
2. wsmaster retrieves the editor and plug-in metadata from the plug-in registry.
3. For every plug-in type, wsmaster 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 towsmaster in the format of a Che Plugin.
5. wsmaster starts the editor and the plug-in sidecars.
6. The editor loads the plug-ins from the plug-in persistent volume.