



# Red Hat CodeReady Workspaces 2.2

## End-user Guide

Using Red Hat CodeReady Workspaces 2.2



# Red Hat CodeReady Workspaces 2.2 End-user Guide

---

Using 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

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

Information for users using Red Hat CodeReady Workspaces.

## Table of Contents

<b>CHAPTER 1. NAVIGATING CODEREADY WORKSPACES USING THE DASHBOARD</b>	<b>6</b>
1.1. LOGGING IN TO CODEREADY WORKSPACES ON OPENSIFT FOR THE FIRST TIME USING OAUTH	6
1.2. LOGGING IN TO CODEREADY WORKSPACES ON OPENSIFT FOR THE FIRST TIME REGISTERING AS A NEW USER	6
1.3. FINDING CODEREADY WORKSPACES CLUSTER URL USING THE OPENSIFT 4 CLI	7
<b>CHAPTER 2. CHE-THEIA IDE BASICS</b>	<b>8</b>
2.1. DEFINING CUSTOM COMMANDS FOR CHE-THEIA	8
2.1.1. Che-Theia task types	9
2.1.2. Running and debugging	10
2.1.3. Editing a task and launch configuration	13
2.2. VERSION CONTROL	13
2.2.1. Managing Git configuration: identity	13
2.2.2. Accessing a Git repository using HTTPS	15
2.2.3. Accessing a Git repository using a generated SSH key pair	16
2.2.3.1. Generating an SSH key using the CodeReady Workspaces command palette	16
2.2.3.2. Adding the associated public key to a repository or account on GitHub	16
2.2.3.3. Adding the associated public key to a Git repository or account on GitLab	17
2.2.4. Managing pull requests using the GitHub PR plug-in	17
2.2.4.1. Using the GitHub Pull Requests plug-in	17
2.2.4.2. Creating a new pull request	17
2.3. CHE-THEIA TROUBLESHOOTING	18
<b>CHAPTER 3. WORKSPACES OVERVIEW</b>	<b>19</b>
3.1. CONFIGURING A WORKSPACE USING A DEVFILE	20
3.1.1. What is a devfile	20
3.1.2. Disambiguation between stacks and devfiles	21
3.1.3. Creating a workspace from the default branch of a Git repository	21
3.1.4. Creating a workspace from a feature branch of a Git repository	21
3.1.5. Creating a workspace from a publicly accessible standalone devfile using HTTP	22
3.1.6. Overriding devfile values using factory parameters	22
3.1.7. Creating a workspace using crwctl and a local devfile	25
3.2. MAKING A WORKSPACE PORTABLE USING A DEVFILE	25
3.2.1. What is a devfile	26
3.2.2. A minimal devfile	26
3.2.3. Generating workspace names	27
3.2.4. Writing a devfile for a project	27
3.2.4.1. Preparing a minimal devfile	27
3.2.4.2. Specifying multiple projects in a devfile	28
3.2.5. Devfile reference	29
3.2.5.1. Adding projects to a devfile	29
3.2.5.1.1. Project-source type: git	29
3.2.5.1.2. Project-source type: zip	30
3.2.5.1.3. Project clone-path parameter: clonePath	30
3.2.5.2. Adding components to a devfile	31
3.2.5.2.1. Component type: cheEditor	31
3.2.5.2.2. Component type: chePlugin	31
3.2.5.2.3. Specifying an alternative component registry	31
3.2.5.2.4. Specifying a component by linking to its descriptor	32
3.2.5.2.5. Tuning chePlugin component configuration	32
3.2.5.2.6. Component type: kubernetes	32
3.2.5.2.7. Overriding container entrypoints	33

3.2.5.2.8. Overriding container environment variables	34
3.2.5.2.9. Specifying mount-source option	34
3.2.5.2.10. Component type: dockerimage	34
3.2.5.2.10.1. Mounting project sources	35
3.2.5.2.10.2. Container Entrypoint	35
3.2.5.2.11. Persistent Storage	35
3.2.5.2.12. Specifying container memory limit for components	36
3.2.5.2.13. Specifying container memory request for components	37
3.2.5.2.14. Specifying container CPU limit for components	37
3.2.5.2.15. Specifying container CPU request for components	38
3.2.5.2.16. Environment variables	38
3.2.5.2.17. Endpoints	39
3.2.5.2.18. OpenShift resources	42
3.2.5.3. Adding commands to a devfile	45
3.2.5.3.1. CodeReady Workspaces-specific commands	45
3.2.5.3.2. Editor-specific commands	46
3.2.5.3.3. Command preview URL	47
3.2.5.3.3.1. Setting the default way of opening preview URLs	47
3.2.5.4. Devfile attributes	48
3.2.5.4.1. Attribute: editorFree	48
3.2.5.4.2. Attribute: persistVolumes (ephemeral mode)	48
3.2.6. Objects supported in Red Hat CodeReady Workspaces 2.2	48
3.3. CREATING AND CONFIGURING A NEW CODEREADY WORKSPACES 2.2 WORKSPACE	49
3.3.1. Creating a new workspace from the dashboard	49
3.3.2. Adding projects to your workspace	50
3.3.3. Configuring the workspace and adding tools	51
3.3.3.1. Adding plug-ins	51
3.3.3.2. Defining the workspace editor	52
3.3.3.3. Defining specific container images	53
3.3.3.4. Adding commands to your workspace	56
3.4. IMPORTING A OPENSIFT APPLICATION INTO A WORKSPACE	57
3.4.1. Including a OpenShift application in a workspace devfile definition	58
3.4.2. Adding a OpenShift application to an existing workspace using the dashboard	59
3.4.3. Generating a devfile from an existing OpenShift application	60
3.5. REMOTELY ACCESSING WORKSPACES	61
3.5.1. Remotely accessing workspaces using the OpenShift command-line tool	62
3.5.2. Downloading and uploading a file to a workspace using the command-line interface	63
3.6. CREATING A WORKSPACE FROM CODE SAMPLE	64
3.6.1. Creating a workspace from Get Started view of User Dashboard	64
3.6.2. Creating a workspace from Custom Workspace view of User Dashboard	66
3.6.3. Changing the configuration of an existing workspace	67
3.6.4. Running an existing workspace from the User Dashboard	69
3.6.4.1. Running an existing workspace from the User Dashboard with the Run button	69
3.6.4.2. Running an existing workspace from the User Dashboard using the Open button	69
3.6.4.3. Running an existing workspace from the User Dashboard using the Recent Workspaces	70
3.7. CREATING A WORKSPACE BY IMPORTING THE SOURCE CODE OF A PROJECT	71
3.7.1. Select a sample from the Dashboard, then change the devfile to include your project	72
3.7.2. Importing from the Dashboard into an existing workspace	72
3.7.2.1. Editing the commands after importing a project	73
3.7.3. Importing to a running workspace using the Git: Clone command	74
3.7.4. Importing to a running workspace with git clone in a terminal	75
3.8. CONFIGURING WORKSPACE EXPOSURE STRATEGIES	76
3.8.1. Workspace exposure strategies	76

3.8.1.1. Multi-host strategy	77
3.8.2. Security considerations	77
3.8.2.1. JSON web token (JWT) proxy	77
3.8.2.2. Secured plug-ins and editors	77
3.8.2.3. Secured container-image components	77
3.8.2.4. Cross-site request forgery attacks	78
3.8.2.5. Phishing attacks	78
3.9. MOUNTING A SECRET AS A FILE OR AN ENVIRONMENT VARIABLE INTO A WORKSPACE CONTAINER	78
3.9.1. Mounting a secret as a file into a workspace container	79
3.9.2. Mounting a secret as an environment variable into a workspace container	81
3.9.3. The use of annotations in the process of mounting a secret into a workspace container	83
<b>CHAPTER 4. CUSTOMIZING DEVELOPER ENVIRONMENTS</b>	<b>85</b>
4.1. WHAT IS A CHE-THEIA PLUG-IN	86
4.1.1. Features and benefits of Che-Theia plug-ins	86
4.1.2. Che-Theia plug-in concept in detail	86
4.1.2.1. Client-side and server-side Che-Theia plug-ins	87
4.1.2.2. Che-Theia plug-in APIs	87
4.1.2.3. Che-Theia plug-in capabilities	87
4.1.2.4. VS Code extensions and Eclipse Theia plug-ins	88
4.1.3. Che-Theia plug-in metadata	88
4.1.4. Che-Theia plug-in lifecycle	92
4.1.5. Embedded and remote Che-Theia plug-ins	94
4.1.5.1. Embedded (or local) plug-ins	94
4.1.5.2. Remote plug-ins	95
4.1.5.3. Comparison matrix	96
4.1.6. Remote plug-in endpoint	97
4.1.6.1. Defining a launch remote plug-in endpoint using Dockerfile	97
4.1.6.1.1. Using a wrapper script	98
4.1.6.2. Defining a launch remote plug-in endpoint in a meta.yaml file	99
4.2. USING ALTERNATIVE IDES IN CODEREADY WORKSPACES	100
4.3. USING A VISUAL STUDIO CODE EXTENSION IN CODEREADY WORKSPACES	101
4.3.1. Publishing a VS Code extension into the CodeReady Workspaces plug-in registry	101
4.3.1.1. Writing a meta.yaml file and adding it to a plug-in registry	101
4.3.2. Adding a plug-in registry VS Code extension to a workspace	103
4.3.2.1. Adding a VS Code extension using the CodeReady Workspaces Plugins panel	103
4.3.2.2. Adding a VS Code extension using the workspace configuration	104
4.3.3. Choosing the sidecar image	105
4.3.4. Verifying the VS Code extension API compatibility level	106
4.4. ADDING TOOLS TO CODEREADY WORKSPACES AFTER CREATING A WORKSPACE	107
4.4.1. Additional tools in the CodeReady Workspaces workspace	107
4.4.2. Adding language support plug-in to the CodeReady Workspaces workspace	107
<b>CHAPTER 5. CONFIGURING OAUTH AUTHORIZATION</b>	<b>110</b>
5.1. CONFIGURING GITHUB OAUTH	110
5.2. CONFIGURING OPENSIFT OAUTH	110
<b>CHAPTER 6. USING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT</b>	<b>112</b>
6.1. USING MAVEN ARTIFACT REPOSITORIES	112
6.1.1. Defining repositories in settings.xml	112
6.1.2. Defining Maven settings.xml file across workspaces	114
6.1.3. Using self-signed certificates in Java projects	115
6.2. USING GRADLE ARTIFACT REPOSITORIES	117

6.2.1. Downloading different versions of Gradle	117
6.2.2. Configuring global Gradle repositories	117
6.2.3. Using self-signed certificates in Java projects	118
6.3. USING PYTHON ARTIFACT REPOSITORIES	119
6.3.1. Configuring Python to use a non-standard registry	119
6.3.2. Using self-signed certificates in Python projects	119
6.4. USING GO ARTIFACT REPOSITORIES	120
6.4.1. Configuring Go to use a non-standard-registry	120
6.4.2. Using self-signed certificates in Go projects	121
6.5. USING NUGET ARTIFACT REPOSITORIES	121
6.5.1. Configuring NuGet to use a non-standard artifact repository	121
6.5.2. Using self-signed certificates in NuGet projects	122
6.6. USING NPM ARTIFACT REPOSITORIES	123
<b>CHAPTER 7. TROUBLESHOOTING FOR CODEREADY WORKSPACES END USERS</b> .....	<b>124</b>
7.1. RESTARTING A CODEREADY WORKSPACES WORKSPACE IN DEBUG MODE AFTER START FAILURE	124
7.2. STARTING A CODEREADY WORKSPACES WORKSPACE IN DEBUG MODE	125
<b>CHAPTER 8. OPENSIFT CONNECTOR OVERVIEW</b> .....	<b>127</b>
8.1. FEATURES OF OPENSIFT CONNECTOR	127
8.2. INSTALLING OPENSIFT CONNECTOR IN CODEREADY WORKSPACES	128
8.3. AUTHENTICATING WITH OPENSIFT CONNECTOR FROM CODEREADY WORKSPACES	128
8.4. CREATING COMPONENTS WITH OPENSIFT CONNECTOR IN CODEREADY WORKSPACES	130
8.5. CONNECTING SOURCE CODE FROM GITHUB TO AN OPENSIFT COMPONENT USING OPENSIFT CONNECTOR	131





# CHAPTER 1. NAVIGATING CODEREADY WORKSPACES USING THE DASHBOARD

The **Dashboard** is accessible on your cluster from a URL like **http://<che-instance>.<IP-address>.mycluster.mycompany.com/dashboard/**. This section describes how to access this URL on OpenShift.

## 1.1. LOGGING IN TO CODEREADY WORKSPACES ON OPENSHIFT FOR THE FIRST TIME USING OAUTH

This section describes how to log in to CodeReady Workspaces on OpenShift for the first time using OAuth.

### Prerequisites

- Contact the administrator of the OpenShift instance to obtain the **Red Hat CodeReady Workspaces URL**.

### Procedure

1. Navigate to the **Red Hat CodeReady Workspaces URL** to display the Red Hat CodeReady Workspaces login page.
2. Choose the **OpenShift OAuth** option.
3. The **Authorize Access** page is displayed.
4. Click on the **Allow selected permissions** button.
5. Update the account information: specify the **Username, Email, First name** and **Last name** fields and click the **Submit** button.

### Validation steps

- The browser displays the Red Hat CodeReady Workspaces **Dashboard**.

## 1.2. LOGGING IN TO CODEREADY WORKSPACES ON OPENSHIFT FOR THE FIRST TIME REGISTERING AS A NEW USER

This section describes how to log in to CodeReady Workspaces on OpenShift for the first time registering as a new user.

### Prerequisites

- Contact the administrator of the OpenShift instance to obtain the **Red Hat CodeReady Workspaces URL**.

### Procedure

1. Navigate to the **Red Hat CodeReady Workspaces URL** to display the Red Hat CodeReady Workspaces login page.
2. Choose the **Register as a new user** option.

3. Update the account information: specify the **Username**, **Email**, **First name** and **Last name** field and click the **Submit** button.

#### Validation steps

- The browser displays the Red Hat CodeReady Workspaces **Dashboard**.

## 1.3. FINDING CODEREADY WORKSPACES CLUSTER URL USING THE OPENSIFT 4 CLI

This section describes how to obtain the CodeReady Workspaces cluster URL using the OpenShift 4 CLI (command line interface). The URL can be retrieved from the OpenShift logs or from the **checluster** Custom Resource.

#### Prerequisites

- An instance of Red Hat CodeReady Workspaces running on OpenShift.
- User is located in a CodeReady Workspaces installation project.

#### Procedure

1. To retrieve the CodeReady Workspaces cluster URL from the **checluster** CR (Custom Resource), run:

```
$ oc get checluster --output jsonpath='{.items[0].status.cheURL}'
```

2. Alternatively, to retrieve the CodeReady Workspaces cluster URL from the OpenShift logs, run:

```
$ oc logs --tail=10 `(oc get pods -o name | grep operator)` | \
grep "available at" | \
awk -F'available at: ' '{print $2}' | sed 's/"//'
```

## CHAPTER 2. CHE-THEIA IDE BASICS

This section describes basics workflows and commands for Che-Theia: the native integrated development environment for Red Hat CodeReady Workspaces.

- [Defining custom commands for Che-Theia](#)
- [Version Control](#)
- [Troubleshooting](#)

### 2.1. DEFINING CUSTOM COMMANDS FOR CHE-THEIA

The Che-Theia IDE allows users to define custom commands in a devfile that are then available when working in a workspace.

The following is an example of the **commands** section of a devfile.

```
commands:
- name: theia:build
  actions:
  - type: exec
    component: che-dev
    command: >
      yarn
    workdir: /projects/theia
- name: run
  actions:
  - type: vscode-task
    referenceContent: |
      {
        "version": "2.0.0",
        "tasks":
        [
          {
            "label": "theia:watch",
            "type": "shell",
            "options": {"cwd": "/projects/theia"},
            "command": "yarn",
            "args": ["watch"]
          }
        ]
      }
- name: debug
  actions:
  - type: vscode-launch
    referenceContent: |
      {
        "version": "0.2.0",
        "configurations": [
          {
            "type": "node",
            "request": "attach",
            "name": "Attach by Process ID",
            "processId": "${command:PickProcess}"
          }
        ]
      }
```

```

|
}
]
}

```

## CodeReady Workspaces commands

### theia:build

- The **exec** type implies that the CodeReady Workspaces runner is used for command execution. The user can specify the component in whose container the command is executed.
- The **command** field contains the command line for execution.
- The **workdir** is the working directory in which the command is executed.

## Visual Studio Code (VS Code) tasks

### run

- The type is **vscode-task**.
- For this type of command, the **referenceContent** field must contain content with task configurations in the VS Code format.
- For more information about VS Code tasks, see the Task section on the [Visual Studio User Guide page](#).

## VS Code launch configurations

### debug

- The type is **vscode-launch**.
- It contains the launch configurations in the VS Code format.
- For more information about VS Code launch configurations, see the Debugging section on the [Visual Studio documentation page](#).

For a list of available tasks and launch configurations, see the **tasks.json** and the **launch.json** configuration files in the **/workspace/.theia** directory where the configuration from the devfile is exported to.

### 2.1.1. Che-Theia task types

Two types of tasks exist in a devfile: tasks in the VS Code format and CodeReady Workspaces commands. Tasks from the devfile are copied to the configuration file when the workspace is started. Depending on the type of the task, the task is then available for running:

- CodeReady Workspaces commands: From the **Terminal** → **Run Task** menu in the **configured tasks** section, or from the **My Workspace** panel
- Tasks in the VS Code format: From the **Run Tasks** menu

To run the task definitions provided by plug-ins, select the **Terminal** → **Run Task** menu option. The tasks are placed in the **detected tasks** section.

## 2.1.2. Running and debugging

Che-Theia supports the [Debug Adapter Protocol](#). This protocol defines a generic way for how a development tool can communicate with a debugger. It means Che-Theia works with all [implementations](#).

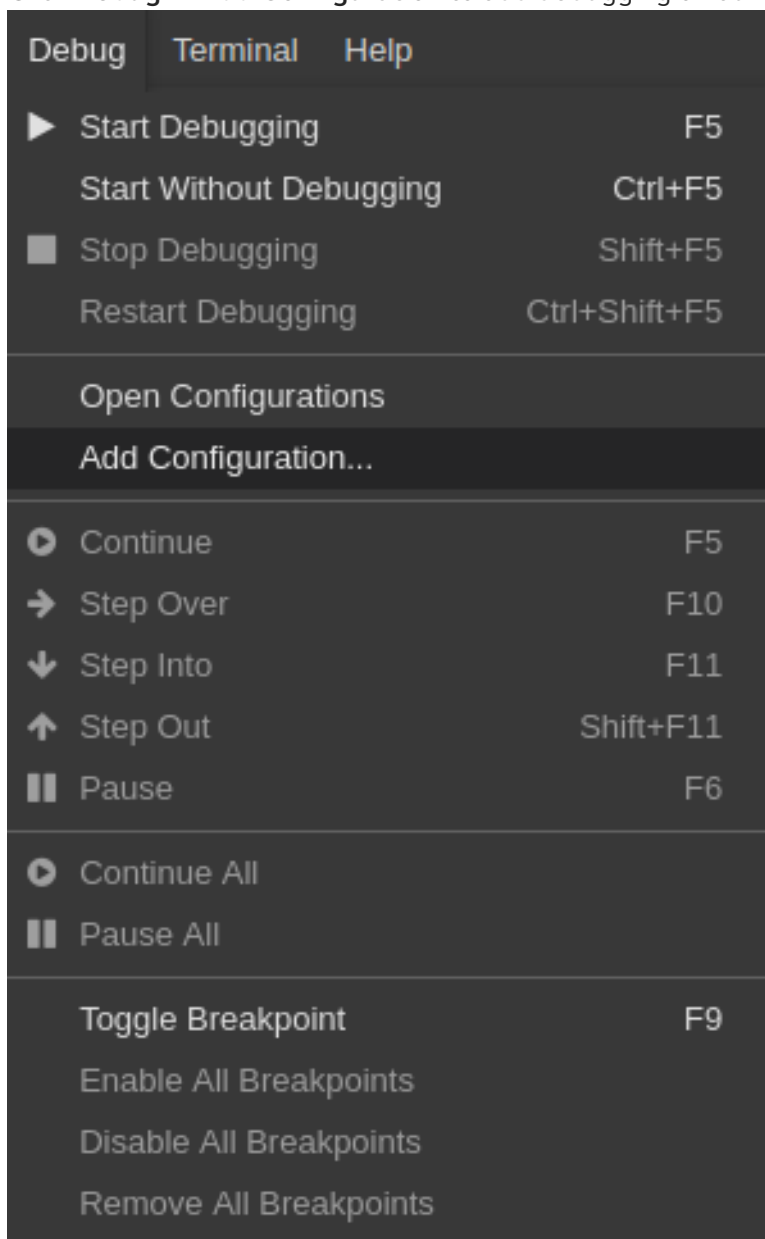
### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces 'quick-starts'](#).

### Procedure

To debug an application:

1. Click **Debug** → **Add Configuration** to add debugging or launch configuration to the project.



2. From the pop-up menu, select the appropriate configuration for the application that you want to debug.

```

launch.json
1  {
2    // Use IntelliSense to learn about possible attributes.
3    // Hover to view descriptions of existing attributes.
4    "version": "0.2.0",
5    "configurations": [
6
7    ]
8  }

```

- {} Java: Attach to Remote Program
- {} Java: Launch Program
- {} Java: Launch Program with Arguments Prompt
- {} Node.js: Attach
- {} Node.js: Attach to Process
- {} Node.js: Attach to Remote Program
- {} Node.js: Electron Main
- {} Node.js: Gulp task
- {} Node.js: Launch Program
- {} Node.js: Launch via NPM
- {} Node.js: Mocha Tests
- {} Node.js: Nodemon Setup

3. Update the configuration by modifying or adding attributes.

```

launch.json x
1  {
2    // Use IntelliSense to learn about possible attributes.
3    // Hover to view descriptions of existing attributes.
4    "version": "0.2.0",
5    "configurations": [
6      {
7        "type": "java",
8        "name": "Debug (Launch)",
9        "request": "launch",
10       "cwd": "${workspaceFolder}",
11       "console": "internalConsole",
12       "stopOnEntry": false,
13       "mainClass": "HelloWorld",
14       "args": ""
15     }
16   ]
17 }

```

4. Breakpoints can be toggled by clicking the editor margin.

```

HelloWorld.java x
1  /*
2     * HelloWorld.java
3     */
4  public class HelloWorld
5  {
6     public static void main(String[] args) {
7         System.out.println("Hello World!");
8     }
9 }

```

- Open the context menu of the breakpoint to add conditions.

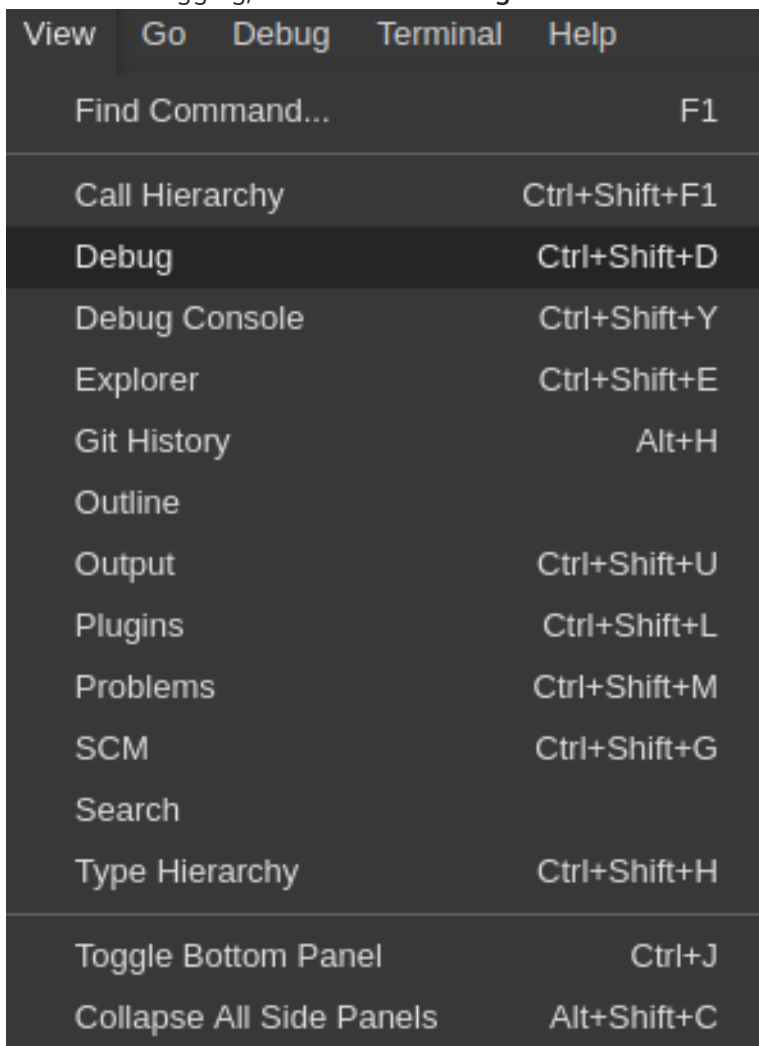
```

6     public static void main(String[] args) {
7         System.out.println("Hello World!");
}

```

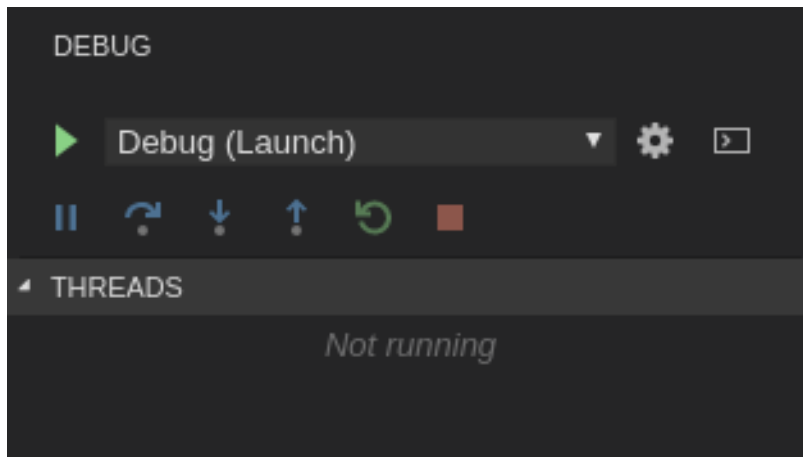
Expression  
Expression  
Hit Count  
Log Message

- To start debugging, click **View** → **Debug**.



- In the **Debug** view, select the configuration and press **F5** to debug the application. Or, start the application without debugging by pressing **Ctrl+F5**.



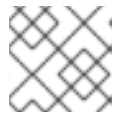


### 2.1.3. Editing a task and launch configuration

#### Procedure

To customize the configuration file:

1. Edit the **tasks.json** or **launch.json** configuration files.
2. Add new definitions to the configuration file or modify the existing ones.



#### NOTE

The changes are stored in the configuration file.

3. To customize the task configuration provided by plug-ins, select the **Terminal → Configure TasksS** menu option, and choose the task to configure. The configuration is then copied to the **tasks.json** file and is available for editing.

## 2.2. VERSION CONTROL

Red Hat CodeReady Workspaces natively supports the [VS Code SCM model](#). By default, Red Hat CodeReady Workspaces includes the native [VS Code Git extension](#) as a Source Code Management (SCM) provider.

### 2.2.1. Managing Git configuration: identity

The first thing to do before starting to use Git is to set a user name and email address. This is important because every Git commit uses this information.

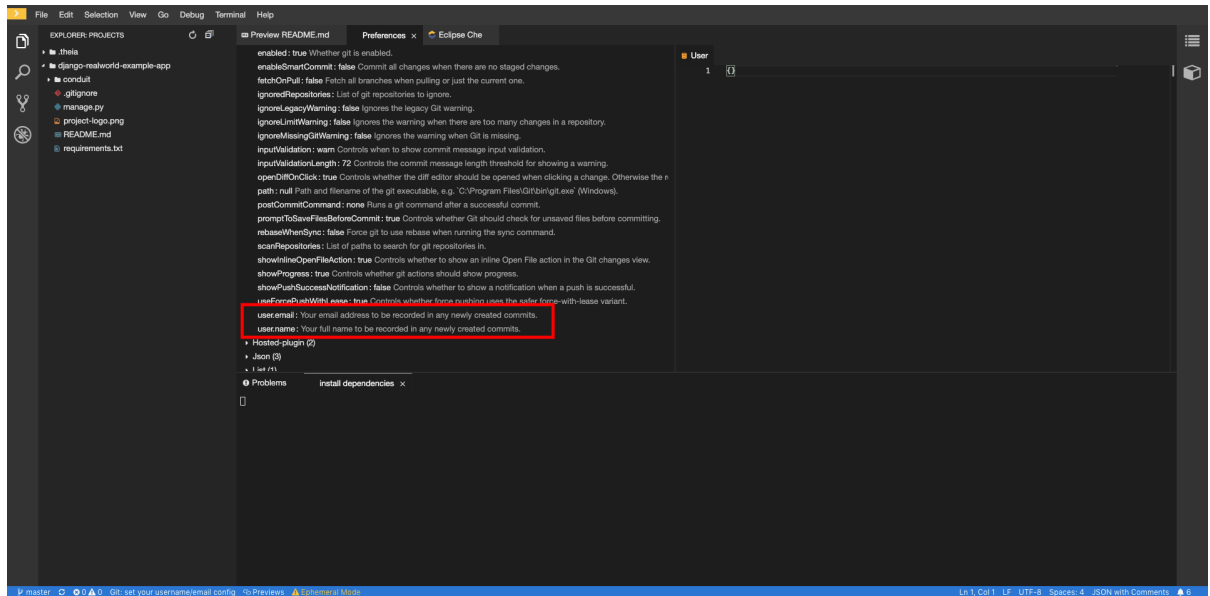
#### Prerequisites

- The Visual Studio Code **Git** extension installed.

#### Procedure

To configure Git identity using the CodeReady Workspaces user interface, go to in **Preferences**.

1. Open **File > Settings > Open Preferences**



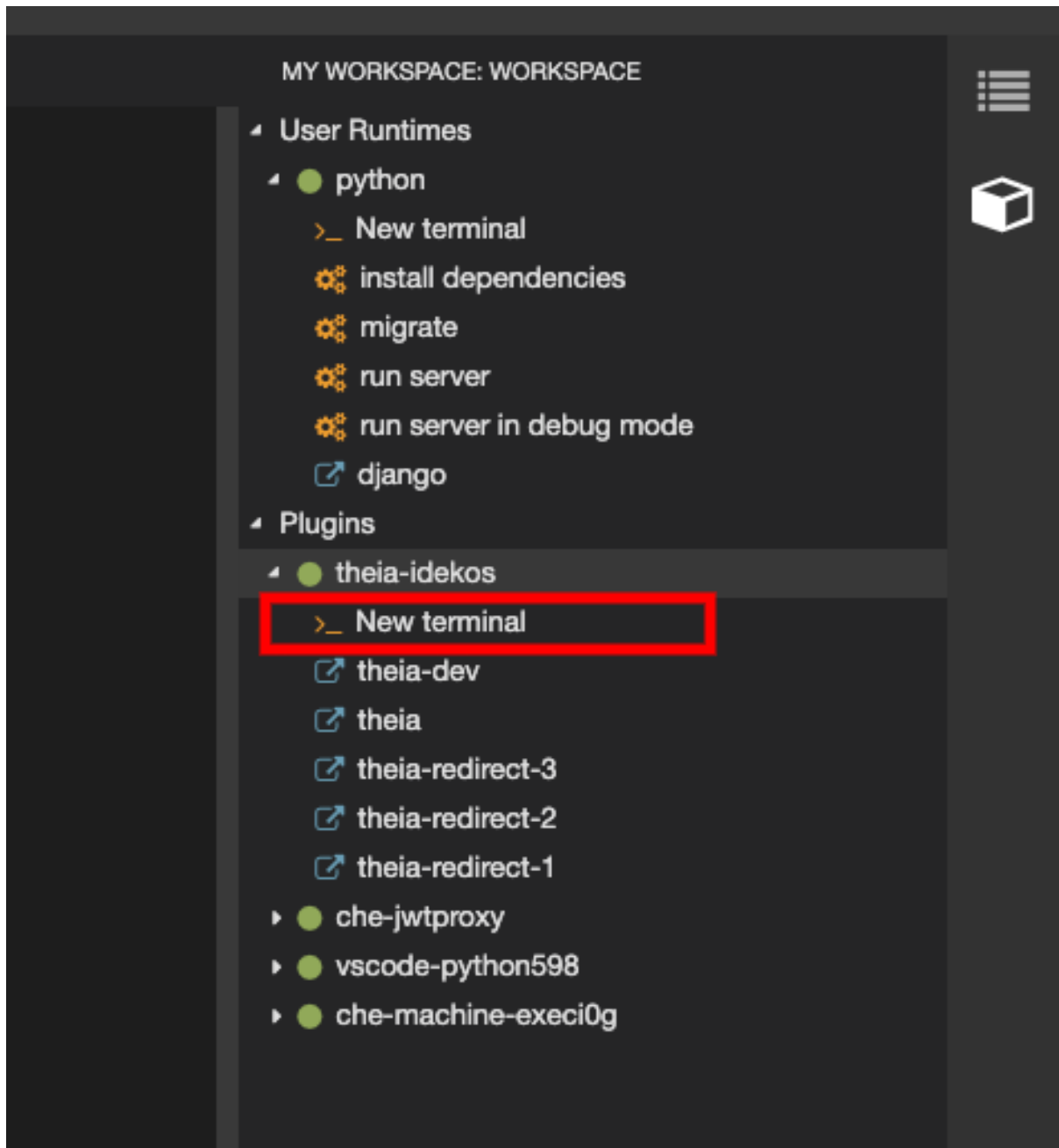
2. In the opened window, navigate to the **Git** section, and find:

```
user.name
user.email
```

And configure the identity.

To configure Git identity using the command line, open the terminal of the Che-Theia container.

1. Navigate to the **My Workspace** view, and open **Plugins > theia-ide... > New terminal**



2. Execute the following commands:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Che-Theia permanently stores this information and restores it on future workspace starts.

### 2.2.2. Accessing a Git repository using HTTPS

#### Prerequisites

- Git is installed. Install Git if needed by following [Getting Started - Installing Git](#).

#### Procedure

To clone a repository using HTTPS:

1. Use the `clone` command provided by the Visual Studio Code **Git** extension.

Alternatively, use the native Git commands in the terminal to clone a project.

1. Navigate to destination folder using the **cd** command.
2. Use **git clone** to clone a repository:

```
$ git clone <link>
```

Red Hat CodeReady Workspaces supports git self-signed SSL certificates. See [link:https://access.redhat.com/documentation/en-us/red\\_hat\\_codeready\\_workspaces/2.2/html/installation\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.2/html/installation_guide/) to learn more.

## 2.2.3. Accessing a Git repository using a generated SSH key pair

### 2.2.3.1. Generating an SSH key using the CodeReady Workspaces command palette

The following section describes a generation of an SSH key using the CodeReady Workspaces command palette and its further use in Git provider communication. This SSH key restricts permissions for the specific Git provider; therefore, the user has to create a unique SSH key for each Git provider in use.

#### Prerequisites

- A running instance of CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of CodeReady Workspaces [Creating a workspace from user dashboard](#).
- Personal [GitHub account](#) or other Git provider account created.

#### Procedure

A common SSH key pair that works with all the Git providers is present by default. To start using it, add the public key to the Git provider.

1. Generate an SSH key pair that only works with a particular Git provider:
  - In the CodeReady Workspaces IDE, press **F1** to open the Command Palette, or navigate to **View → Find Command** in the top menu. The **command palette** can be also activated by pressing **Ctrl+Shift+p** (or **Cmd+Shift+p** on macOS).
  - Search for **SSH: generate key pair for particular host** by entering **generate** into the search box and pressing **Enter** once filled.
  - Provide the hostname for the SSH key pair such as, for example, **github.com**. The SSH key pair is generated.
2. Click the **View** button and copy the public key from the editor and add it to the Git provider. Because of this action, the user can now use another command from the command palette: **Clone git repository** by providing an SSH secured URL.

### 2.2.3.2. Adding the associated public key to a repository or account on GitHub

To add the associated public key to a repository or account on GitHub:

1. Navigate to [github.com](https://github.com).
2. Click the drop-down arrow next to the user icon in the upper right corner of the window.
3. Click **Settings** → **SSH and GPG keys** and then click the **New SSH key** button.
4. In the **Title** field, type a title for the key, and in the **Key** field, paste the public key copied from CodeReady Workspaces.
5. Click the **Add SSH key** button.

### 2.2.3.3. Adding the associated public key to a Git repository or account on GitLab

To add the associated public key to a Git repository or account on GitLab:

1. Navigate to [gitlab.com](https://gitlab.com).
2. Click the user icon in the upper right corner of the window.
3. Click **Settings** → **SSH Keys**.
4. In the **Title** field, type a title for the key and in the **Key** field, paste the public key copied from CodeReady Workspaces.
5. Click the **Add key** button.

### 2.2.4. Managing pull requests using the GitHub PR plug-in

To manage GitHub pull requests, the VS Code GitHub Pull Request plug-in is available in the list of plug-ins of the workspace.

#### 2.2.4.1. Using the GitHub Pull Requests plug-in

##### Prerequisites

- GitHub OAuth is configured. See [Configuring GitHub OAuth](#).

##### Procedure

1. Authenticate by running the **GitHub authenticate** command.
2. You will be redirected to GitHub to authorize CodeReady Workspaces.
3. When CodeReady Workspaces is authorized, refresh the browser page where CodeReady Workspaces is running to update the plug-in with the GitHub token.

Alternatively, manually [fetch the GitHub token](#) and paste it to the plug-in by running the **GitHub Pull Requests: Manually Provide Authentication Response** command.

#### 2.2.4.2. Creating a new pull request

1. Open the GitHub repository. To be able to execute remote operations, the repository must have a *remote* with an SSH URL.

2. Checkout a new branch and make changes that you want to publish.
3. Run the **GitHub Pull Requests: Create Pull Request** command.

## 2.3. CHE-THEIA TROUBLESHOOTING

This section describes some of the most frequent issues with the Che-Theia IDE.

**Che-Theia shows a notification with the following message: Plugin runtime crashed unexpectedly, all plugins are not working, please reload the page. Probably there is not enough memory for the plugins.**

This means that one of the Che-Theia plug-ins that are running in the Che-Theia IDE container requires more memory than the container has. To fix this problem, increase the amount of memory for the Che-Theia IDE container:

1. Navigate to the CodeReady Workspaces Dashboard.
2. Select the workspace in which the problem happened.
3. Switch to the **Devfile** tab.
4. In the **components** section of the devfile, find a component of the **cheEditor** type.
5. Add a new property, **memoryLimit: 1024M** (or increase the value if it already exists).
6. Save changes and restart the workspace.

### Additional resources

- Asking the community for help: [Mattermost channel](#) dedicated to Red Hat CodeReady Workspaces.
- Reporting a bug: [Red Hat CodeReady Workspaces repository issues](#) .

## CHAPTER 3. WORKSPACES OVERVIEW

Red Hat CodeReady Workspaces provides developer workspaces with everything needed to a code, build, test, run, and debug applications. To allow that, the developer workspaces provide four main components:

1. The source code of a project.
2. A web-based IDE.
3. Tool dependencies, needed by developers to work on a project
4. Application runtime: a replica of the environment where the application runs in production

Pods manage each component of a CodeReady Workspaces workspace. Therefore, everything running in a CodeReady Workspaces workspace is running inside containers. This makes a CodeReady Workspaces workspace highly portable.

The embedded browser-based IDE is the point of access for everything running in a CodeReady Workspaces workspace. This makes a CodeReady Workspaces workspace easily shareable.



### IMPORTANT

By default, it is possible to run only one workspace at a time. To change the default value, see: [{link-limits-for-user-workspaces}](#).

Table 3.1. Features and benefits

Features	Traditional IDE workspaces	Red Hat CodeReady Workspaces workspaces
<b>Configuration and installation required</b>	Yes.	No.
<b>Embedded tools</b>	Partial. IDE plug-ins need configuration. Dependencies need installation and configuration. Example: JDK, Maven, Node.	Yes. Plug-ins provide their dependencies.
<b>Application runtime provided</b>	No. Developers have to manage that separately.	Yes. Application runtime is replicated in the workspace.
<b>Shareable</b>	No. Or not easily	Yes. Developer workspaces are shareable with a URL.
<b>Versionable</b>	No	Yes. Devfiles exist with project source code.
<b>Accessible from anywhere</b>	No. Installation is needed.	Yes. Only requires a browser.

To start a CodeReady Workspaces workspace, following options are available:

- [Creating and configuring a new workspace using the Dashboard](#)
- [Configuring a workspace using a devfile](#)

Use the Dashboard to discover CodeReady Workspaces 2.2:

- [Creating a workspace from code sample](#)
- [Creating a workspace by importing source code of a project](#)

Use a devfile as the preferred way to start a CodeReady Workspaces 2.2 workspace:

- [Making a workspace portable using a devfile](#)
- [Importing a OpenShift application into a workspace](#)

Use the browser-based IDE as the preferred way to interact with a CodeReady Workspaces 2.2 workspace. For an alternative way to interact with a CodeReady Workspaces 2.2 workspace, see: [Remotely accessing workspaces](#).

## 3.1. CONFIGURING A WORKSPACE USING A DEVFILE

To quickly and easily configure a CodeReady Workspaces workspace, use a devfile. For an introduction to devfiles and instructions for their use, see the instructions in this section.

### 3.1.1. What is a devfile

A devfile is a file that describes and define a development environment:

- the source code
- the development components (browser IDE tools and application runtimes)
- a list of pre-defined commands
- projects to clone

Devfiles are YAML files that CodeReady Workspaces consumes and transforms into a cloud workspace composed of multiple containers. The devfile can be saved in the root folder of a Git repository, a feature branch of a Git repository, a publicly accessible destination, or as a separate, locally stored artifact.

When creating a workspace, CodeReady Workspaces uses that definition to initiate everything and run all the containers for the required tools and application runtimes. CodeReady Workspaces also mounts file-system volumes to make source code available to the workspace.

Devfiles can be versioned with the project source code. When there is a need for a workspace to fix an old maintenance branch, the project devfile provides a definition of the workspace with the tools and the exact dependencies to start working on the old branch. Use it to instantiate workspaces on demand.

CodeReady Workspaces maintains the devfile up-to-date with the tools used in the workspace:

- Projects of the workspace (path, Git location, branch)
- Commands to perform daily tasks (build, run, test, debug)
- Runtime environment (container images to run the application)



- Che-Theia plug-ins with tools, IDE features, and helpers that a developer would use in the workspace (Git, Java support, SonarLint, Pull Request)

### 3.1.2. Disambiguation between stacks and devfiles

This section describes differences between stacks in CodeReady Workspaces 2.1 and devfiles in CodeReady Workspaces 2.2

Starting with CodeReady Workspaces 2.2:

- A stack is a pre-configured CodeReady Workspaces workspace.
- A devfile is a configuration YAML file that CodeReady Workspaces consumes and transforms in a cloud workspace composed of multiple containers.

In CodeReady Workspaces 2.1, stacks were defined by a **stacks.json** file that was included with the **che server**. In contrast, in CodeReady Workspaces 2.2, the **stacks.json** file does not exist. Instead, a stack is defined in the devfile registry, which is a separate service. Every single devfile in the registry corresponds to a stack.

Note that in CodeReady Workspaces 2.1, stacks and workspaces were defined using two different formats. However, with CodeReady Workspaces 2.2, the devfile format is used to define both the stacks and the workspaces. Nevertheless, a user opening the user dashboard does not notice any difference: in CodeReady Workspaces 2.2, a list of stacks is still present to choose from as a starting point to create a workspace.

### 3.1.3. Creating a workspace from the default branch of a Git repository

A CodeReady Workspaces workspace can be created by pointing to a devfile that is stored in a Git source repository. The CodeReady Workspaces instance then uses the discovered [devfile.yaml](#) file to build a workspace using the **/f?url=** API.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#)[CodeReady Workspaces quick-starts](#).
- The **devfile.yaml** file in the root folder of a Git repository available over HTTPS. See [Making a workspace portable using a devfile](#) for detailed information about creating and using devfiles.

#### Procedure

Run the workspace by opening the following URL: **https://codeready-<openshift\_deployment\_name>.<domain\_name>/f?url=https://<GitRepository>**

#### Example

```
https://che.openshift.io/f?url=https://github.com/eclipse/che
```

### 3.1.4. Creating a workspace from a feature branch of a Git repository

A CodeReady Workspaces workspace can be created by pointing to devfile that is stored in a Git source repository on a feature branch of the user's choice. The CodeReady Workspaces instance then uses the discovered devfile to build a workspace.

## Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#)[CodeReady Workspaces quick-starts](#).
- The **devfile.yaml** file in the root folder of a Git repository on a specific branch of the user's choice available over HTTPS. See [Making a workspace portable using a devfile](#) for detailed information about creating and using devfiles.

## Procedure

Execute the workspace by opening the following URL: **https://codeready-  
<openshift\_deployment\_name>.<domain\_name>/f?url=<GitHubBranch>**

## Example

Use following URL format to open an experimental [quarkus-quickstarts](#) branch hosted on [che.openshift.io](#).

```
https://che.openshift.io/f?url=https://github.com/maxandersen/quarkus-quickstarts/tree/che
```

### 3.1.5. Creating a workspace from a publicly accessible standalone devfile using HTTP

A workspace can be created using a devfile, the URL of which is pointing to the raw content of the devfile. The CodeReady Workspaces instance then uses the discovered devfile to build a workspace.

## Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#)[CodeReady Workspaces quick-starts](#).
- The publicly-accessible standalone **devfile.yaml** file. See [Making a workspace portable using a Devfile](#) for detailed information about creating and using devfiles.

## Procedure

1. Execute the workspace by opening the following URL: **{prod-fun}/f?url=https://<yourhosturl>/devfile.yaml**

## Example

```
https://che.openshift.io/f?url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c20743cde712bbc822521463359a60d1f7a9/devfile.yaml
```

### 3.1.6. Overriding devfile values using factory parameters

Values in the following sections of a remote devfile can be overridden using specially constructed additional factory parameters:

- **apiVersion**
- **metadata**

- **projects**
- **attributes**

### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#)[CodeReady Workspaces quick-starts](#).
- A publicly accessible standalone **devfile.yaml** file. See [Making a workspace portable using a Devfile](#) for detailed information about creating and using devfiles.

### Procedure

1. Open the workspace by navigating to the following URL: **`https://codeready-  
<openshift_deployment_name>.<domain_name>/f?  
url=https://<hostURL>/devfile.yaml&override.<parameter.path>=<value>`**

### Example of overriding the `generateName` property

Consider the following initial devfile:

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
projects:
...
```

To add or override **`generateName`** value, the following factory URL can be used:

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.metadata.generateName=myprefix
```

The resulting workspace will have the following devfile model:

```
---
apiVersion: 1.0.0
metadata:
  generateName: myprefix
projects:
...
```

### Example of overriding project source branch property

Consider the following initial devfile:

```
---
apiVersion: 1.0.0
metadata:
  generateName: java-mysql-
projects:
  - name: web-java-spring-petclinic
```

```

source:
  type: git
  location: "https://github.com/spring-projects/spring-petclinic.git"
...

```

To add or override source **branch** value, the following factory URL can be used:

```

https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.projects.web-java-spring-
petclinic.source.branch=1.0.x

```

The resulting workspace will have the following devfile model:

```

apiVersion: 1.0.0
metadata:
  generateName: java-mysql-
projects:
  - name: web-java-spring-petclinic
    source:
      type: git
      location: "https://github.com/spring-projects/spring-petclinic.git"
      branch: 1.0.x
...

```

### Example of overriding or creating an attribute value

Consider the following initial devfile:

```

---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
  persistVolumes: false
projects:
...

```

To add or override **persistVolumes** attribute value, the following factory URL can be used:

```

https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.attributes.persistVolumes=true

```

The resulting workspace will have the following devfile model:

```

---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
  persistVolumes: true
projects:
...

```

When overriding attributes, everything that follows the **attributes** keyword treat as an attribute name, so it's possible to use dot-separated names:

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.attributes.dot.name.format.attribute=true
```

The resulting workspace will have the following devfile model:

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
  dot.name.format.attribute: true
projects:
...
```

### 3.1.7. Creating a workspace using crwctl and a local devfile

A CodeReady Workspaces workspace can be created by pointing the **crwctl** tool to a locally stored devfile. The CodeReady Workspaces instance then uses the discovered devfile to build a workspace.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#).
- The CodeReady Workspaces CLI management tool. See [the CodeReady Workspaces 2.2 Installation Guide](#).
- The devfile is available on the local filesystem in the current working directory. See [Making a workspace portable using a Devfile](#) for detailed information about creating and using devfiles.

#### Example

Download the **devfile.yaml** file from the [GitHub repository](#) to the current working directory.

#### Procedure

1. Run a workspace from a devfile using the **workspace:start** parameter with the **crwctl** tool as follows:

```
$ crwctl workspace:start --devfile=devfile.yaml
```

#### Additional resources

- [Making a workspace portable using a Devfile](#)

## 3.2. MAKING A WORKSPACE PORTABLE USING A DEVFILE

To transfer a configured CodeReady Workspaces workspace, create and export the devfile of the workspace and load the devfile on a different host to initialize a new instance of the workspace. For detailed instructions on how to create such a devfile, see below.

### 3.2.1. What is a devfile

A devfile is a file that describes and define a development environment:

- the source code
- the development components (browser IDE tools and application runtimes)
- a list of pre-defined commands
- projects to clone

Devfiles are YAML files that CodeReady Workspaces consumes and transforms into a cloud workspace composed of multiple containers. The devfile can be saved in the root folder of a Git repository, a feature branch of a Git repository, a publicly accessible destination, or as a separate, locally stored artifact.

When creating a workspace, CodeReady Workspaces uses that definition to initiate everything and run all the containers for the required tools and application runtimes. CodeReady Workspaces also mounts file-system volumes to make source code available to the workspace.

Devfiles can be versioned with the project source code. When there is a need for a workspace to fix an old maintenance branch, the project devfile provides a definition of the workspace with the tools and the exact dependencies to start working on the old branch. Use it to instantiate workspaces on demand.

CodeReady Workspaces maintains the devfile up-to-date with the tools used in the workspace:

- Projects of the workspace (path, Git location, branch)
- Commands to perform daily tasks (build, run, test, debug)
- Runtime environment (container images to run the application)
- Che-Theia plug-ins with tools, IDE features, and helpers that a developer would use in the workspace (Git, Java support, SonarLint, Pull Request)

### 3.2.2. A minimal devfile

The following is the minimum content required in a **devfile.yaml** file:

- `apiVersion`
- `metadata name`

```
apiVersion: 1.0.0
metadata:
  name: che-in-che-out
```

For a complete devfile example, see [Red Hat CodeReady Workspaces in CodeReady Workspaces devfile.yaml](#).



## NAME OR GENERATE NAME MUST BE DEFINED

Both **name** and **generateName** are optional parameters, but at least one of them must be defined. See [Section 3.2.3, “Generating workspace names”](#).

### 3.2.3. Generating workspace names

To specify a prefix for automatically generated workspace names, set the **generateName** parameter in the **devfile.yaml** file:

```
apiVersion: 1.0.0
metadata:
  generateName: che-
```

The workspace name will be in the **<generateName>YYYYY** format (for example, **che-2y7kp**). **Y** is random **[a-z0-9]** character.

The following naming rules apply when creating workspaces:

- When **name** is defined, it is used as the workspace name: **<name>**
- When only **generateName** is defined, it is used as the base of the generated name: **<generateName>YYYYY**



## NOTE

For workspaces created using a factory, defining **name** or **generateName** has the same effect. The defined value is used as the name prefix: **<name>YYYYY** or **<generateName>YYYYY**. When both **generateName** and **name** are defined, **generateName** takes precedence.

### 3.2.4. Writing a devfile for a project

This section describes how to create a minimal devfile for your project and how to include more than one projects in a devfile.

#### 3.2.4.1. Preparing a minimal devfile

A minimal devfile sufficient to run a workspace consists of the following parts:

- Specification version
- Name

#### Example of a minimal devfile with no project

```
apiVersion: 1.0.0
metadata:
  name: minimal-workspace
```

Without any further configuration, a workspace with the default editor is launched along with its default plug-ins, which are configured on the CodeReady Workspaces Server. Che-Theia is configured as the default editor along with the **CodeReady Workspaces Machine Execplug-in**. When launching a

workspace within a Git repository using a factory, the project from the given repository and branch is be created by default. The project name then matches the repository name.

Add the following parts for a more functional workspace:

- List of components: Development components and user runtimes
- List of projects: Source code repositories
- List of commands: Actions to manage the workspace components, such as running the development tools, starting the runtime environments, and others

### Example of a minimal devfile with a project

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/spring-projects/spring-petclinic.git'
components:
  - type: chePlugin
    id: redhat/java/latest
```

#### 3.2.4.2. Specifying multiple projects in a devfile

A single devfile can specify multiple projects. For each project, specify the type of the source repository, its location, and, optionally, the directory the project is cloned to.

### Example of a devfile with two projects

```
apiVersion: 1.0.0
metadata:
  name: example-devfile
projects:
  - name: frontend
    source:
      type: git
      location: https://github.com/acmecorp/frontend.git
  - name: backend
    clonePath: src/github.com/acmecorp/backend
    source:
      type: git
      location: https://github.com/acmecorp/backend.git
```

In the preceding example, there are two projects defined, **frontend** and **backend**. Each project is located in its own repository. The **backend** project has a specific requirement to be cloned into the **src/github.com/acmecorp/backend/** directory under the source root (implicitly defined by the CodeReady Workspaces runtime) while the **frontend** project will be cloned into the **frontend/** directory under the source root.

### Additional resources



For a detailed explanation of all devfile component assignments and possible values, see:

- [Specification repository](#)
- [Detailed json-schema documentation](#)

These sample devfiles are a good source of inspiration:

- [Sample devfiles for Red Hat CodeReady Workspaces workspaces used by default in the user interface.](#)
- [Sample devfiles for Red Hat CodeReady Workspaces workspaces from Red Hat Developer program.](#)

### 3.2.5. Devfile reference

This section contains devfile reference and instructions on how to use the various elements that devfiles consist of.

#### 3.2.5.1. Adding projects to a devfile

Usually a devfile contains one or more projects. A workspace is created to develop those projects. Projects are added in the **projects** section of devfiles.

Each project in a single devfile must have:

- Unique name
- Source specified

Project source consists of two mandatory values: **type** and **location**.

##### **type**

The kind of project-source provider.

##### **location**

The URL of project source.

CodeReady Workspaces supports the following project types:

##### **git**

Projects with sources in Git. The location points to a clone link.

##### **github**

Same as **git** but for projects hosted on [GitHub](#) only. Use **git** for projects that do not use GitHub-specific features.

##### **zip**

Projects with sources in a ZIP archive. Location points to a ZIP file.

#### 3.2.5.1.1. Project-source type: git

```
source:
  type: git
  location: https://github.com/eclipse/che.git
  startPoint: master
```

1

```

tag: 7.2.0
commitId: 36fe587
branch: master
sparseCheckoutDir: wsmaster 2

```

- 1 **startPoint** is the general value for **tag**, **commitId**, and **branch**. The **startPoint**, **tag**, **commitId**, and **branch** parameters are mutually exclusive. When more than one is supplied, the following order is used: **startPoint**, **tag**, **commitId**, **branch**.
- 2 **sparseCheckoutDir** the template for the sparse checkout Git feature. This is useful when only a part of a project (typically only a single directory) is needed.

### Example 3.1. sparseCheckoutDir parameter settings

- Set to **/my-module/** to create only the root **my-module** directory (and its content).
- Omit the leading slash (**my-module/**) to create all **my-module** directories that exist in the project. Including, for example, **/addons/my-module/**.  
The trailing slash indicates that only directories with the given name (including their content) are created.
- Use wildcards to specify more than one directory name. For example, setting **module-\*** checks out all directories of the given project that start with **module-**.

For more information, see [Sparse checkout in Git documentation](#).

#### 3.2.5.1.2. Project-source type: zip

```

source:
  type: zip
  location: http://host.net/path/project-src.zip

```

#### 3.2.5.1.3. Project clone-path parameter: clonePath

The **clonePath** parameter specifies the path into which the project is to be cloned. The path must be relative to the **/projects/** directory, and it cannot leave the **/projects/** directory. The default value is the project name.

### Example devfile with projects

```

apiVersion: 1.0.0
metadata:
  name: my-project-dev
projects:
  - name: my-project-resource
    clonePath: resources/my-project
    source:
      type: zip
      location: http://host.net/path/project-res.zip
  - name: my-project
    source:

```

```

type: git
location: https://github.com/my-org/project.git
branch: develop

```

### 3.2.5.2. Adding components to a devfile

Each component in a single devfile must have a unique name.

#### 3.2.5.2.1. Component type: `cheEditor`

Describes the editor used in the workspace by defining its **id**. A devfile can only contain one component of the **cheEditor** type.

```

components:
- alias: theia-editor
  type: cheEditor
  id: eclipse/che-theia/next

```

When **cheEditor** is missing, a default editor is provided along with its default plug-ins. The default plug-ins are also provided for an explicitly defined editor with the same **id** as the default one (even if it is a different version). Che-Theia is configured as default editor along with the **CodeReady Workspaces Machine Exec** plug-in.

To specify that a workspace requires no editor, use the **editorFree:true** attribute in the devfile attributes.

#### 3.2.5.2.2. Component type: `chePlugin`

Describes plug-ins in a workspace by defining their **id**. It is allowed to have several **chePlugin** components.

```

components:
- alias: exec-plugin
  type: chePlugin
  id: eclipse/che-machine-exec-plugin/0.0.1

```

Both types above use an ID, which is slash-separated publisher, name and version of plug-in from the CodeReady Workspaces Plug-in registry.

List of available CodeReady Workspaces plug-ins and more information about registry can be found in the [CodeReady Workspaces plug-in registry](#) GitHub repository.

#### 3.2.5.2.3. Specifying an alternative component registry

To specify an alternative registry for the **cheEditor** and **chePlugin** component types, use the **registryUrl** parameter:

```

components:
- alias: exec-plugin
  type: chePlugin
  registryUrl: https://my-customregistry.com
  id: eclipse/che-machine-exec-plugin/0.0.1

```

### 3.2.5.2.4. Specifying a component by linking to its descriptor

An alternative way of specifying **cheEditor** or **chePlugin**, instead of using the editor or plug-in **id** (and optionally an alternative registry), is to provide a direct link to the component descriptor (typically named **meta.yaml**) by using the **reference** field:

```
components:
- alias: exec-plugin
  type: chePlugin
  reference: https://raw.githubusercontent.com.../plugin/1.0.1/meta.yaml
```



#### NOTE

It is impossible to mix the **id** and **reference** fields in a single component definition; they are mutually exclusive.

### 3.2.5.2.5. Tuning chePlugin component configuration

A chePlugin component may need to be precisely tuned, and in such case, component preferences can be used. The example shows how to configure JVM using plug-in preferences.

```
id: redhat/java/0.38.0
type: chePlugin
preferences:
  java.jdt.ls.vmargs: '-noverify -Xmx1G -XX:+UseG1GC -XX:+UseStringDeduplication'
```

Preferences may also be specified as an array:

```
id: redhat/java/0.38.0
type: chePlugin
preferences:
  go.lintFlags: ['--enable-all', "--new"]
```

### 3.2.5.2.6. Component type: kubernetes

A complex component type that allows to apply configuration from a list of OpenShift components.

The content can be provided through the **reference** attribute, which points to the file with the component content.

```
components:
- alias: mysql
  type: kubernetes
  reference: petclinic.yaml
  selector:
    app.kubernetes.io/name: mysql
    app.kubernetes.io/component: database
    app.kubernetes.io/part-of: petclinic
```

Alternatively, to post a devfile with such components to REST API, the contents of the OpenShift list can be embedded into the devfile using the **referenceContent** field:

```
components:
```

```

- alias: mysql
  type: kubernetes
  reference: petclinic.yaml
  referenceContent: |
    kind: List
    items:
    -
      apiVersion: v1
      kind: Pod
      metadata:
        name: ws
      spec:
        containers:
        ... etc

```

### 3.2.5.2.7. Overriding container endpoints

As with the understood by OpenShift).

There can be more containers in the list (contained in Pods or Pod templates of deployments). To select which containers to apply the endpoint changes to.

The endpoints can be defined as follows:

```

components:
- alias: appDeployment
  type: kubernetes
  reference: app-deployment.yaml
  endpoints:
  - parentName: mysqlServer
    command: ['sleep']
    args: ['infinity']
  - parentSelector:
      app: prometheus
    args: ['-f', '/opt/app/prometheus-config.yaml']

```

The **endpoints** list contains constraints for picking the containers along with the **command** and **args** parameters to apply to them. In the example above, the constraint is **parentName: mysqlServer**, which will cause the command to be applied to all containers defined in any parent object called **mysqlServer**. The parent object is assumed to be a top level object in the list defined in the referenced file, which is **app-deployment.yaml** in the example above.

Other types of constraints (and their combinations) are possible:

#### **containerName**

the name of the container

#### **parentName**

the name of the parent object that (indirectly) contains the containers to override

#### **parentSelector**

the set of labels the parent object needs to have

A combination of these constraints can be used to precisely locate the containers inside the referenced OpenShift list.

### 3.2.5.2.8. Overriding container environment variables

To provision or override entrypoints in a OpenShift or Openshift component, configure it in the following way:

```
components:
  - alias: appDeployment
    type: kubernetes
    reference: app-deployment.yaml
    env:
      - name: ENV_VAR
        value: value
```

This is useful for temporary content or without access to editing the referenced content. The specified environment variables are provisioned into each init container and containers inside all Pods and Deployments.

### 3.2.5.2.9. Specifying mount-source option

To specify a project sources directory mount into container(s), use the **mountSources** parameter:

```
components:
  - alias: appDeployment
    type: kubernetes
    reference: app-deployment.yaml
    mountSources: true
```

If enabled, project sources mounts will be applied to every container of the given component. This parameter is also applicable for **chePlugin** type components.

### 3.2.5.2.10. Component type: dockerimage

A component type that allows to define a container image-based configuration of a container in a workspace. A devfile can only contain one component of the **dockerimage** type. The **dockerimage** type of component brings in custom tools into the workspace. The component is identified by its image.

```
components:
  - alias: maven
    type: dockerimage
    image: eclipse/maven-jdk8:latest
    volumes:
      - name: mavenrepo
        containerPath: /root/.m2
    env:
      - name: ENV_VAR
        value: value
    endpoints:
      - name: maven-server
        port: 3101
        attributes:
          protocol: http
          secure: 'true'
          public: 'true'
          discoverable: 'false'
```

```
memoryLimit: 1536M
command: ['tail']
args: ['-f', '/dev/null']
```

### Example of a `minimaldockerimage` component

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  type: dockerimage
  image: golang
  memoryLimit: 512Mi
  command: ['sleep', 'infinity']
```

It specifies the type of the component, **dockerimage** and the **image** attribute names the image to be used for the component using the usual Docker naming conventions, that is, the above **type** attribute is equal to **docker.io/library/golang:latest**.

A **dockerimage** component has many features that enable augmenting the image with additional resources and information needed for meaningful integration of the **tool** provided by the image with Red Hat CodeReady Workspaces.

#### 3.2.5.2.10.1. Mounting project sources

For the **dockerimage** component to have access to the project sources, you must set the **mountSources** attribute to **true**.

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  type: dockerimage
  image: golang
  memoryLimit: 512Mi
  mountSources: true
  command: ['sleep', 'infinity']
```

The sources is mounted on a location stored in the **CHE\_PROJECTS\_ROOT** environment variable that is made available in the running container of the image. This location defaults to **/projects**.

#### 3.2.5.2.10.2. Container Entrypoint

The **command** attribute of the **dockerimage** along with other arguments, is used to modify the **entrypoint** command of the container created from the image. In Red Hat CodeReady Workspaces the container is needed to run indefinitely so that you can connect to it and execute arbitrary commands in it at any time. Because the availability of the **sleep** command and the support for the **infinity** argument for it is different and depends on the base image used in the particular images, CodeReady Workspaces cannot insert this behavior automatically on its own. However, you can take advantage of this feature to, for example, start necessary servers with modified configurations, etc.

#### 3.2.5.2.11. Persistent Storage

Components of any type can specify the custom volumes to be mounted on specific locations within the image. Note that the volume names are shared across all components and therefore this mechanism can also be used to share file systems between components.

Example specifying volumes for **dockerimage** type:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: dockerimage
  image: golang
  memoryLimit: 512Mi
  mountSources: true
  command: ['sleep', 'infinity']
  volumes:
  - name: cache
    containerPath: /.cache
```

Example specifying volumes for **cheEditor/chePlugin** type:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: cheEditor
  alias: theia-editor
  id: eclipse/che-theia/next
  env:
  - name: HOME
    value: $(CHE_PROJECTS_ROOT)
  volumes:
  - name: cache
    containerPath: /.cache
```

Example specifying volumes for **kubernetes/openshift** type:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: openshift
  alias: mongo
  reference: mongo-db.yaml
  volumes:
  - name: mongo-persistent-storage
    containerPath: /data/db
```

### 3.2.5.2.12. Specifying container memory limit for components

To specify a container(s) memory limit for **dockerimage**, **chePlugin**, **cheEditor**, use the **memoryLimit** parameter:

```
components:
```



```

- alias: exec-plugin
  type: chePlugin
  id: eclipse/che-machine-exec-plugin/0.0.1
  memoryLimit: 1Gi
- type: dockerimage
  image: eclipse/maven-jdk8:latest
  memoryLimit: 512M

```

This limit will be applied to every container of the given component.

For the **cheEditor** and **chePlugin** component types, RAM limits can be described in the plug-in descriptor file, typically named **meta.yaml**.

If none of them are specified, system-wide defaults will be applied (see description of **CHE\_WORKSPACE\_SIDE CAR\_DEFAULT\_MEMORY\_LIMIT\_MB** system property).

### 3.2.5.2.13. Specifying container memory request for components

To specify a container(s) memory request for **chePlugin** or **cheEditor**, use the **memoryRequest** parameter:

```

components:
- alias: exec-plugin
  type: chePlugin
  id: eclipse/che-machine-exec-plugin/0.0.1
  memoryLimit: 1Gi
  memoryRequest: 512M
- type: dockerimage
  image: eclipse/maven-jdk8:latest
  memoryLimit: 512M
  memoryRequest: 256M

```

This limit will be applied to every container of the given component.

For the **cheEditor** and **chePlugin** component types, RAM requests can be described in the plug-in descriptor file, typically named **meta.yaml**.

If none of them are specified, system-wide defaults are applied (see description of **CHE\_WORKSPACE\_SIDE CAR\_DEFAULT\_MEMORY\_REQUEST\_MB** system property).

### 3.2.5.2.14. Specifying container CPU limit for components

To specify a container(s) CPU limit for **chePlugin**, **cheEditor** or **dockerimage** use the **cpuLimit** parameter:

```

components:
- alias: exec-plugin
  type: chePlugin
  id: eclipse/che-machine-exec-plugin/0.0.1
  cpuLimit: 1.5
- type: dockerimage
  image: eclipse/maven-jdk8:latest
  cpuLimit: 750m

```

This limit will be applied to every container of the given component.

For the **cheEditor** and **chePlugin** component types, CPU limits can be described in the plug-in descriptor file, typically named **meta.yaml**.

If none of them are specified, system-wide defaults are applied (see description of **CHE\_WORKSPACE\_SIDECAR\_DEFAULT\_CPU\_LIMIT\_CORES** system property).

### 3.2.5.2.15. Specifying container CPU request for components

To specify a container(s) CPU request for **chePlugin**, **cheEditor** or **dockerimage** use the **cpuRequest** parameter:

```
components:
- alias: exec-plugin
  type: chePlugin
  id: eclipse/che-machine-exec-plugin/0.0.1
  cpuLimit: 1.5
  cpuRequest: 0.225
- type: dockerimage
  image: eclipse/maven-jdk8:latest
  cpuLimit: 750m
  cpuRequest: 450m
```

This limit will be applied to every container of the given component.

For the **cheEditor** and **chePlugin** component types, CPU requests can be described in the plug-in descriptor file, typically named **meta.yaml**.

If none of them are specified, system-wide defaults are applied (see description of **CHE\_WORKSPACE\_SIDECAR\_DEFAULT\_CPU\_REQUEST\_CORES** system property).

### 3.2.5.2.16. Environment variables

Red Hat CodeReady Workspaces allows you to configure Docker containers by modifying the environment variables available in component's configuration. Environment variables are supported by the following component types: **dockerimage**, **chePlugin**, **cheEditor**, **kubernetes**, **openshift**. In case component has multiple containers, environment variables will be provisioned to each container.

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: dockerimage
  image: golang
  memoryLimit: 512Mi
  mountSources: true
  command: ['sleep', 'infinity']
  env:
  - name: GOPATH
    value: $(CHE_PROJECTS_ROOT)/go
- type: cheEditor
  alias: theia-editor
  id: eclipse/che-theia/next
  memoryLimit: 2Gi
```

```
env:
- name: HOME
  value: $(CHE_PROJECTS_ROOT)
```



## NOTE

- The variable expansion works between the environment variables, and it uses the OpenShift convention for the variable references.
- The predefined variables are available for use in custom definitions.

The following environment variables are pre-set by the CodeReady Workspaces server:

- **CHE\_PROJECTS\_ROOT**: The location of the projects directory (note that if the component does not mount the sources, the projects will not be accessible).
- **CHE\_WORKSPACE\_LOGS\_ROOT\_\_DIR**: The location of the logs common to all the components. If the component chooses to put logs into this directory, the log files are accessible from all other components.
- **CHE\_API\_INTERNAL**: The URL to the CodeReady Workspaces server API endpoint used for communication with the CodeReady Workspaces server.
- **CHE\_WORKSPACE\_ID**: The ID of the current workspace.
- **CHE\_WORKSPACE\_NAME**: The name of the current workspace.
- **CHE\_WORKSPACE\_NAMESPACE**: The CodeReady Workspaces project of the current workspace. This environment variable is the name of the user or organization that the workspace belongs to. Note that this is different from the OpenShift project or OpenShift project to which the workspace is deployed.
- **CHE\_MACHINE\_TOKEN**: The token used to authenticate the request against the CodeReady Workspaces server.
- **CHE\_MACHINE\_AUTH\_SIGNATUREPUBLICKEY**: The public key used to secure the communication with the CodeReady Workspaces server.
- **CHE\_MACHINE\_AUTH\_SIGNATURE\_\_ALGORITHM**: The encryption algorithm used in the secured communication with the CodeReady Workspaces server.

A devfiles may only need the **CHE\_PROJECTS\_ROOT** environment variable to locate the cloned projects in the component's container. More advanced devfiles might use the **CHE\_WORKSPACE\_LOGS\_ROOT\_\_DIR** environment variable to read the logs (for example as part of a devfile command). The environment variables used to securely access the CodeReady Workspaces server are mostly out of scope for devfiles and are present only for advanced use cases that are usually handled by the CodeReady Workspaces plug-ins.

### 3.2.5.2.17. Endpoints

Components of any type can specify the endpoints that the Docker image exposes. These endpoints can be made accessible to the users if the CodeReady Workspaces cluster is running using a OpenShift ingress or an OpenShift route and to the other components within the workspace. You can create an endpoint for your application or database, if your application or database server is listening on a port and you want to be able to directly interact with it yourself or you want other components to interact with it.

Endpoints have several properties as shown in the following example:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  - type: dockerimage
    image: golang
    memoryLimit: 512Mi
    mountSources: true
    command: ['sleep', 'infinity']
    env:
      - name: GOPATH
        value: $(CHE_PROJECTS_ROOT)/go
      - name: GOCACHE
        value: /tmp/go-cache
    endpoints:
      - name: web
        port: 8080
        attributes:
          discoverable: false
          public: true
          protocol: http
      - type: dockerimage
        image: postgres
        memoryLimit: 512Mi
        env:
          - name: POSTGRES_USER
            value: user
          - name: POSTGRES_PASSWORD
            value: password
          - name: POSTGRES_DB
            value: database
        endpoints:
          - name: postgres
            port: 5432
            attributes:
              discoverable: true
              public: false
```

Here, there are two Docker images, each defining a single endpoint. Endpoint is an accessible port that can be made accessible inside the workspace or also publicly (example, from the UI). Each endpoint has a name and port, which is the port on which certain server running inside the container is listening. The following are a few attributes that you can set on the endpoint:

- **discoverable:** If an endpoint is discoverable, it means that it can be accessed using its name as the host name within the workspace containers (in the OpenShift parlance, a service is created for it with the provided name).

- **public:** The endpoint will be accessible outside of the workspace, too (such endpoint can be accessed from the CodeReady Workspaces user interface). Such endpoints are publicized always on port **80** or **443** (depending on whether **tls** is enabled in CodeReady Workspaces).
- **protocol:** For public endpoints the protocol is a hint to the UI on how to construct the URL for the endpoint access. Typical values are **http**, **https**, **ws**, **wss**.
- **secure:** A boolean (defaulting to **false**) specifying whether the endpoint is put behind a JWT proxy requiring a JWT workspace token to grant access. The JWT proxy is deployed in the same Pod as the server and assumes the server listens solely on the local loopback interface, such as **127.0.0.1**.



### WARNING

Listening on any other interface than the local loopback poses a security risk because such server is accessible without the JWT authentication within the cluster network on the corresponding IP addresses.

- **path:** The URL of the endpoint.
- **unsecuredPaths:** A comma-separated list of endpoint paths that are to stay unsecured even if the **secure** attribute is set to **true**.
- **cookiesAuthEnabled:** When set to **true** (the default is **false**), the JWT workspace token is automatically fetched and included in a workspace-specific cookie to allow requests to pass through the JWT proxy.



### WARNING

This setting potentially allows a [CSRF](#) attack when used in conjunction with a server using POST requests.

When starting a new server within a component, CodeReady Workspaces autodetects this, and the UI offers to automatically expose this port as a **public** port. This is useful for debugging a web application, for example. It is impossible to do this for servers that autostart with the container (for example, a database server). For such components, specify the endpoints explicitly.

Example specifying endpoints for **kubernetes/openshift** and **chePlugin/cheEditor** types:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: cheEditor
    alias: theia-editor
    id: eclipse/che-theia/next
```

```
endpoints:  
- name: 'theia-extra-endpoint'  
  port: 8880  
  attributes:  
    discoverable: true  
    public: true  
  
- type: chePlugin  
  id: redhat/php/latest  
  memoryLimit: 1Gi  
  endpoints:  
  - name: 'php-endpoint'  
    port: 7777  
  
- type: chePlugin  
  alias: theia-editor  
  id: eclipse/che-theia/next  
  endpoints:  
  - name: 'theia-extra-endpoint'  
    port: 8880  
    attributes:  
      discoverable: true  
      public: true  
  
- type: openshift  
  alias: webapp  
  reference: webapp.yaml  
  endpoints:  
  - name: 'web'  
    port: 8080  
    attributes:  
      discoverable: false  
      public: true  
    protocol: http  
  
- type: openshift  
  alias: mongo  
  reference: mongo-db.yaml  
  endpoints:  
  - name: 'mongo-db'  
    port: 27017  
    attributes:  
      discoverable: true  
      public: false
```

### 3.2.5.2.18. OpenShift resources

Complex deployments can be described using OpenShift resource lists that can be referenced in the devfile. This makes them a part of the workspace.



## IMPORTANT

- Because a CodeReady Workspaces workspace is internally represented as a single deployment, all resources from the OpenShift list are merged into that single deployment.
- Be careful when designing such lists because this can result in name conflicts and other problems.
- Only the following subset of the OpenShift objects are supported: **deployments**, **Pods**, **services**, **persistent volume claims**, **secrets**, and **ConfigMaps**. Kubernetes Ingresses are ignored, but OpenShift routes are supported. A workspace created from a devfile using any other object types fails to start.
- When running CodeReady Workspaces on a OpenShift cluster, only OpenShift lists are supported. When running CodeReady Workspaces on an OpenShift cluster, both OpenShift lists are supported.

```

apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  - type: kubernetes
    reference: ../relative/path/postgres.yaml

```

The preceding component references a file that is relative to the location of the devfile itself. Meaning, this devfile is only loadable by a CodeReady Workspaces factory to which you supply the location of the devfile and therefore it is able to figure out the location of the referenced OpenShift resource list.

The following is an example of the **postgres.yaml** file.

```

apiVersion: v1
kind: List
items:
  -
    apiVersion: v1
    kind: Deployment
    metadata:
      name: postgres
      labels:
        app: postgres
    spec:
      template:
        metadata:
          name: postgres
          app:
            name: postgres
      spec:
        containers:

```

```

- image: postgres
  name: postgres
  ports:
  - name: postgres
    containerPort: 5432
  volumeMounts:
  - name: pg-storage
    mountPath: /var/lib/postgresql/data
  volumes:
  - name: pg-storage
    persistentVolumeClaim:
      claimName: pg-storage
-
  apiVersion: v1
  kind: Service
  metadata:
    name: postgres
    labels:
      app: postgres
      name: postgres
  spec:
    ports:
    - port: 5432
      targetPort: 5432
    selector:
      app: postgres
-
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: pg-storage
    labels:
      app: postgres
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi

```

For a basic example of a devfile with an associated OpenShift list, see [web-nodejs-with-db-sample](#) on redhat-developer GitHub.

If you use generic or large resource lists from which you will only need a subset of resources, you can select particular resources from the list using a selector (which, as the usual OpenShift selectors, works on the labels of the resources in the list).

```

apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
- name: my-go-project
  clonePath: go/src/github.com/acme/my-go-project
  source:
    type: git
    location: https://github.com/acme/my-go-project.git

```



```

components:
- type: kubernetes
  reference: ../relative/path/postgres.yaml
  selector:
    app: postgres

```

Additionally, it is also possible to modify the entrypoints (command and arguments) of the containers present in the resource list. For details of the advanced use case, see the reference (TODO: link).

### 3.2.5.3. Adding commands to a devfile

A devfile allows to specify commands to be available for execution in a workspace. Every command can contain a subset of actions, which are related to a specific component in whose container it will be executed.

```

commands:
- name: build
  actions:
- type: exec
  component: mysql
  command: mvn clean
  workdir: /projects/spring-petclinic

```

You can use commands to automate the workspace. You can define commands for building and testing your code, or cleaning the database.

The following are two kinds of commands:

- CodeReady Workspaces specific commands: You have full control over what component executes the command.
- Editor specific commands: You can use the editor-specific command definitions (example: **tasks.json** and **launch.json** in Che-Theia, which is equivalent to how these files work in VS Code).

#### 3.2.5.3.1. CodeReady Workspaces-specific commands

Each CodeReady Workspaces-specific command features:

- An **action** attribute that is a command to execute.
- A **component** attribute that specifies the container in which to execute the command.

The commands are run using the default shell in the container.

```

apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
- name: my-go-project
  clonePath: go/src/github.com/acme/my-go-project
  source:
    type: git
    location: https://github.com/acme/my-go-project.git
components:

```

```

- type: dockerimage
  image: golang
  alias: go-cli
  memoryLimit: 512Mi
  mountSources: true
  command: ['sleep', 'infinity']
  env:
    - name: GOPATH
      value: ${CHE_PROJECTS_ROOT}/go
    - name: GOCACHE
      value: /tmp/go-cache
  commands:
    - name: compile and run
      actions:
        - type: exec
          component: go-cli
          command: "go get -d && go run main.go"
          workdir: "${CHE_PROJECTS_ROOT}/src/github.com/acme/my-go-project"

```

+

**NOTE**

- If a component to be used in a command must have an alias. This alias is used to reference the component in the command definition. Example: **alias: go-cli** in the component definition and **component: go-cli** in the command definition. This ensures that Red Hat CodeReady Workspaces can find the correct container to run the command in.
- A command can have only one action.

**3.2.5.3.2. Editor-specific commands**

If the editor in the workspace supports it, the devfile can specify additional configuration in the editor-specific format. This is dependent on the integration code present in the workspace editor itself and so is not a generic mechanism. However, the default Che-Theia editor within Red Hat CodeReady Workspaces is equipped to understand the **tasks.json** and **launch.json** files provided in the devfile.

```

apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
  commands:
    - name: tasks
      actions:
        - type: vscode-task
          referenceContent: >
            {
              "version": "2.0.0",
              "tasks": [

```

```

    {
      "label": "create test file",
      "type": "shell",
      "command": "touch ${workspaceFolder}/test.file"
    }
  ]
}

```

This example shows association of a **tasks.json** file with a devfile. Notice the **vscode-task** type that instructs the Che-Theia editor to interpret this command as a tasks definition and **referenceContent** attribute that contains the contents of the file itself. You can also save this file separately from the devfile and use **reference** attribute to specify a relative or absolute URL to it.

In addition to the **vscode-task** commands, the Che-Theia editor understands **vscode-launch** type using which you can specify the launch configurations.

### 3.2.5.3.3. Command preview URL

It is possible to specify a preview URL for commands that expose web UI. This URL is offered for opening when the command is executed.

```

commands:
- name: tasks
  previewUrl:
    port: 8080
    path: /myweb
  actions:
- type: exec
  component: go-cli
  command: "go run webserver.go"
  workdir: ${CHE_PROJECTS_ROOT}/webserver

```

- 1 TCP port where the application listens. Mandatory parameter.
- 2 The path part of the URL to the UI. Optional parameter. The default is root (/).

The example above opens **http://\_\_<server-domain>\_/myweb**, where **<server-domain>** is the URL to the dynamically created OpenShift Ingress or OpenShift Route.

#### 3.2.5.3.3.1. Setting the default way of opening preview URLs

By default, a notification that asks the user about the URL opening preference is displayed.

To specify the preferred way of previewing a service URL:

1. Open CodeReady Workspaces preferences in **File → Settings → Open Preferences** and find **che.task.preview.notifications** in the **CodeReady Workspaces** section.
2. Choose from the list of possible values:
  - **on** – enables a notification for asking the user about the URL opening preferences
  - **alwaysPreview** – the preview URL opens automatically in the **Preview** panel as soon as a task is running

- **alwaysGoTo** – the preview URL opens automatically in a separate browser tab as soon as a task is running
- **off** – disables opening the preview URL (automatically and with a notification)

#### 3.2.5.4. Devfile attributes

Devfile attributes can be used to configure various features.

##### 3.2.5.4.1. Attribute: editorFree

When an editor is not specified in a devfile, a default is provided. When no editor is needed, use the **editorFree** attribute. The default value of **false** means that the devfile requests the provisioning of the default editor.

#### Example of a devfile without an editor

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
components:
  - alias: myApp
    type: kubernetes
    local: my-app.yaml
attributes:
  editorFree: true
```

##### 3.2.5.4.2. Attribute: persistVolumes (ephemeral mode)

By default, volumes and PVCs specified in a devfile are bound to a host folder to persist data even after a container restart. To disable data persistence to make the workspace faster, such as when the volume back end is slow, modify the **persistVolumes** attribute in the devfile. The default value is **true**. Set to **false** to use **emptyDir** for configured volumes and PVC.

#### Example of a devfile with ephemeral mode enabled

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/che-samples/web-java-spring-petclinic.git'
attributes:
  persistVolumes: false
```

### 3.2.6. Objects supported in Red Hat CodeReady Workspaces 2.2

The following table lists the objects that are partially supported in Red Hat CodeReady Workspaces 2.2:

Object	API	OpenShift Infra	OpenShift Infra	Notes
Pod	OpenShift	Yes	Yes	-
Deployment	OpenShift	Yes	Yes	-
ConfigMap	OpenShift	Yes	Yes	-
PVC	OpenShift	Yes	Yes	-
Secret	OpenShift	Yes	Yes	-
Service	OpenShift	Yes	Yes	-
Ingress	OpenShift	Yes	No	Minishift allows you to create Ingress and it works when the host is specified (OpenShift creates a route for it). But, the <b>loadBalancer</b> IP is not provisioned. To add Ingress support for the OpenShift infrastructure node, generate routes based on the provided Ingress.
Route	OpenShift	No	Yes	The OpenShift recipe must be made compatible with the OpenShift Infrastructure and, instead of the provided route, generate Ingress.
Template	OpenShift	Yes	Yes	The OpenShift API does not support templates. A workspace with a template in the recipe starts successfully and the default parameters are resolved.

#### Additional resources

- [Devfile specifications](#)

## 3.3. CREATING AND CONFIGURING A NEW CODEREADY WORKSPACES 2.2 WORKSPACE

### 3.3.1. Creating a new workspace from the dashboard

This procedure describes how to create and edit a new CodeReady Workspaces devfile using the **Dashboard**.

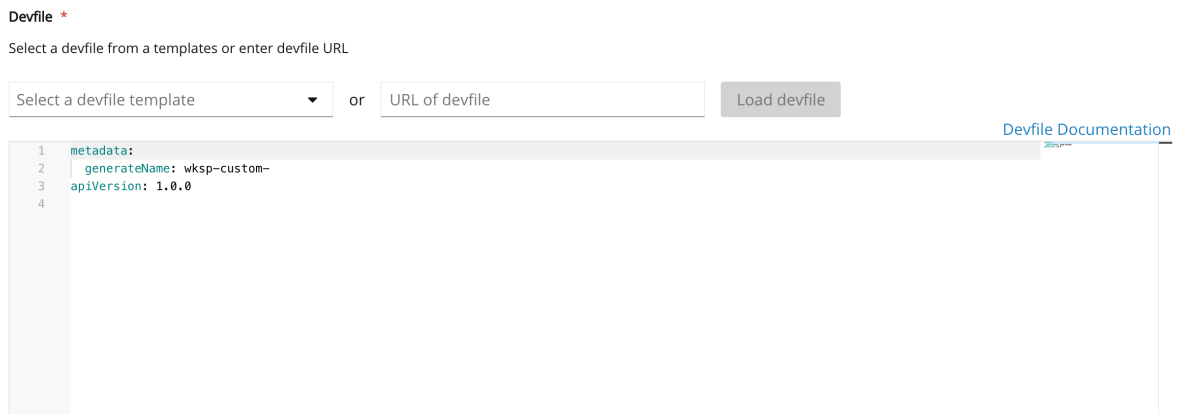
#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).

## Procedure

To edit the devfile:

1. In the **Workspaces** window, click the **Add Workspace** button. The **Custom Workspace** page should be opened.
2. Scroll down to the **Devfile** section. In the **Devfile editor**, add necessary changes.



### EXAMPLE: ADD A PROJECT

To add a project into the workspace, add or edit the following section:

```
projects:
  - name: che
    source:
      type: git
      location: 'https://github.com/eclipse/che.git'
```

See the [Devfile reference](#).

## 3.3.2. Adding projects to your workspace

### Prerequisites

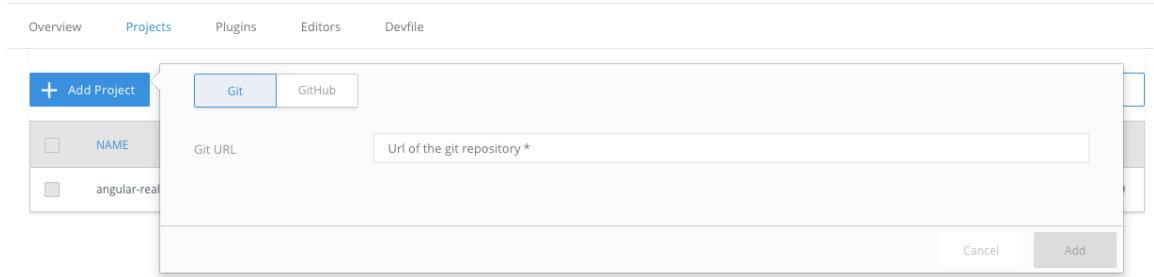
- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

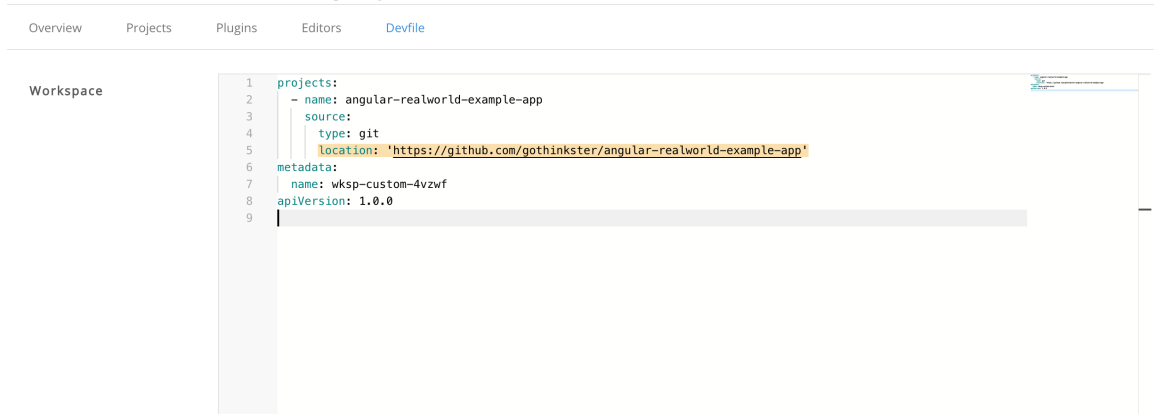
To add a project to your workspace:

1. Navigate to **Workspaces** page and click the workspace you want to update. Here you have two ways to add a project to your workspace:
2. From the **Projects** tab.

- a. Open the **Projects** tab, and then click the **Add Project** button.
- b. Choose if you want to import the project by Git URL or from your GitHub account.



3. From the **Devfile** tab.
  - a. Open the **Devfile** tab.
  - b. In the **Devfile editor**, add **projects** section with desired project.



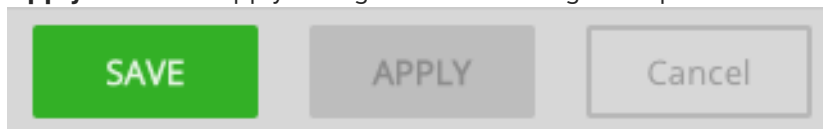
### EXAMPLE: ADD A PROJECT

To add a project into the workspace, add or edit the following section:

```
projects:
- name: che
  source:
    type: git
    location: 'https://github.com/eclipse/che.git'
```

See the [Devfile reference](#).

4. Once the project is added, click **Save** button to save this workspace configuration, or click **Apply** button to apply changes to the running workspace.



### 3.3.3. Configuring the workspace and adding tools

#### 3.3.3.1. Adding plug-ins

##### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

To add plug-ins to your workspace:

1. Click the **Plugins** tab.
2. Enable the plug-in that you want to add and click the **Save** button.

### 3.3.3.2. Defining the workspace editor

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

To define the editor to use with the workspace:

1. Click the **Editors** tab.



#### NOTE

The recommended editor for CodeReady Workspaces 2.2 is Che-Theia.

2. Enable the editor to add and click the **Save** button.
3. Click the **Devfile** tab to view the changes.

Overview   Projects   Plugins   Editors   **Devfile**

#### Workspace

```

1 metadata:
2   name: wksp-che7
3 projects:
4   - name: web-spring-java-simple
5     source:
6       location: 'https://github.com/codenvy-templates/web-spring-java-simple.git'
7       type: git
8 components:
9   - mountSources: false
10     id: eclipse/che-machine-exec-plugin/latest
11     type: chePlugin
12   - mountSources: false
13     id: redhat/java/latest
14     type: chePlugin
15   - mountSources: false
16     id: eclipse/che-theia/latest
17     type: cheEditor
18 apiVersion: 1.0.0
19
```



### 3.3.3.3. Defining specific container images

#### Procedure

To add a new container image:

1. Copy the following section from the **devfile** into **components**:

```
- mountSources: true
  command:
    - sleep
  args:
    - infinity
  memoryLimit: 1Gi
  alias: maven3-jdk11
  type: dockerimage
  endpoints:
    - name: 8080/tcp
      port: 8080
  volumes:
    - name: projects
      containerPath: /projects
  image: 'maven:3.6.0-jdk-11'
```

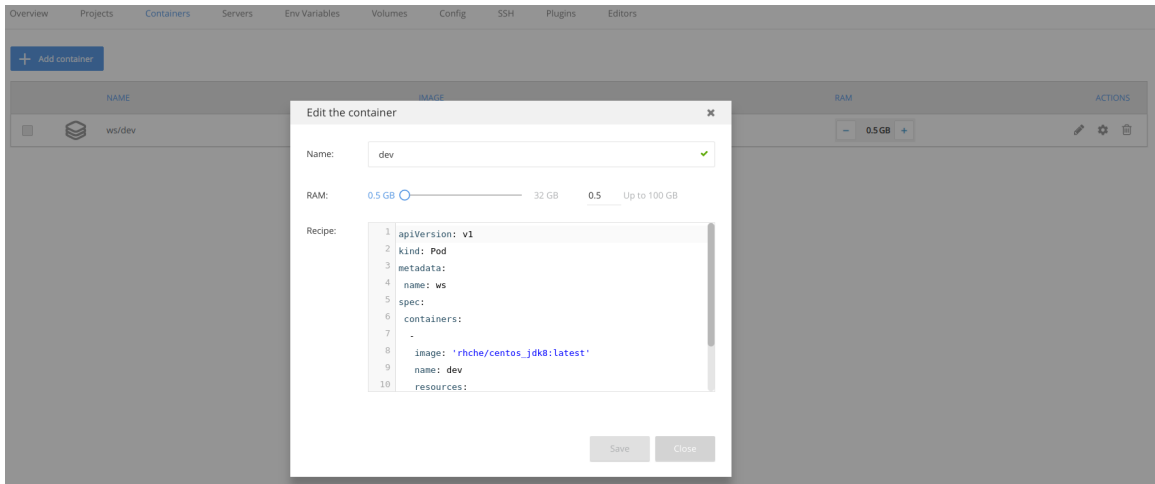
2. When using **type: kubernetes** or **type: openshift**, you must:

- Use separate recipe files.  
To use separate recipe files, you may specify relative or absolute paths. For example:

```
...
  type: kubernetes
  reference: deploy_k8s.yaml
...

...
  type: openshift
  reference: deploy_openshift.yaml
...
```

3. Add a CodeReady Workspaces 2.1 recipe content to the CodeReady Workspaces 2.2 devfile as **referenceContent**:
  - a. Click the **Containers** tab (**Workspace** → **Details** → **Containers**).



- b. Copy the CodeReady Workspaces 2.1 recipe, and paste it into the separate CodeReady Workspaces 2.2 component as a **referenceContent**.

```

Overview  Projects  Plugins  Editors  Devfile
-----
Workspace
45 - env: []
46 args:
47   - infinity
48 selector: {}
49 memoryLimit: 512Mi
50 preferences: {}
51 volumes: []
52 entrypoints: []
53 referenceContent: |
54   apiVersion: v1
55   kind: Pod
56   metadata:
57     name: ws
58   spec:
59     containers:
60     -
61       image: 'rhche/centos_jdk8:latest'
62       name: dev
63       resources:
64         limits:
65           memory: 512Mi
66 command:
67   - sleep
68 endpoints: []
69 mountSources: true
70 type: kubernetes
71 apiVersion: 1.0.0
72 attributes: {}
73

```

- c. Set the type from the original CodeReady Workspaces 2.1 configuration. The following is an example of the resulting file:

```

type: kubernetes
referenceContent: |
  apiVersion: v1
  kind: Pod
  metadata:
    name: ws
  spec:
    containers:
    -
      image: 'rhche/centos_jdk8:latest'
      name: dev
      resources:
        limits:
          memory: 512Mi

```

4. Copy the required fields from the old workspace (**image, volumes, endpoints**). For example:

```

17     endpoints:
18       - name: 8080/tcp
19         port: 8080
20     volumes:
21       - name: m2
22         containerPath: /home/user/.m2
23     image: 'maven:3.6.0-jdk-11'

```

- Change the **memoryLimit** and **alias** variables, if needed. Here, the field **alias** is used to set a name for the component. It is generated automatically from the **image** field, if not set.

```

image: 'maven:3.6.0-jdk-11'
alias: maven3-jdk11

```

- Change the **memoryLimit**, **memoryRequest**, or both fields to specify the **RAM** required for the component.

```

alias: maven3-jdk11
memoryLimit: 256M
memoryRequest: 128M

```

- Open the **Devfile** tab to see the changes.

Overview    Projects    Plugins    Editors    **Devfile**

### Workspace

```

7     type: git
8     branch: master
9 components:
10    - mountSources: true
11      endpoints:
12        - name: 8080/tcp
13          port: 8080
14      command:
15        - sleep
16      args:
17        - infinity
18      memoryLimit: 1Gi
19      type: dockerimage
20      volumes:
21        - name: projects
22          containerPath: /projects
23      image: 'maven:3.6.0-jdk-11'
24      alias: maven3-jdk11
25    - mountSources: false
26      id: redhat/java/latest
27      type: chePlugin
28    - mountSources: false
29      id: eclipse/che-machine-exec-plugin/latest
30      type: chePlugin
31    - mountSources: false
32      id: eclipse/che-theia/latest
33      type: cheEditor
34  apiVersion: 1.0.0
35

```

- Repeat the steps to add additional container images.

### 3.3.3.4. Adding commands to your workspace

The following is a comparison between workspace configuration commands in CodeReady Workspaces 2.1 (Figure 1) and CodeReady Workspaces 2.2 (Figure 2):

**Figure 3.1. An example of the Workspace configuration commands in CodeReady Workspaces 2.2**

Overview
Projects
Plugins
Editors
Devfile

**Workspace**

```

1 metadata:
2   name: wksp-che7
3 projects:
4   - name: web-spring-java-simple
5     source:
6       location: 'https://github.com/codenvy-templates/web-spring-java-simple.git'
7       type: git
8 components:
9   - mountSources: false
10     id: eclipse/che-machine-exec-plugin/latest
11     type: chePlugin
12   - mountSources: false
13     id: redhat/java/latest
14     type: chePlugin
15   - mountSources: false
16     id: eclipse/che-theia/latest
17     type: cheEditor
18 apiVersion: 1.0.0
19
```

## Procedure

To define commands to your workspace, edit the workspace devfile:

- Add (or replace) the **commands** section with the first command. Change the **name** and the **command** fields from the original workspace configuration (see the preceding equivalence table).

```

commands:
- name: build
  actions:
- type: exec
  command: mvn clean install
```

- Copy the following YAML code into the **commands** section to add a new command. Change the **name** and the **command** fields from the original workspace configuration (see the preceding equivalence table).

```

- name: build and run
  actions:
- type: exec
  command: mvn clean install && java -jar
```

- Optionally, add the **component** field into **actions**. This indicates the component alias where the command will be performed.
- Repeat step 2 to add more commands to the devfile.
- Click the **Devfile** tab to view the changes.

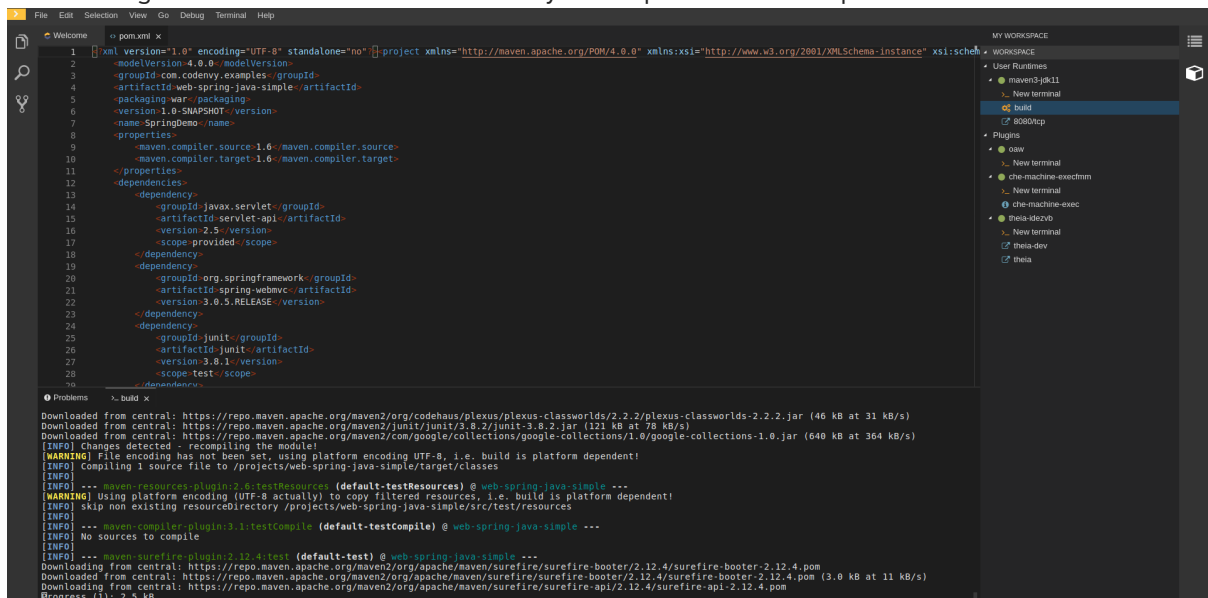
## Workspace

```

13     port: 8080
14     command:
15     - sleep
16     args:
17     - infinity
18     memoryLimit: 1Gi
19     type: dockerimage
20     volumes:
21     - name: projects
22       containerPath: /projects
23     image: 'maven:3.6.0-jdk-11'
24     alias: maven3-jdk11
25     - mountSources: false
26       id: redhat/java/latest
27       type: chePlugin
28     - mountSources: false
29       id: eclipse/che-machine-exec-plugin/latest
30       type: chePlugin
31     - mountSources: false
32       id: eclipse/che-theia/latest
33       type: cheEditor
34   apiVersion: 1.0.0
35   commands:
36   - name: build
37     actions:
38     - workdir: /projects/web-spring-java-simple
39       type: exec
40       command: mvn clean install
41       component: maven3-jdk11

```

6. Save changes and start the new CodeReady Workspaces 2.2 workspace.



### 3.4. IMPORTING A OPENSIFT APPLICATION INTO A WORKSPACE

This section describes how to import a OpenShift application into a CodeReady Workspaces workspace.

For demonstration purposes, the section uses a sample OpenShift application having the following two Pods:

- A Node.js application specified by this [nodejs-app.yaml](#)

- A MongoDB Pod specified by this [mongo-db.yaml](#)

To run the application on a OpenShift cluster:

```
$ node=https://raw.githubusercontent.com/redhat-developer/devfile/master/samples/web-nodejs-with-db-sample/nodejs-app.yaml && \
mongo=https://raw.githubusercontent.com/redhat-developer/devfile/master/samples/web-nodejs-with-db-sample/mongo-db.yaml && \
oc apply -f ${mongo} && \
oc apply -f ${node}
```

To deploy a new instance of this application in a CodeReady Workspaces workspace, use one of the following three scenarios:

- Starting from scratch: [Writing a new devfile](#)
- Modifying an existing workspace: [Using the Dashboard user interface](#)
- From a running application: [Generating a devfile with `crwctl`](#)

### 3.4.1. Including a OpenShift application in a workspace devfile definition

This procedure demonstrates how to define the CodeReady Workspaces 2.2 workspace devfile by OpenShift application.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces 'quick-starts'](#).
- `crwctl` management tool is installed. See [the CodeReady Workspaces 2.2 Installation Guide](#).

The devfile format is used to define a CodeReady Workspaces workspace, and its format is described in the [Making a workspace portable using a devfile](#) section. The following is an example of the simplest devfile:

```
apiVersion: 1.0.0
metadata:
  name: minimal-workspace
```

Only the name (`minimal-workspace`) is specified. After the CodeReady Workspaces server processes this devfile, the devfile is converted to a minimal CodeReady Workspaces workspace that only has the default editor (Che-Theia) and the default editor plug-ins (example: the terminal).

Use the **OpenShift** type of components in the devfile to add OpenShift applications to a workspace.

For example, the user can embed the **NodeJS-Mongo** application in the minimal-workspace defined in this paragraph by adding a **components** section.

```
apiVersion: 1.0.0
metadata:
  name: minimal-workspace
components:
  - type: kubernetes
```

```

reference: https://raw.githubusercontent.com/.../mongo-db.yaml
- alias: nodejs-app
  type: kubernetes
reference: https://raw.githubusercontent.com/.../nodejs-app.yaml
entrypoints:
- command: ['sleep']
  args: ['infinity']

```

Note that the **sleep infinity** command is added as the entrypoint of the Node.js application. This prevents the application from starting at the workspace start phase. It allows the user to start it when needed for testing or debugging purposes.

To make it easier for a developer to test the application, add the commands in the devfile:

```

apiVersion: 1.0.0
metadata:
  name: minimal-workspace
components:
- type: kubernetes
  reference: https://raw.githubusercontent.com/.../mongo-db.yaml
- alias: nodejs-app
  type: kubernetes
  reference: https://raw.githubusercontent.com/.../nodejs-app.yaml
entrypoints:
- command: ['sleep']
  args: ['infinity']
commands:
- name: run
  actions:
- type: exec
  component: nodejs-app
  command: cd ${CHE_PROJECTS_ROOT}/nodejs-mongo-app/EmployeeDB/ && npm install &&
sed -i -- "s/localhost/mongo/g" app.js && node app.js

```

Use this devfile to create and start a workspace with the **crwctl** command:

```
$ crwctl workspace:start --devfile <devfile-path>
```

The **run** command added to the devfile is available as a task in Che-Theia from the command palette. When executed, the command starts the Node.JS application.

### 3.4.2. Adding a OpenShift application to an existing workspace using the dashboard

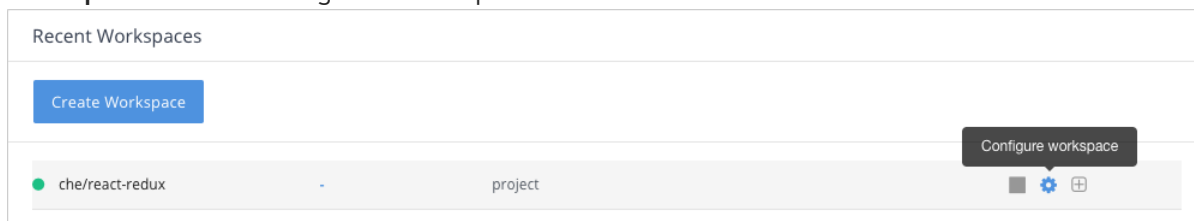
This procedure demonstrates how to modify an existing workspace and import the OpenShift application using the newly created devfile.

#### Prerequisites

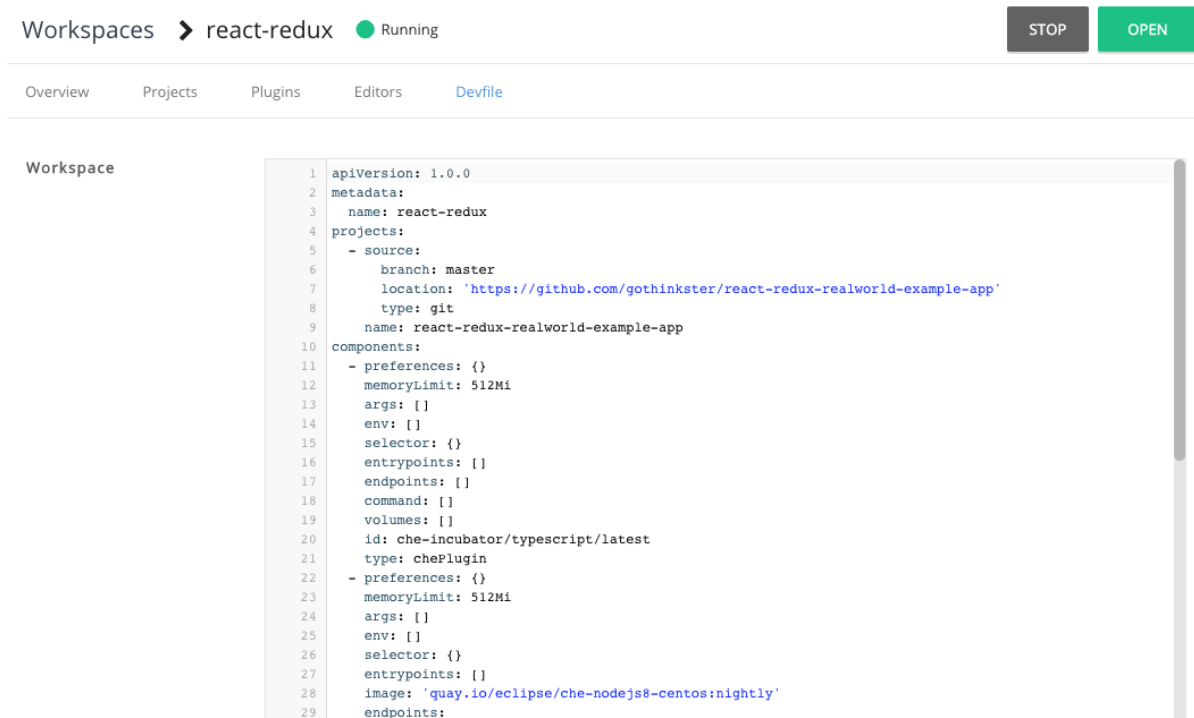
- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

1. After the creation of a workspace, use the **Workspace** menu and then the **Configure workspace** icon to manage the workspace.



2. To modify the workspace details, use the **Devfile** tab. The workspace details are displayed in this tab in the devfile format.



3. To add a OpenShift component, use the **Devfile** editor on the dashboard.
4. For the changes to take effect, save the devfile and restart the CodeReady Workspaces workspace.

### 3.4.3. Generating a devfile from an existing OpenShift application

This procedure demonstrates how to generate a devfile from an existing OpenShift application using the **crwctl** tool.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) CodeReady Workspaces 'quick-starts'.
- **crwctl** management tool is installed. See [the CodeReady Workspaces 2.2 Installation Guide](#).

#### Procedure

1. Use the **crwctl devfile:generate** command to generate a devfile:



```
$ crwctl devfile:generate
```

- The user can also use the **crwctl devfile:generate** command to generate a devfile from, for example, the **NodeJS-MongoDB** application.  
The following example generates a devfile that includes the **NodeJS** component:

```
$ crwctl devfile:generate --selector="app=nodejs"
apiVersion: 1.0.0
metadata:
  name: crwctl-generated
components:
- type: kubernetes
  alias: app=nodejs
  referenceContent: |
    kind: List
    apiVersion: v1
    metadata:
      name: app=nodejs
    items:
    - apiVersion: apps/v1
      kind: Deployment
      metadata:
        labels:
          app: nodejs
          name: web
  (...)

```

The Node.js application YAML definition is included in the devfile, inline, using the **referenceContent** attribute.

- To include support for a language, use the **--language** parameter:

```
$ crwctl devfile:generate --selector="app=nodejs" --language="typescript"
apiVersion: 1.0.0
metadata:
  name: crwctl-generated
components:
- type: kubernetes
  alias: app=nodejs
  referenceContent: |
    kind: List
    apiVersion: v1
  (...)
- type: chePlugin
  alias: typescript-ls
  id: che-incubator/typescript/latest

```

2. Use the generated devfile to start a CodeReady Workspaces workspace with **crwctl**.

### 3.5. REMOTELY ACCESSING WORKSPACES

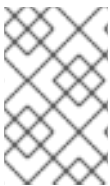
This section describes how to remotely access CodeReady Workspaces workspaces outside of the browser.

CodeReady Workspaces workspaces exist as containers and are, by default, modified from a browser window. In addition to this, there are the following methods of interacting with a CodeReady Workspaces workspace:

- Opening a command line in the workspace container using the OpenShift command-line tool, **kubectl**
- Uploading and downloading files using the **kubectl** tool

### 3.5.1. Remotely accessing workspaces using the OpenShift command-line tool

To access CodeReady Workspaces workspaces remotely using OpenShift command-line tool (**kubectl**), follow the instructions in this section.



#### NOTE

The **kubectl** tool is used in this section to open a shell and manage files in a CodeReady Workspaces workspace. Alternatively, it is possible to use the **oc** OpenShift command-line tool.

#### Prerequisites

- The **kubectl** binary file from the [OpenShift website](#).
- Verify the installation of **kubectl** using the **oc version** command:

```
$ oc version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0",
GitCommit:"e8462b5b5dc2584fdcd18e6bcfe9f1e4d970a529", GitTreeState:"clean",
BuildDate:"2019-06-19T16:40:16Z", GoVersion:"go1.12.5", Compiler:"gc",
Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0",
GitCommit:"e8462b5b5dc2584fdcd18e6bcfe9f1e4d970a529", GitTreeState:"clean",
BuildDate:"2019-06-19T16:32:14Z", GoVersion:"go1.12.5", Compiler:"gc",
Platform:"linux/amd64"}
```

For versions 1.5.0 or higher, proceed with the steps in this section.

#### Procedure

1. Use the **exec** command to open a remote shell.
2. To find the name of the OpenShift project and the Pod that runs the CodeReady Workspaces workspace:

```
$ oc get pod -l che.workspace_id --all-namespaces
NAMESPACE NAME READY STATUS RESTARTS AGE
che workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4 4/4 Running 0
6m4s
```

In the example above, the Pod name is **workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4**, and the project is **codeready**.

1. To find the name of the container:

```
$ NAMESPACE=che
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ oc get pod ${POD} -o custom-columns=CONTAINERS:.spec.containers[*].name
CONTAINERS
maven,che-machine-execpau,theia-ide6dj,vscode-javaw92
```

- When you have the project, pod name, and the name of the container, use the **kubectl** command to open a remote shell:

```
$ NAMESPACE=che
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ CONTAINER=maven
$ oc exec -ti -n ${NAMESPACE} ${POD} -c ${CONTAINER} bash
user@workspace7b2wemdf3hx7s3ln $
```

- From the container, execute the **build** and **run** commands (as if from the CodeReady Workspaces workspace terminal):

```
user@workspace7b2wemdf3hx7s3ln $ mvn clean install
[INFO] Scanning for projects...
(...)
```

#### Additional resources

- For more about **kubectl**, see the [OpenShift documentation](#).

### 3.5.2. Downloading and uploading a file to a workspace using the command-line interface

This procedure describes how to use the **kubectl** tool to download or upload files remotely from or to an Red Hat CodeReady Workspaces workspace.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- Remote access to the CodeReady Workspaces workspace you intend to modify. For instructions see [Section 3.5.1, "Remotely accessing workspaces using the OpenShift command-line tool"](#).
- The **kubectl** binary file from the [OpenShift website](#).
- Verify the installation of **kubectl** using the **oc version** command:

#### Procedure

- To download a local file named **downloadme.txt** from a workspace container to the current home directory of the user, use the following in the CodeReady Workspaces remote shell.

```
$ REMOTE_FILE_PATH=/projects/downloadme.txt
$ NAMESPACE=che
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
```

```
$ CONTAINER=maven
$ oc cp ${NAMESPACE}/${POD}:${REMOTE_FILE_PATH} ~/downloadme.txt -c
${CONTAINER}
```

- To upload a local file named **uploadme.txt** to a workspace container in the **/projects** directory:

```
$ LOCAL_FILE_PATH=./uploadme.txt
$ NAMESPACE=che
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ CONTAINER=maven
$ oc cp ${LOCAL_FILE_PATH} ${NAMESPACE}/${POD}:/projects -c ${CONTAINER}
```

Using the preceding steps, the user can also download and upload directories.

## 3.6. CREATING A WORKSPACE FROM CODE SAMPLE

Every stack includes a sample codebase, which is defined by the devfile of the stack. This section explains how to create a workspace from this code sample in a sequence of three procedures.

1. Creating a workspace from the user dashboard:
  - a. Using the [Get Started view](#).
  - b. Using the [Custom Workspace view](#).
2. [Changing the configuration of the workspace](#) to add code sample.
3. [Running an existing workspace from the user dashboard](#).

For more information about devfiles, see [Configuring a CodeReady Workspaces workspace using a devfile](#).

### 3.6.1. Creating a workspace from Get Started view of User Dashboard

This section describes how to create a workspace from the User Dashboard.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#)

#### Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the Dashboard](#).
2. In the left navigation panel, go to **Get Started**.
3. Click the **Get Started** tab.
4. In the gallery, there is list of samples that may be used to build and run projects.

Get Started







Custom Workspace

## Select a Sample

Select a sample to create your first workspace.

Filter by

 Temporary Storage ⓘ 26 items


 <p><b>NodeJS Angular Web Application</b> Stack for developing NodeJS Angular Web Application</p>	 <p><b>Apache Camel K</b> Stack with tooling ready to develop Integration projects with Apache Camel K</p>	 <p><b>Apache Camel based on Spring Boot</b> Stack with environment ready to develop Integration projects with Apache Camel based on Spring Boot.</p>
 <p><b>Mainframe Basic Stack</b> Che4z Mainframe Basic Stack is an all-in-one extension pack for developers working with z/OS applications, suitable for all levels of mainframe experience, even beginners.</p>	 <p><b>C/C++</b> Stack with C/C++ and Clang 8</p>	 <p><b>.NET Core</b> Stack with .Net 2.2</p>



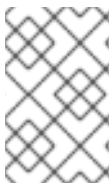
## CHANGING RESOURCE LIMITS

Changing the memory requirements is only possible [from the devfile](#).

5. Start the workspace: click the chosen stack card.



**NodeJS Angular Web Application**  
Stack for developing NodeJS Angular Web Application



## NEW WORKSPACE NAME

Workspace name can be auto-generated based on the underlying devfile of the stack. Generated names always consist of the devfile **metadata.generateName** property as the prefix and four random characters.

### 3.6.2. Creating a workspace from Custom Workspace view of User Dashboard

This section describes how to create a workspace from the User Dashboard.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#)

#### Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the Dashboard](#).
2. In the left navigation panel, go to **Get Started**.
3. Click the **Custom Workspace** tab.
4. Define a **Name** for the workspace.



## NEW WORKSPACE NAME

Workspace name can be auto-generated based on the underlying devfile of the stack. Generated names always consist of the devfile **metadata.generateName** property as the prefix and four random characters.

5. In the **Devfile** section, select the devfile template that will be used to build and run projects.

**Devfile \***  
Select a devfile from a templates or enter devfile URL

Select a devfile template  or  URL of devfile

- Go
- Java Gradle
- Java Maven
- Java with Spring Boot and MongoDB
- Java with Spring Boot and MySQL
- Java Spring Boot
- Java Vert.x
- NodeJS Express Web Application
- NodeJS MongoDB Web Application

[Devfile Documentation](#)



## CHANGING RESOURCE LIMITS

Changing the memory requirements is only possible [from the devfile](#).

6. Start the workspace: click the **Create & Open** button at the bottom of the form:

## Create & Open

### 3.6.3. Changing the configuration of an existing workspace

This section describes how to change the configuration of an existing workspace from the User Dashboard.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) CodeReady Workspaces 'quick-starts'.
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

#### Procedure

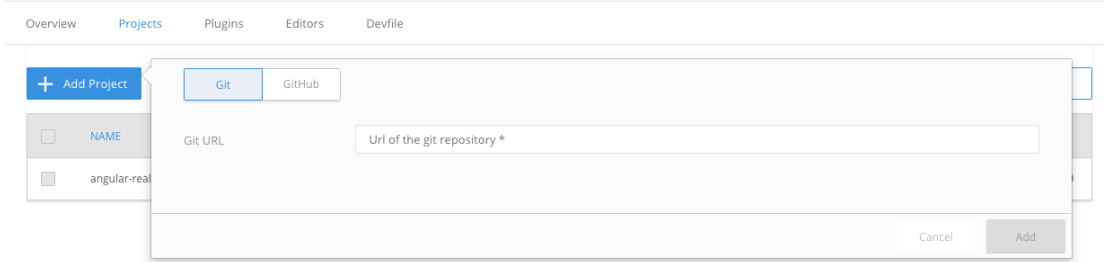
1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the dashboard](#).
2. In the left navigation panel, go to **Workspaces**.
3. Click the name of a workspace to navigate to the configuration overview page.
4. Click the **Overview** tab and execute following actions:
  - Change the **Workspace name**.
  - Toggle **Ephemeral mode**.
  - **Export** the workspace configuration to a file or private cloud.
  - **Delete** the workspace.

The screenshot shows the configuration overview page for a workspace named 'wksp-2ajq'. The workspace is currently 'Stopped'. At the top right, there are 'RUN' and 'OPEN' buttons. Below the workspace name, there are tabs for 'Overview', 'Projects', 'Plugins', 'Editors', and 'Devfile'. The 'Overview' tab is selected. The configuration options are:

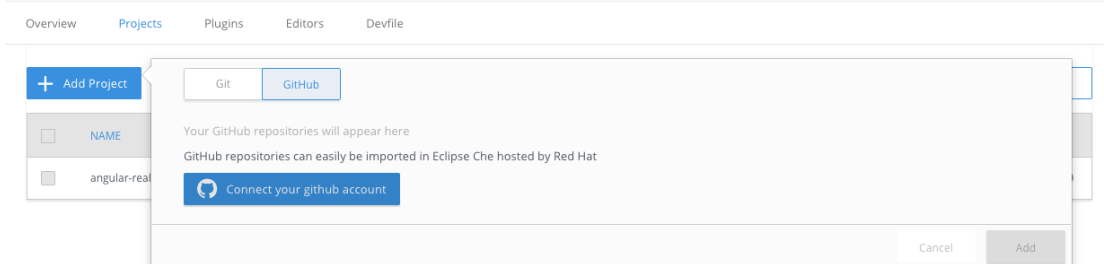
- Workspace name:** A text input field containing 'wksp-2ajq' with a green checkmark on the right.
- Ephemeral mode:** A toggle switch that is currently turned off.
- Export:** Two buttons: 'Export as a file' and 'Export to private cloud'.
- Delete:** A red button labeled 'Delete'.

5. In the **Projects** section, choose the projects to integrate in the workspace.

- a. Click the **Add Project** button and do one of the following:
  - i. Enter the project Git repository URL to integrate in the workspace:



- ii. Connect your GitHub account and select projects to integrate:



- b. Click the **Add** button.
6. In the **Plugins** section, choose the plug-ins to integrate in the workspace.

## EXAMPLE

Start with a generic Java-based stack, then add support for Node.js or Python.

7. In the **Editors** section, choose the editors to integrate in the workspace. The CodeReady Workspaces 2.2 editor is based on Che-Theia.
8. From the **Devfile** tab, edit YAML configuration of the workspace. See the [Devfile reference](#).

## EXAMPLE: ADD COMMANDS

Workspace

```

47     -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true
48     -Xms20m -Djava.security.egd=file:/dev/./urandom
49     name: JAVA_TOOL_OPTIONS
50     - value: '${echo ${0}}\${$'
51     name: PS1
52     - value: /home/user
53     name: HOME
54 apiVersion: 1.0.0
55 commands:
56   - name: build the project
57     actions:
58       - type: exec
59         command: 'cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn clean install'
60         component: maven
61   - name: run the services
62     actions:
63       - type: exec
64         command: >-
65           cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
66           -DskipTests
67         component: maven
68   - name: run and debug the services
69     actions:
70       - type: exec
71         command: >-
72           cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
73           -DskipTests -Drun.jvmArguments="-Xdebug
74           -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=5005"
75         component: maven

```



## EXAMPLE: ADD A PROJECT

To add a project into the workspace, add or edit the following section:

```

projects:
  - name: che
    source:
      type: git
      location: 'https://github.com/eclipse/che.git'

```

### 3.6.4. Running an existing workspace from the User Dashboard

This section describes how to run an existing workspace from the User Dashboard.

#### 3.6.4.1. Running an existing workspace from the User Dashboard with the Run button

This section describes how to run an existing workspace from the User Dashboard using the **Run** button.

##### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

##### Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the dashboard](#).
2. In the left navigation panel, navigate to **Workspaces**.
3. Click on the name of a non-running workspace to navigate to the overview page.
4. Click on the **Run** button in the top right corner of the page.
5. The workspace is started.
6. The browser **does not** navigates to the workspace.

#### 3.6.4.2. Running an existing workspace from the User Dashboard using the Open button

This section describes how to run an existing workspace from the User Dashboard using the **Open** button.

##### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the dashboard](#).
2. In the left navigation panel, navigate to **Workspaces**.
3. Click on the name of a non-running workspace to navigate to the overview page.
4. Click on the **Open** button in the top right corner of the page.
5. The workspace is started.
6. The browser navigates to the workspace.

### 3.6.4.3. Running an existing workspace from the User Dashboard using the Recent Workspaces

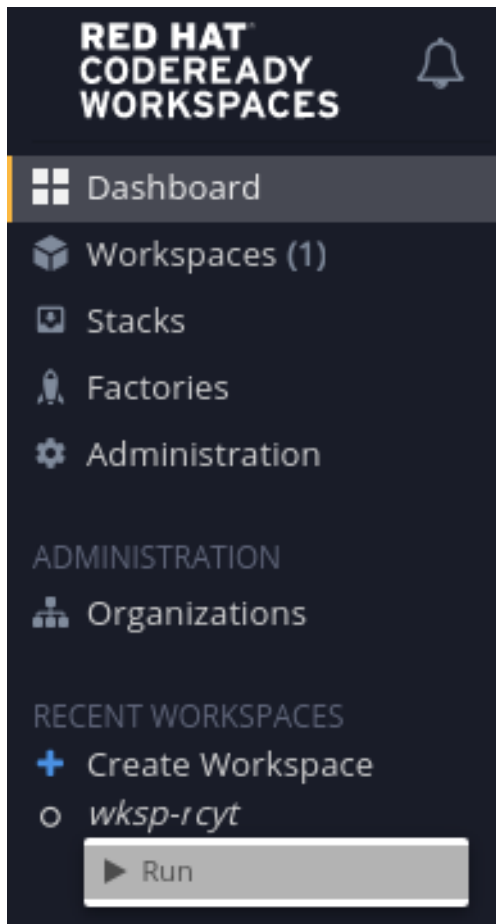
This section describes how to run an existing workspace from the User Dashboard using the Recent Workspaces.

## Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide CodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

## Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the dashboard](#).
2. In the left navigation panel, in the **Recent Workspaces** section, right-click the name of a non-running workspace and click **Run** in the contextual menu to start it.



### 3.7. CREATING A WORKSPACE BY IMPORTING THE SOURCE CODE OF A PROJECT

This section describes how to create a new workspace to edit an existing codebase.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace with plug-ins related to your development environment defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

There are two ways to do that **before** starting a workspace:

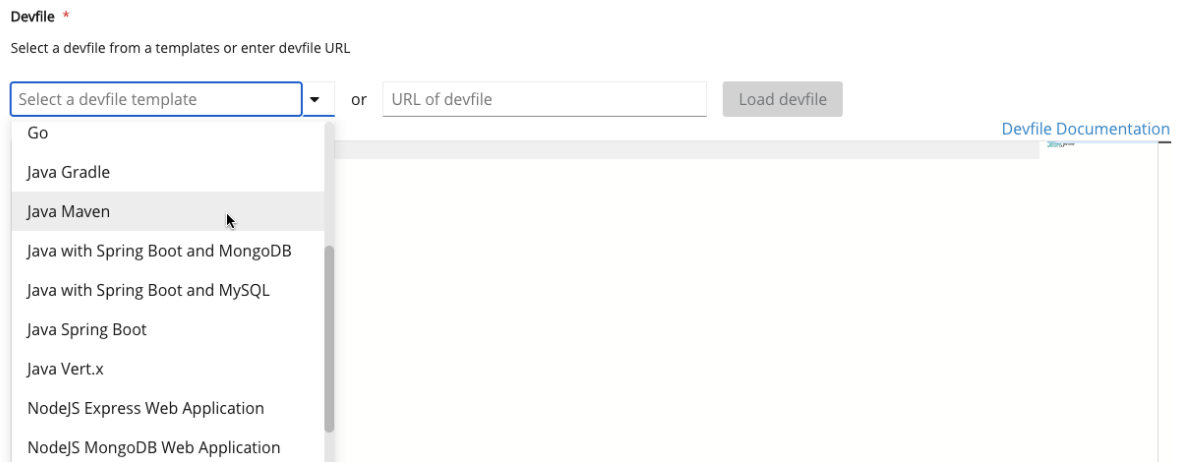
- [Select a sample from the Dashboard, then change the devfile to include your project](#)
- [Add a devfile to a git repository and start the workspace using crwctl or a factory](#)

To create a new workspace to edit an existing codebase, use one of the following three methods **after** you have started the workspace:

- [Import from the Dashboard into an existing workspace](#)
- [Import to a running workspace using the `git clone` command](#)
- [Import to a running workspace using `git clone` in a terminal](#)

### 3.7.1. Select a sample from the Dashboard, then change the devfile to include your project

- In the left navigation panel, go to **Get Started**.
- Click the **Custom Workspace** tab if it's not already selected.
- In the **Devfile** section, select the devfile template that will be used to build and run projects.



- In the **Devfile editor**, update **projects** section:



#### EXAMPLE: ADD A PROJECT

To add a project into the workspace, add or edit the following section:

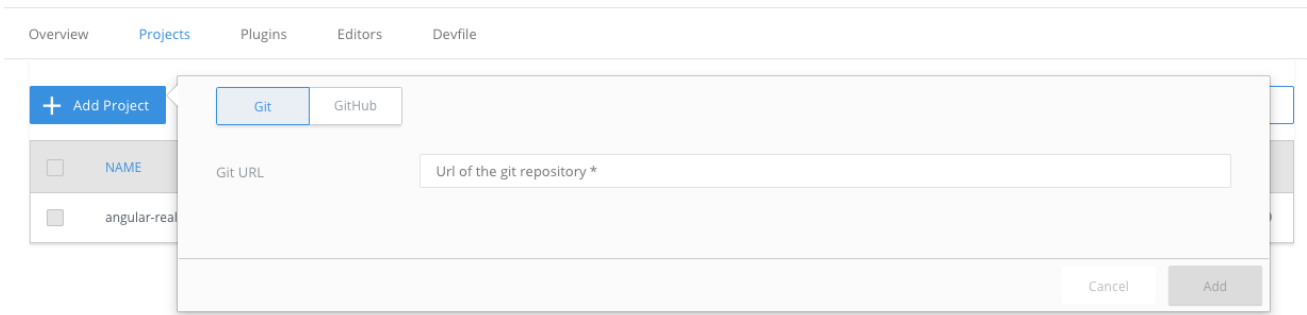
```
projects:
- name: che
  source:
    type: git
    location: 'https://github.com/eclipse/che.git'
```

See the [Devfile reference](#).

- To open the workspace, click the **Create & Open** button.

### 3.7.2. Importing from the Dashboard into an existing workspace

1. Import the project. There are at least two ways to import a project using the **Dashboard**.
  - From the **Dashboard**, select **Workspaces**, then select your workspace by clicking on its name. This will link you to the workspace's **Overview** tab.
  - Or, use the gear icon. This will link to the **Devfile** tab where you can enter your own YAML configuration.
2. Click the **Projects** tab.
3. Click **Add Project**. You can then import project by a repository Git URL or from GitHub.



## NOTE

You can add a project to a non-running workspace, but you must start the workspace to delete it.

### 3.7.2.1. Editing the commands after importing a project

After you have a project in your workspace, you can add commands to it. Adding commands to your projects allows you to run, debug, or launch your application in a browser.

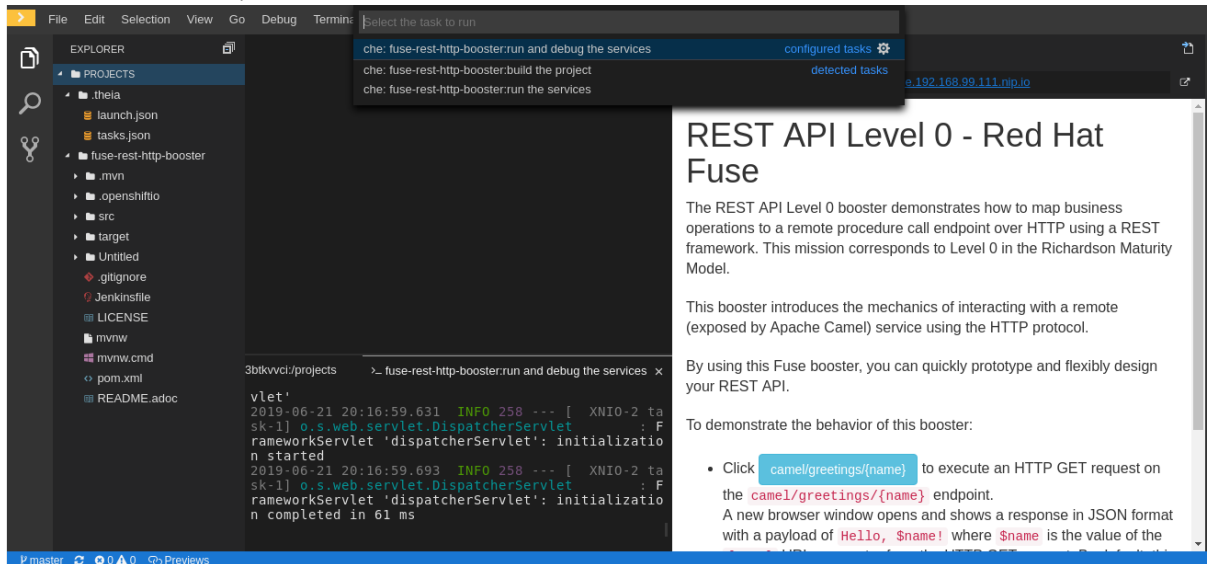
To add commands to the project:

1. Open the workspace configuration in the **Dashboard**, then select the **Devfile** tab.

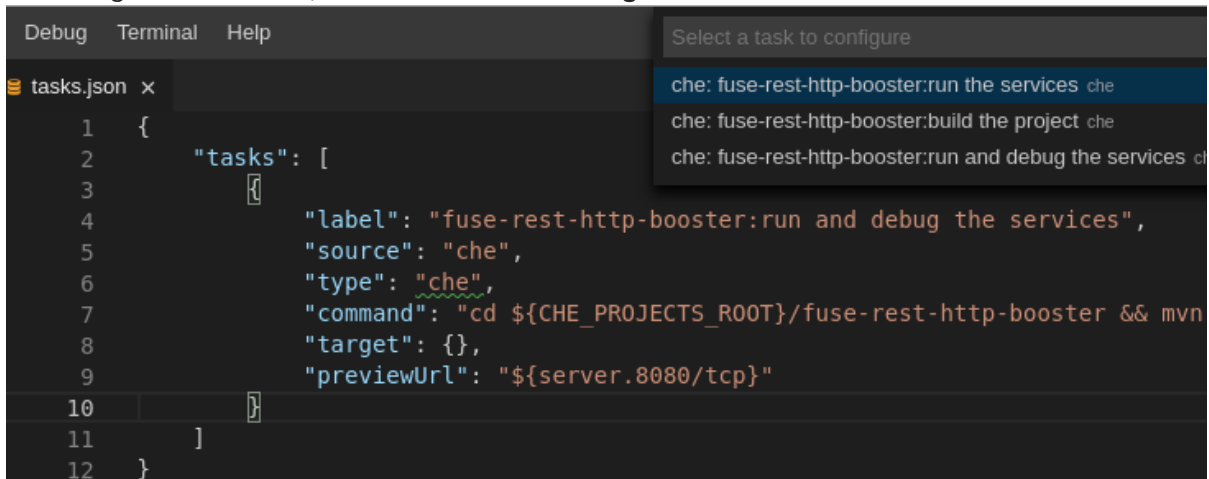
Workspace	<pre> 47     -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true 48     -Xms20m -Djava.security.egd=file:/dev/./urandom 49     name: JAVA_TOOL_OPTIONS 50     - value: '\${echo \${0}}\\${' 51     name: PS1 52     - value: /home/user 53     name: HOME 54     apiVersion: 1.0.0 55     commands: 56     - name: build the project 57       actions: 58         - type: exec 59           command: 'cd \${CHE_PROJECTS_ROOT}/fuse-rest-http-booster &amp;&amp; mvn clean install' 60           component: maven 61     - name: run the services 62       actions: 63         - type: exec 64           command: &gt;- 65             cd \${CHE_PROJECTS_ROOT}/fuse-rest-http-booster &amp;&amp; mvn spring-boot:run 66             -DskipTests 67           component: maven 68     - name: run and debug the services 69       actions: 70         - type: exec 71           command: &gt;- 72             cd \${CHE_PROJECTS_ROOT}/fuse-rest-http-booster &amp;&amp; mvn spring-boot:run 73             -DskipTests -Drun.jvmArguments="-Xdebug 74             -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=5005" 75           component: maven </pre>
-----------	--

2. Open the workspace.

- To run a command, select **Terminal** > **Run Task** from the main menu.



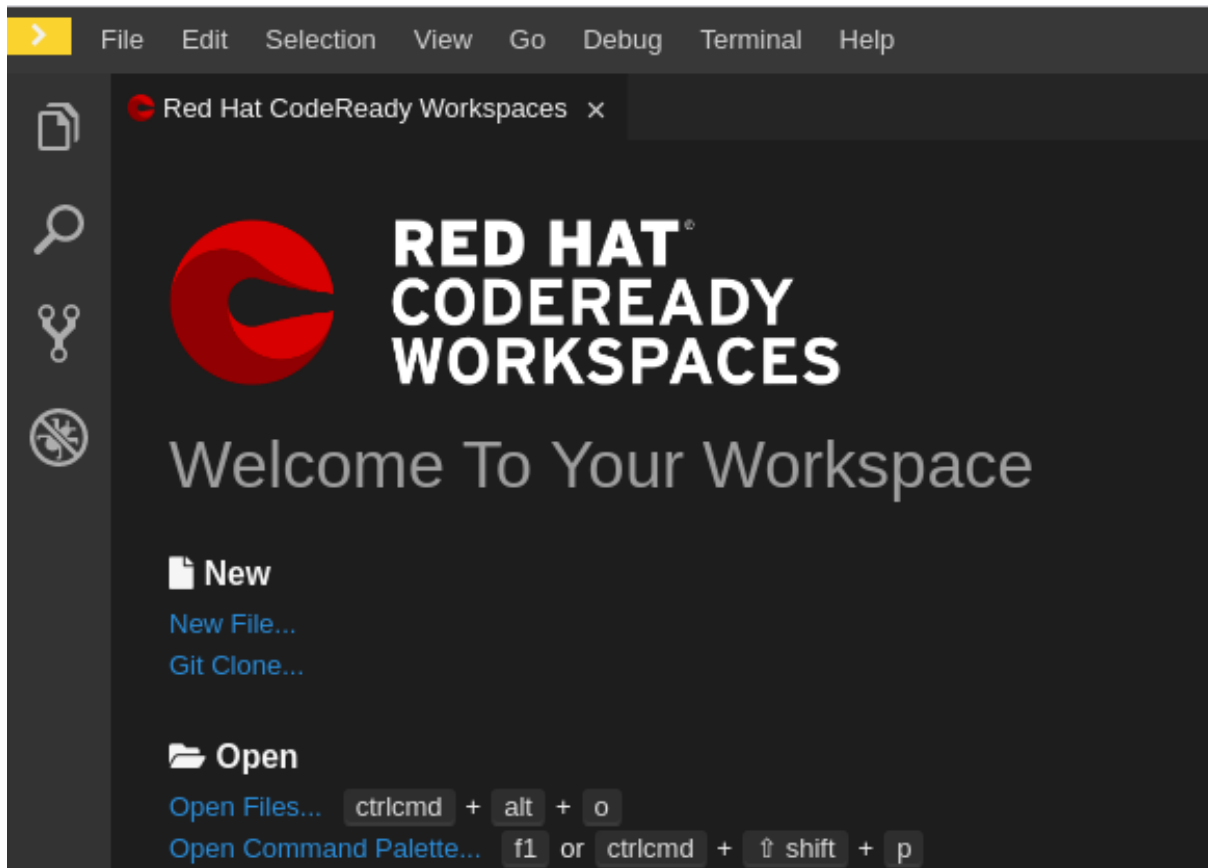
- To configure commands, select **Terminal** > **Configure Tasks** from the main menu.



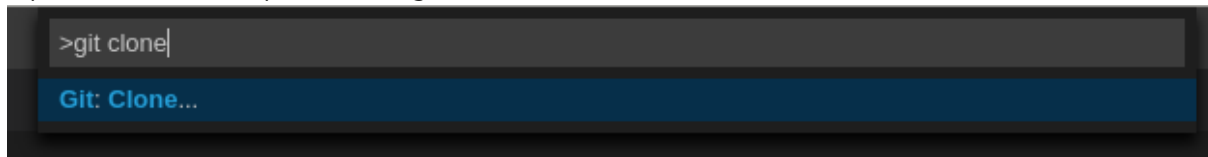
### 3.7.3. Importing to a running workspace using the Git: Clone command

To import to a running workspace using the **Git: Clone** command:

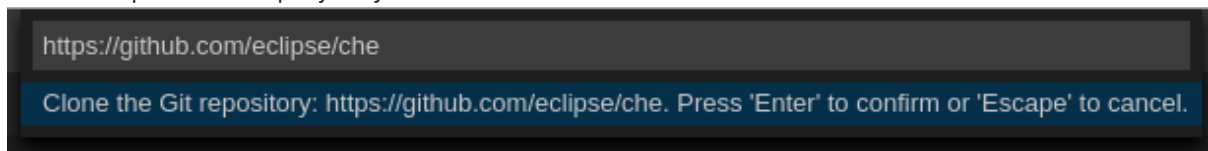
- Start a workspace, then use the **Git: Clone** command from the command palette or the Welcome screen to import a project to a running workspace.



2. Open the command palette using **F1** or **CTRL-SHIFT-P**, or from the link in the Welcome screen.

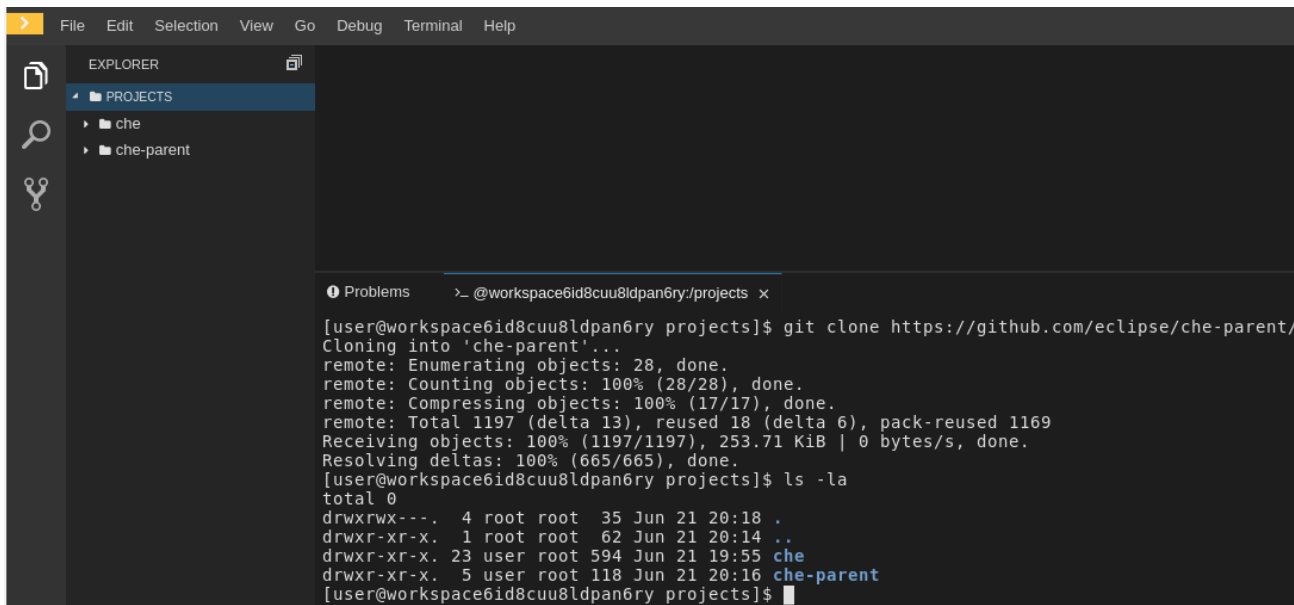


3. Enter the path to the project you want to clone.



### 3.7.4. Importing to a running workspace with git clone in a terminal

In addition to the approaches above, you can also start a workspace, open a **Terminal**, and type **git clone** to pull code.



```

File Edit Selection View Go Debug Terminal Help
EXPLORER
PROJECTS
  che
  che-parent

Problems
>_ @workspace6id8cuu8ldpan6ry/projects x

[user@workspace6id8cuu8ldpan6ry projects]$ git clone https://github.com/eclipse/che-parent/
Cloning into 'che-parent'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 1197 (delta 13), reused 18 (delta 6), pack-reused 1169
Receiving objects: 100% (1197/1197), 253.71 KiB | 0 bytes/s, done.
Resolving deltas: 100% (665/665), done.
[user@workspace6id8cuu8ldpan6ry projects]$ ls -la
total 0
drwxrwx---. 4 root root 35 Jun 21 20:18 .
drwxr-xr-x. 1 root root 62 Jun 21 20:14 ..
drwxr-xr-x. 23 user root 594 Jun 21 19:55 che
drwxr-xr-x. 5 user root 118 Jun 21 20:16 che-parent
[user@workspace6id8cuu8ldpan6ry projects]$

```



## NOTE

Importing or deleting workspace projects in the terminal does not update the workspace configuration, and the change is not reflected in the **Project** and **Devfile** tabs in the dashboard.

Similarly, if you add a project using the **Dashboard**, then delete it with **rm -fr myproject**, it may still appear in the **Projects** or **Devfile** tab.

## 3.8. CONFIGURING WORKSPACE EXPOSURE STRATEGIES

The following section describes how to configure workspace exposure strategies of a CodeReady Workspaces server and ensure that applications running inside are not vulnerable to outside attacks.

The workspace exposure strategy is configured per CodeReady Workspaces server, using the **che.infra.kubernetes.server\_strategy** configuration property or the **CHE\_INFRA\_KUBERNETES\_SERVER\_STRATEGY** environment variable.

The supported values for **che.infra.kubernetes.server\_strategy** are:

- **multi-host**

For the multi-host strategy, set the **che.infra.kubernetes.ingress.domain** (or the **CHE\_INFRA\_KUBERNETES\_INGRESS\_DOMAIN** environment variable) configuration property to the domain name that will host workspace component subdomains.

### 3.8.1. Workspace exposure strategies

Specific components of workspaces need to be made accessible outside of the OpenShift cluster. This is typically the user interface of the workspace's IDE, but it can also be the web UI of the application being developed. This enables developers to interact with the application during the development process.

CodeReady Workspaces supports three ways to make workspace components available to the users, also referred to as *strategies*:

- multi-host strategy



The strategies define whether new subdomains are created for components of the workspace, and what hosts these components are available on.

### 3.8.1.1. Multi-host strategy

With this strategy, each workspace component is assigned a new subdomain of the main domain configured for the CodeReady Workspaces server. On OpenShift, this is the only possible strategy, and manual configuration of the workspace exposure strategy is therefore always ignored.

This strategy is the easiest to understand from the perspective of component deployment because any paths present in the URL to the component are received as they are by the component.

On a CodeReady Workspaces server secured using the Transport Layer Security (TLS) protocol, creating new subdomains for each component of each workspace requires a wildcard certificate to be available for all such subdomains for the CodeReady Workspaces deployment to be practical.

## 3.8.2. Security considerations

This section explains the security impact of using different CodeReady Workspaces workspace exposure strategies.

All the security-related considerations in this section are only applicable to CodeReady Workspaces in multiuser mode. The single user mode does not impose any security restrictions.

### 3.8.2.1. JSON web token (JWT) proxy

All CodeReady Workspaces plug-ins, editors, and components can require authentication of the user accessing them. This authentication is performed using a JSON web token (JWT) proxy that functions as a reverse proxy of the corresponding component, based on its configuration, and performs the authentication on behalf of the component.

The authentication uses a redirect to a special page on the CodeReady Workspaces server that propagates the workspace and user-specific authentication token (workspace access token) back to the originally requested page.

The JWT proxy accepts the workspace access token from the following places in the incoming requests, in the following order:

1. The token query parameter
2. The Authorization header in the bearer-token format
3. The **access\_token** cookie

### 3.8.2.2. Secured plug-ins and editors

CodeReady Workspaces users do not need to secure workspace plug-ins and workspace editors (such as Che-Theia). This is because the JWT proxy authentication is transparent to the user and is governed by the plug-in or editor definition in their **meta.yaml** descriptors.

### 3.8.2.3. Secured container-image components

Container-image components can define custom endpoints for which the devfile author can require CodeReady Workspaces-provided authentication, if needed. This authentication is configured using two optional attributes of the endpoint:

- **secure** - A boolean attribute that instructs the CodeReady Workspaces server to put the JWT proxy in front of the endpoint. Such endpoints have to be provided with the workspace access token in one of the several ways explained in [Section 3.8.2.1, "JSON web token \(JWT\) proxy"](#). The default value of the attribute is **false**.
- **cookiesAuthEnabled** - A boolean attribute that instructs the CodeReady Workspaces server to automatically redirect the unauthenticated requests for current user authentication as described in [Section 3.8.2.1, "JSON web token \(JWT\) proxy"](#). Setting this attribute to **true** has security consequences because it makes Cross-site request forgery (CSRF) attacks possible. The default value of the attribute is **false**.

#### 3.8.2.4. Cross-site request forgery attacks

Cookie-based authentication can make an application secured by a JWT proxy prone to Cross-site request forgery (CSRF) attacks. See the [Cross-site request forgery](#) Wikipedia page and other resources to ensure your application is not vulnerable.

#### 3.8.2.5. Phishing attacks

An attacker who is able to create an Ingress or route inside the cluster with the workspace that shares the host with some services behind a JWT proxy, the attacker may be able to create a service and a specially forged Ingress object. When such a service or Ingress is accessed by a legitimate user that was previously authenticated with a workspace, it can lead to the attacker stealing the workspace access token from the cookies sent by the legitimate user's browser to the forged URL. To eliminate this attack vector, configure OpenShift to disallow setting the host of an Ingress.

## 3.9. MOUNTING A SECRET AS A FILE OR AN ENVIRONMENT VARIABLE INTO A WORKSPACE CONTAINER

Secrets are OpenShift objects that store sensitive data such as user names, passwords, authentication tokens, and configurations in an encrypted form.

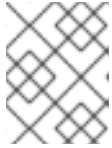
Users can mount a secret that contains sensitive data in a workspace container. This reapplies the stored data from the secret automatically for every newly created workspace. As a result, the user does not have to provide these credentials and configuration settings manually.

The following section describes how to automatically mount an OpenShift secret in a workspace container and create permanent mount points for components such as:

- Maven configuration, the **settings.xml** file
- SSH key pairs
- AWS authorization tokens

A OpenShift secret can be mounted into a workspace container as:

- A file - This creates automatically mounted Maven settings that will be applied to every new workspace with Maven capabilities.
- An environment variable - This uses SSH key pairs and AWS authorization tokens for automatic authentication.

**NOTE**

SSH key pairs can also be mounted as a file, but this format is primarily aimed at the settings of the Maven configuration.

The mounting process uses the standard OpenShift mounting mechanism, but it requires additional annotations and labeling for a proper bound of a secret with the required CodeReady Workspaces workspace container.

### 3.9.1. Mounting a secret as a file into a workspace container

**WARNING**

Red Hat CodeReady Workspaces uses OpenShift VolumeMount **subPath** feature to mount files into containers. This is supported and enabled by default since OpenShift v1.15 and OpenShift 4.

This section describes how to mount a secret from the user's project as a file in single-workspace or multiple-workspace containers of CodeReady Workspaces.

**Prerequisites**

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) CodeReady Workspaces 'quick-starts'.

**Procedure**

1. Create a new OpenShift secret in the OpenShift project where a CodeReady Workspaces workspace will be created.
  - The labels of the secret that is about to be created must match the set of labels configured in **che.workspace.provision.secret.labels** property of CodeReady Workspaces. The default labels are:
    - **app.kubernetes.io/part-of: che.eclipse.org**
    - **app.kubernetes.io/component: workspace-secret:**

**NOTE**

Note that the following example describes variations in the usage of the **target-container** annotation in versions 2.1 and 2.2 of Red Hat CodeReady Workspaces.

**Example:**

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: mvn-settings-secret
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspace-secret
...

```

Annotations must indicate the given secret is mounted as a file, provide the mount path, and, optionally, specify the name of the container in which the secret is mounted. If there is no target-container annotation, the secret will be mounted into all user containers of the CodeReady Workspaces workspace, but this is applicable *only for the CodeReady Workspaces version 2.1*.

```

apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
annotations:
  che.eclipse.org/target-container: maven
  che.eclipse.org/mount-path: /home/user/.m2/
  che.eclipse.org/mount-as: file
labels:
...

```

Since the CodeReady Workspaces version 2.2, the **target-container** annotation is deprecated and **automount-workspace-secret** annotation with Boolean values is introduced. Its purpose is to define the default secret mounting behavior, with the ability to be overridden in a devfile. The **true** value enables the automatic mounting into all workspace containers. In contrast, the **false** value disables the mounting process until it is explicitly requested in a devfile component using the **automountWorkspaceSecrets:true** property.

```

apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
annotations:
  che.eclipse.org/automount-workspace-secret: true
  che.eclipse.org/mount-path: /home/user/.m2/
  che.eclipse.org/mount-as: file
labels:
...

```

Data of the Kubernetes secret may contain several items, whose names must match the desired file name mounted into the container.

```

apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
labels:
  app.kubernetes.io/part-of: che.eclipse.org
  app.kubernetes.io/component: workspace-secret
annotations:
  che.eclipse.org/automount-workspace-secret: true

```

```

che.eclipse.org/mount-path: /home/user/.m2/
che.eclipse.org/mount-as: file
data:
settings.xml: <base64 encoded data content here>

```

This results in a file named **settings.xml** being mounted at the **/home/user/.m2/** path of all workspace containers.

The secret-s mount path can be overridden for specific components of the workspace using devfile. To change mount path, an additional volume should be declared in a component of the devfile, with name matching overridden secret name, and desired mount path.

```

apiVersion: 1.0.0
metadata:
...
components:
- type: dockerimage
  alias: maven
  image: maven:3.11
  volumes:
  - name: <secret-name>
    containerPath: /my/new/path
...

```

Note that for this kind of overrides, components must declare an alias to be able to distinguish containers which belong to them and apply override path exclusively for those containers.

### 3.9.2. Mounting a secret as an environment variable into a workspace container

The following section describes how to mount a OpenShift secret from the user's project as an environment variable, or variables, into single-workspace or multiple-workspace containers of CodeReady Workspaces.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).

#### Procedure

1. Create a new OpenShift secret in the k8s project where a CodeReady Workspaces workspace will be created.
  - The labels of the secret that is about to be created must match the set of labels configured in **che.workspace.provision.secret.labels** property of CodeReady Workspaces. By default, it is a set of two labels:
    - **app.kubernetes.io/part-of: che.eclipse.org**
    - **app.kubernetes.io/component: workspace-secret:**

**NOTE**

Note that the following example describes variations in the usage of the **target-container** annotation in versions 2.1 and 2.2 of Red Hat CodeReady Workspaces.

**Example:**

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspace-secret
...
```

Annotations must indicate that the given secret is mounted as an environment variable, provides variable names, and optionally, specifies the container name where this mount will be applied. If there is no **target-container** annotation defined, the secret will be mounted into all user containers of the CodeReady Workspaces workspace, but this is applicable **only for the CodeReady Workspaces version 2.1**.

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/target-container: maven
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    ...
data:
  mykey: myvalue
```

This results in the environment variable named **FOO\_ENV** and the value **myvalue** being provisioned into the container named **maven**.

*Since the CodeReady Workspaces version 2.2*, the **target-container** annotation is deprecated and **automount-workspace-secret** annotation with Boolean values is introduced. Its purpose is to define the default secret mounting behavior, with the ability to be overridden in a devfile. The **true** value enables the automatic mounting into all workspace containers. In contrast, the **false** value disables the mounting process until it is explicitly requested in a devfile component using the **automountWorkspaceSecrets:true** property.

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
```

```

labels:
  ...
data:
  mykey: myvalue

```

This results in the environment variable named **FOO\_ENV** and the value **myvalue** being provisioned into all workspace containers.

If the secret provides more than one data item, the environment variable name must be provided for each of the data keys as follows:

```

apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
annotations:
  che.eclipse.org/automount-workspace-secret: true
  che.eclipse.org/mount-as: env
  che.eclipse.org/mykey_env-name: FOO_ENV
  che.eclipse.org/otherkey_env-name: OTHER_ENV
labels:
  ...
data:
  mykey: myvalue
  otherkey: othervalue

```

This results in two environment variables with names **FOO\_ENV**, **OTHER\_ENV**, and values **myvalue** and **othervalue**, being provisioned into all workspace containers.



#### NOTE

The maximum length of annotation names in a Kubernetes secret is 63 characters, where 9 characters are reserved for a prefix that ends with `/`. This acts as a restriction for the maximum length of the key that can be used for the secret.

### 3.9.3. The use of annotations in the process of mounting a secret into a workspace container

OpenShift annotations and labels are tools used by libraries, tools, and other clients, to attach arbitrary non-identifying metadata to OpenShift native objects.

Labels select objects and connect them to a collection that satisfies certain conditions, where annotations are used for non-identifying information that is not used by OpenShift objects internally.

This section describes OpenShift annotation values used in the process of OpenShift secret mounting in a CodeReady Workspaces workspace.

Annotations must contain items that help identify the proper mounting configuration. These items are:

- **che.eclipse.org/target-container:** *Valid till the version 2.1* The name of the mounting container. If the name is not defined, the secret mounts into all user's containers of the CodeReady Workspaces workspace.
- **che.eclipse.org/automount-workspace-secret:** *Introduced in the version 2.2.* The main

mount selector. When set to **true**, the secret mounts into all user's containers of the CodeReady Workspaces workspace. When set to **false**, the secret does not mount into containers by default. The value of this attribute can be overridden in devfile components, using the **automountWorkspaceSecrets** boolean property that gives more flexibility to workspace owners. This property requires an **alias** to be defined for the component that uses it.

- **che.eclipse.org/env-name**: The name of the environment variable that is used to mount a secret.
- **che.eclipse.org/mount-as**: This item describes if a secret will be mounted as an environmental variable or a file. Options: **env** or **file**.
- **che.eclipse.org/<mykeyName>-env-name: FOO\_ENV**: The name of the environment variable used when data contains multiple items. **mykeyName** is used as an example.



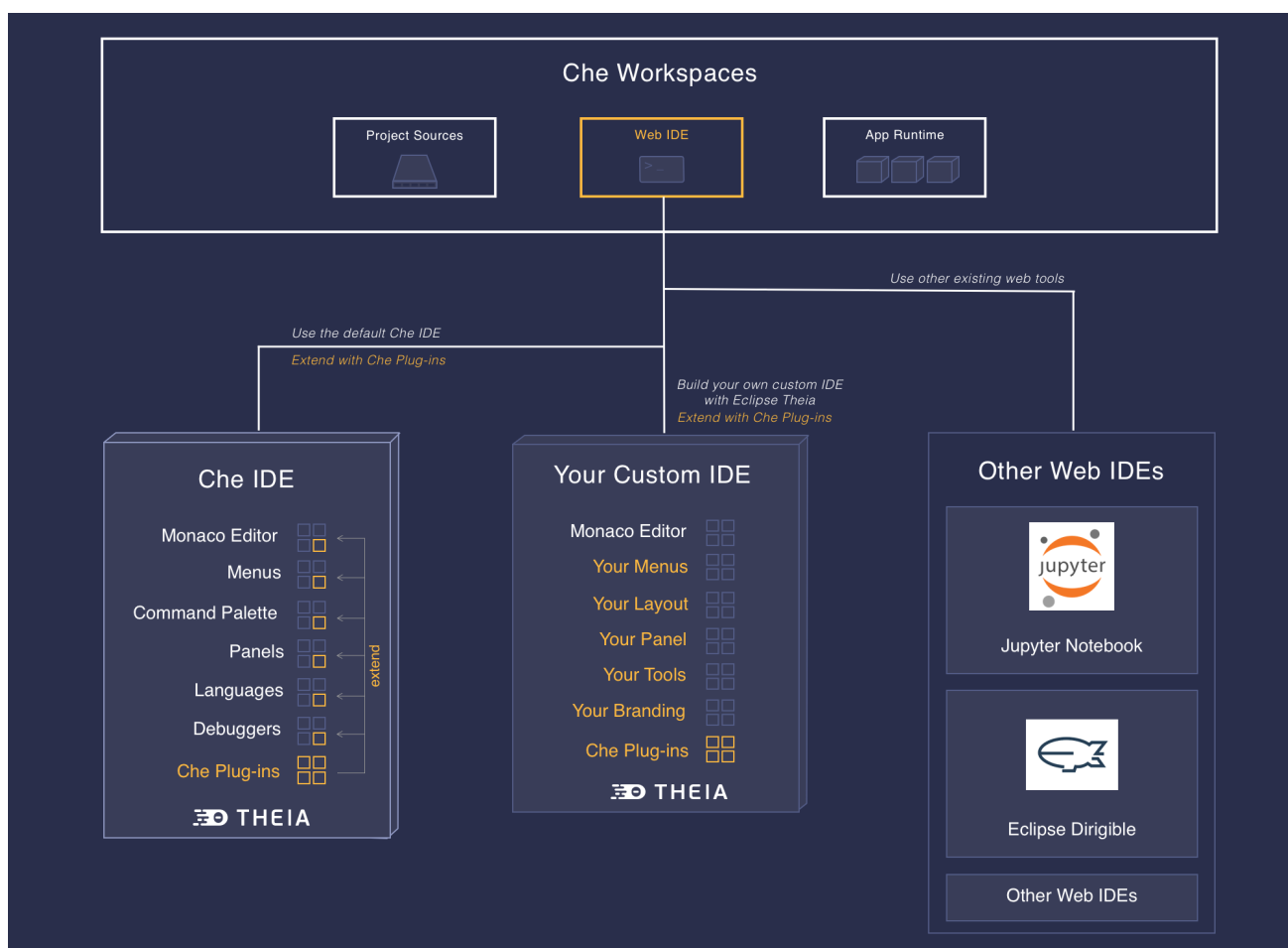
## CHAPTER 4. CUSTOMIZING DEVELOPER ENVIRONMENTS

Red Hat CodeReady Workspaces is an extensible and customizable developer-workspaces platform.

There are three different ways to extend Red Hat CodeReady Workspaces:

- **Alternative IDEs** provide specialized tools for Red Hat CodeReady Workspaces. For example, a Jupyter notebook for data analysis. Alternate IDEs can be based on Eclipse Theia or any other web IDE. The default IDE in Red Hat CodeReady Workspaces is Che-Theia.
- **Che-Theia plug-ins** add capabilities to the Che-Theia IDE. They rely on plug-in APIs that are compatible with Visual Studio Code. The plug-ins are isolated from the IDE itself. They can be packaged as files or as containers to provide their own dependencies.
- **Stacks** are pre-configured CodeReady Workspaces workspaces with a dedicated set of tools, which cover different developer personas. For example, it is possible to pre-configure a workbench for a tester with only the tools needed for their purposes.

Figure 4.1. CodeReady Workspaces extensibility



Extending Red Hat CodeReady Workspaces can be done entirely using Red Hat CodeReady Workspaces. Since version 7, Red Hat CodeReady Workspaces provides a self-hosting mode.

- [What is a Che-Theia plug-in](#)
- [Using alternative IDEs in CodeReady Workspaces](#)
- [Using a Visual Studio Code extension in CodeReady Workspaces](#)

## 4.1. WHAT IS A CHE-THEIA PLUG-IN

A Che-Theia plug-in is an extension of the development environment isolated from the IDE. Plug-ins can be packaged as files or containers to provide their own dependencies.

Extending Che-Theia using plug-ins can enable the following capabilities:

- **Language support:** Extend the supported languages by relying on the [Language Server Protocol](#).
- **Debuggers:** Extend debugging capabilities with the [Debug Adapter Protocol](#).
- **Development Tools:** Integrate your favorite linters, and as testing and performance tools.
- **Menus, panels, and commands:** Add your own items to the IDE components.
- **Themes:** Build custom themes, extend the UI, or customize icon themes.
- **Snippets, formatters, and syntax highlighting:** Enhance comfort of use with supported programming languages.
- **Keybindings:** Add new keymaps and popular keybindings to make the environment feel natural.

### 4.1.1. Features and benefits of Che-Theia plug-ins

Features	Description	Benefits
<b>Fast Loading</b>	Plug-ins are loaded at runtime and are already compiled. IDE is loading the plug-in code.	Avoid any compilation time. Avoid post-installation steps.
<b>Secure Loading</b>	Plug-ins are loaded separately from the IDE. The IDE stays always in a usable state.	Plug-ins do not break the whole IDE if it has bugs. Handle network issue.
<b>Tools Dependencies</b>	Dependencies for the plug-in are packaged with the plug-in in its own container.	No-installation for tools. Dependencies running into container.
<b>Code Isolation</b>	Guarantee that plug-ins cannot block the main functions of the IDE like opening a file or typing	Plug-ins are running into separate threads. Avoid dependencies mismatch.
<b>VS Code Extension Compatibility</b>	Extend the capabilities of the IDE with existing VS Code Extensions.	Target multiple platform. Allow easy discovery of Visual Studio Code Extension with required installation.

### 4.1.2. Che-Theia plug-in concept in detail

Red Hat CodeReady Workspaces provides a default web IDE for workspaces: Che-Theia. It is based on Eclipse Theia. It is a slightly different version than the plain Eclipse Theia one because there are functionalities that have been added based on the nature of the Red Hat CodeReady Workspaces

workspaces. This version of Eclipse Theia for CodeReady Workspaces is called **Che-Theia**.

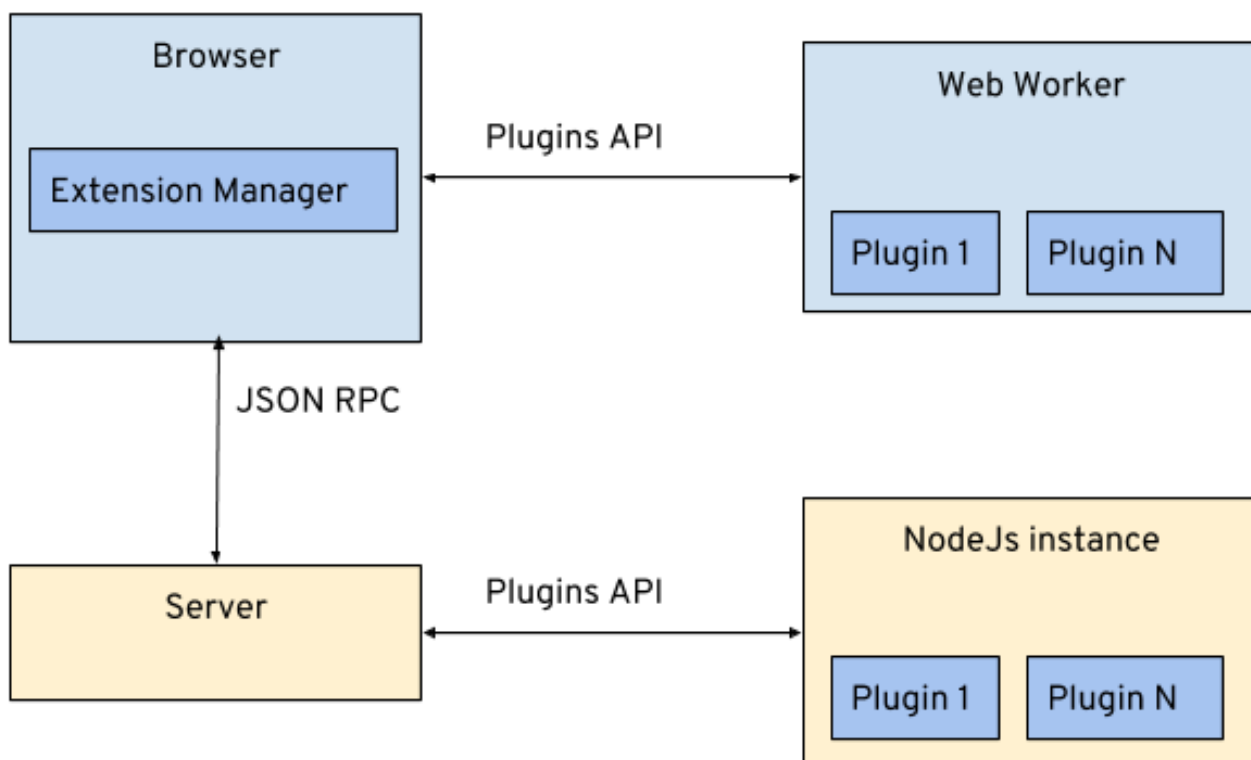
You can extend the IDE provided with Red Hat CodeReady Workspaces by building a **Che-Theia plug-in**. Che-Theia plug-ins are compatible with any other Eclipse Theia-based IDE.

#### 4.1.2.1. Client-side and server-side Che-Theia plug-ins

The Che-Theia editor plug-ins let you add languages, debuggers, and tools to your installation to support your development workflow. Plug-ins run when the editor completes loading. If a Che-Theia plug-in fails, the main Che-Theia editor continues to work.

Che-Theia plug-ins run either on the client side or on the server side. This is a scheme of the client and server-side plug-in concept:

Figure 4.2. Client and server-side Che-Theia plug-ins



The same Che-Theia plug-in API is exposed to plug-ins running on the client side (Web Worker) or the server side (Node.js).

#### 4.1.2.2. Che-Theia plug-in APIs

For the purpose of providing tool isolation and easy extensibility in Red Hat CodeReady Workspaces, the Che-Theia IDE has a set of plug-in APIs. The APIs are compatible with Visual Studio Code extension APIs. Usually, Che-Theia can run VS Code extensions as its own plug-ins.

When developing a plug-in that depends on or interacts with components of CodeReady Workspaces workspaces (containers, preferences, factories), use the CodeReady Workspaces APIs embedded in Che-Theia.

#### 4.1.2.3. Che-Theia plug-in capabilities

Che-Theia plug-ins have the following capabilities:

Plug-in	Description	Repository
<b>CodeReady Workspaces Extended Tasks</b>	Handles the CodeReady Workspaces commands and provides the ability to start those into a specific container of the workspace.	
<b>CodeReady Workspaces Extended Terminal</b>	Allows to provide terminal for any of the containers of the workspace.	
<b>CodeReady Workspaces Factory</b>	Handles the Red Hat CodeReady Workspaces Factories	
<b>CodeReady Workspaces Container</b>	Provides a container view that shows all the containers that are running in the workspace and allows to interact with them.	<a href="#">Containers plugins</a>
<b>Dashboard</b>	Integrates the IDE with the <b>Dashboard</b> and facilitate the navigation.	
<b>CodeReady Workspaces APIs</b>	Extends the IDE APIs to allow interacting with CodeReady Workspaces-specific components (workspaces, preferences).	

#### 4.1.2.4. VS Code extensions and Eclipse Theia plug-ins

A Che-Theia plug-in can be based on a VS Code extension or an Eclipse Theia plug-in.

##### A Visual Studio Code extension

To repackage a VS Code extension as a Che-Theia plug-in with its own set of dependencies, package the dependencies into a container. This ensures that Red Hat CodeReady Workspaces users do not need to install the dependencies when using the extension. See [Using a Visual Studio Code extension in CodeReady Workspaces](#).

##### An Eclipse Theia plug-in

You can build a Che-Theia plug-in by implementing an Eclipse Theia plug-in and packaging it to Red Hat CodeReady Workspaces.

##### Additional resources

- [Section 4.1.5, “Embedded and remote Che-Theia plug-ins”](#)

#### 4.1.3. Che-Theia plug-in metadata

Che-Theia plug-in metadata is information about individual plug-ins for the plug-in registry.

The Che-Theia plug-in metadata, for each specific plug-in, is defined in a **meta.yaml** file.

The [che-plugin-registry repository](#) contains .

Table 4.1. `meta.yml`

<b>apiVersion</b>	API version ( <code>`v2`</code> and higher)
<b>category</b>	Available: <b>Language, Other</b>
<b>description</b>	Description (a phrase)
<b>displayName</b>	Display name
<b>firstPublicationDate</b>	Date in the form <b>"YYYY-MM-DD"</b> Example: <b>"2019-12-02"</b>
<b>icon</b>	URL of an SVG icon
<b>name</b>	Name (no spaces allowed)
<b>publisher</b>	Name of the publisher
<b>repository</b>	URL of the source repository
<b>title</b>	Title (long)
<b>type</b>	<b>Che Plugin, VS Code extension</b>
<b>version</b>	Version information, for example: <b>7.5.1</b>
<b>spec</b>	Specifications (see underneath)

Table 4.2. `spec` attributes

<b>endpoints</b>	Plug-in endpoints
<b>containers</b>	Sidecar containers for the plug-in. <b>Che Plugin</b> and <b>VS Code extension</b> supports only one container
<b>initContainers</b>	Sidecar init containers for the plug-in
<b>workspaceEnv</b>	Environment variables for the workspace
<b>extensions</b>	Optional attribute required for VS Code and Che-Theia plug-ins. A list of URLs to plug-in artefacts, such as <b>.vsix</b> or <b>.theia</b> files

Table 4.3. `spec.containers`. Notice: `spec.initContainers` has absolutely the same container definition.

<b>name</b>	Sidecar container name
<b>image</b>	Absolute or relative container image URL
<b>memoryLimit</b>	OpenShift memory limit string, for example <b>512Mi</b>
<b>memoryRequest</b>	OpenShift memory request string, for example <b>512Mi</b>
<b>cpuLimit</b>	OpenShift CPU limit string, for example <b>2500m</b>
<b>cpuRequest</b>	OpenShift CPU request string, for example <b>125m</b>
<b>env</b>	List of environment variables to set in the sidecar
<b>command</b>	String array definition of the root process command in the container
<b>args</b>	String array arguments for the root process command in the container
<b>volumes</b>	Volumes required by the plug-in
<b>ports</b>	Ports exposed by the plug-in (on the container)
<b>commands</b>	Development commands available to the plug-in container
<b>mountSources</b>	Boolean flag to bound volume with source code <b>/projects</b> to the plug-in container

Table 4.4. **spec.containers.env** (and **spec.initContainers.env**) attributes. Notice: **workspaceEnv** has absolutely the same attributes

<b>name</b>	Environment variable name
<b>value</b>	Environment variable value

Table 4.5. **spec.containers.volumes** (and **spec.initContainers.volumes**) attributes

<b>mountPath</b>	Path to the volume in the container
<b>name</b>	Volume name
<b>ephemeral</b>	If true, the volume is ephemeral, otherwise the volume is persisted

Table 4.6. `spec.containers.ports` (and `spec.initContainers.ports`) attributes

<code>exposedPort</code>	Exposed port
--------------------------	--------------

Table 4.7. `spec.containers.commands` (and `spec.initContainers.commands`) attributes

<code>name</code>	Command name
<code>workingDir</code>	Command working directory
<code>command</code>	String array that defines the development command

Table 4.8. `spec.endpoints` attributes

<code>name</code>	Name (no spaces allowed)
<code>public</code>	<b>true, false</b>
<code>targetPort</code>	Target port
<code>attributes</code>	Endpoint attributes

Table 4.9. `spec.endpoints.attributes` attributes

<code>protocol</code>	Protocol, example: <b>ws</b>
<code>type</code>	<b>ide, ide-dev</b>
<code>discoverable</code>	<b>true, false</b>
<code>secure</code>	<b>true, false</b> . If <b>true</b> , then the endpoint is assumed to listen solely on <b>127.0.0.1</b> and is exposed using a JWT proxy
<code>cookiesAuthEnabled</code>	<b>true, false</b>

### Example `meta.yaml` for a Che-Theia plug-in: the CodeReady Workspaces machine-exec Service

```

apiVersion: v2
category: Other
description: Che Plugin with che-machine-exec service to provide creation terminal or tasks for Red Hat CodeReady Workspaces workspace containers
displayName: CodeReady Workspaces machine-exec Service
firstPublicationDate: "2019-12-02"
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
name: che-machine-exec-plug-in
publisher: eclipse

```

```

repository: https://github.com/eclipse/che-machine-exec/
title: Che machine-exec Service Plugin
type: Che Plugin
version: 7.5.1
spec:
  endpoints:
    - name: "che-machine-exec"
      public: true
      targetPort: 4444
      attributes:
        protocol: ws
        type: terminal
        discoverable: false
        secure: true
        cookiesAuthEnabled: true
  containers:
    - name: che-machine-exec
      image: "quay.io/eclipse/che-machine-exec:7.5.1"
      ports:
        - exposedPort: 4444

```

### Example meta.yaml for a VisualStudio Code extension: the AsciiDoc support extension

```

apiVersion: v2
category: Language
description: This extension provides a live preview, syntax highlighting and snippets for the AsciiDoc
format using AsciiDoctor flavor
displayName: AsciiDoc support
firstPublicationDate: "2019-12-02"
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
name: vscode-asciidoctor
publisher: joaompinto
repository: https://github.com/asciidoctor/asciidoctor-vscode
title: AsciiDoctor Plug-in
type: VS Code extension
version: 2.7.7
spec:
  extensions:
    - https://github.com/asciidoctor/asciidoctor-vscode/releases/download/v2.7.7/asciidoctor-vscode-
      2.7.7.vsix

```

#### 4.1.4. Che-Theia plug-in lifecycle

When a user is starting a workspace, the following procedure is followed:

1. CodeReady Workspaces master checks for plug-ins to start from the workspace definition.
2. Plug-in metadata is retrieved, and the type of each plug-in is recognized.
3. A broker is selected according to the plug-in type.
4. The broker processes the installation and deployment of the plug-in (the installation process is different for each broker).

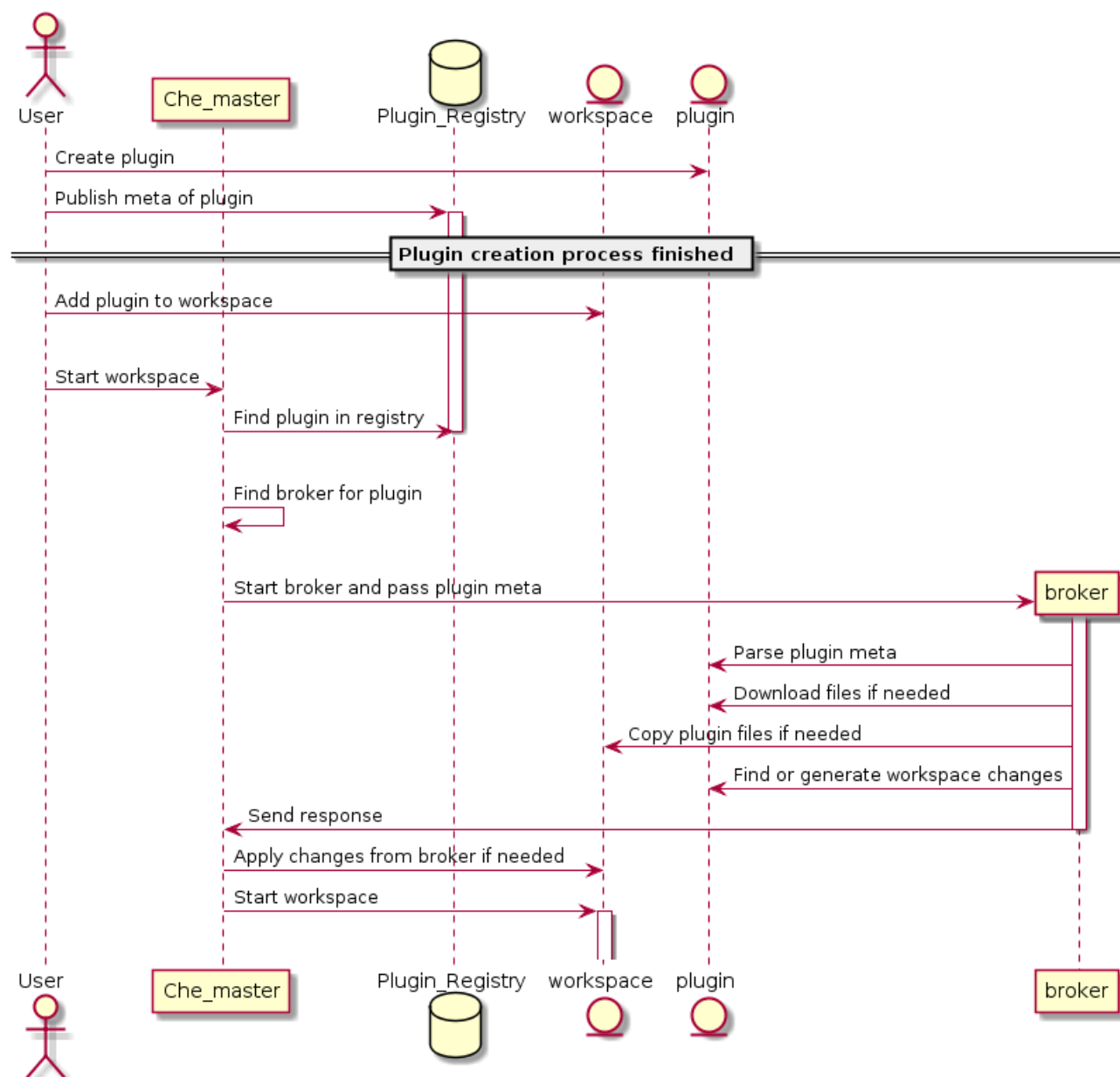




## NOTE

Different types of plug-ins exist. A broker ensures all installation requirements are met for a plug-in to deploy correctly.

Figure 4.3. Che-Theia plug-in lifecycle



Before a CodeReady Workspaces workspace is launched, CodeReady Workspaces master starts containers for the workspace:

1. The Che-Theia plug-in broker extracts the plug-in (from the **.theia** file) to get the sidecar containers that the plug-in needs.
2. The broker sends the appropriate container information to CodeReady Workspaces master.
3. The broker copies the Che-Theia plug-in to a volume to have it available for the Che-Theia editor container.
4. CodeReady Workspaces workspace master then starts all the containers of the workspace.
5. Che-Theia is started in its own container and checks the correct folder to load the plug-ins.

Che-Theia plug-in lifecycle:

1. When a user is opening a browser tab or window with Che-Theia, Che-Theia starts a new plug-in session (browser or remote **TODO: 'what is remote in this context?'**). Every Che-Theia plug-in is notified that a new session has been started (the **start()** function of the plug-in triggered).
2. A Che-Theia plug-in session is running and interacting with the Che-Theia back end and frontend.
3. When the user is closing the browser tab or there is a timeout, every plug-in is notified (the **stop()** function of the plug-in triggered).

### 4.1.5. Embedded and remote Che-Theia plug-ins

Developer workspaces in Red Hat CodeReady Workspaces provide all dependencies needed to work on a project. The application includes the dependencies needed by all the tools and plug-ins used.

There are two different ways a Che-Theia plug-in can run. This is based on the dependencies that are needed for the plug-in: **embedded** (or local) and **remote**.

#### 4.1.5.1. Embedded (or local) plug-ins

The plug-in does not have specific dependencies - it only uses a Node.js runtime, and it runs in the same container as the IDE. The plug-in is injected into the IDE.

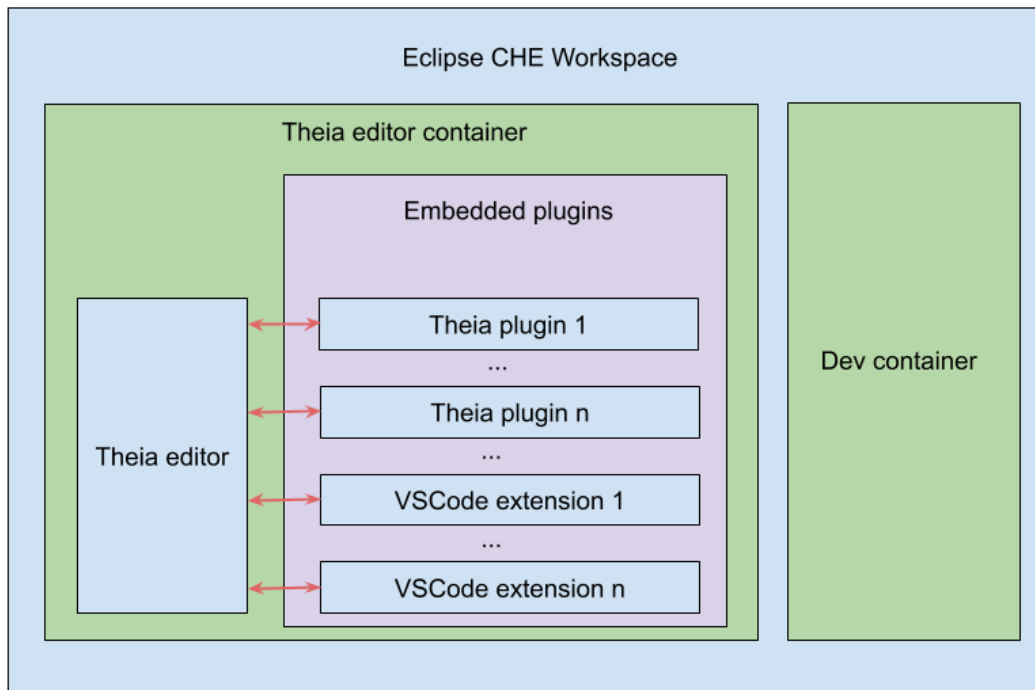
Examples:

- Code linting
- New set of commands
- New UI components

To include a Che-Theia plug-in as embedded, define a URL to the plug-in binary file (the **.theia** archive) in the **meta.yaml** file. In the case of a VS Code extension, provide the extension ID from the Visual Studio Code marketplace (see [Using a Visual Studio Code extension in CodeReady Workspaces](#) ).

When starting a workspace, CodeReady Workspaces downloads and unpacks the plug-in binaries and includes them in the Che-Theia editor container. The Che-Theia editor initializes the plug-ins when it starts.

Figure 4.4. Local Che-Theia plug-in



#### 4.1.5.2. Remote plug-ins

The plug-in relies on dependencies or it has a back end. It runs in its own sidecar container, and all dependencies are packaged in the container.

A remote Che-Theia plug-in consist of two parts:

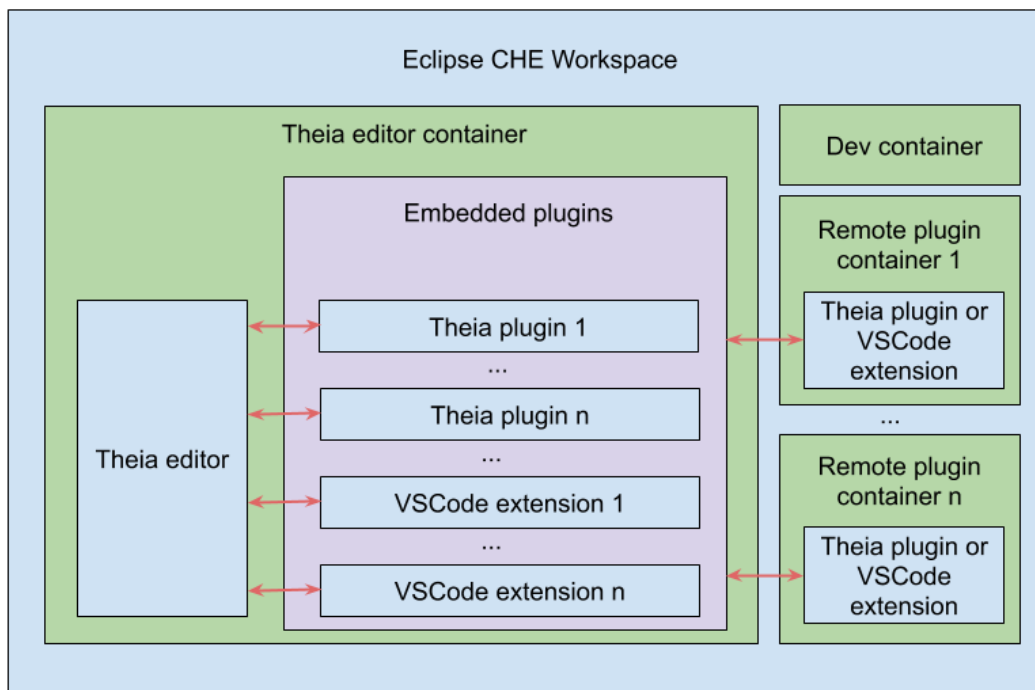
- Che-Theia plug-in or VS Code extension binaries. The definition in the **meta.yaml** file is the same as for embedded plug-ins.
- Container image definition, for example, **eclipse/che-theia-dev:nightly**. From this image, CodeReady Workspaces creates a separate container inside a workspace.

Examples:

- Java Language Server
- Python Language Server

When starting a workspace, CodeReady Workspaces creates a container from the plug-in image, downloads and unpacks the plug-in binaries, and includes them in the created container. The Che-Theia editor connects to the remote plug-ins when it starts.

Figure 4.5. Remote Che-Theia plug-in



### 4.1.5.3. Comparison matrix

When a Che-Theia plug-in (or a VS Code extension) does not need extra dependencies inside its container, it is an embedded plug-in. A container with extra dependencies that includes a plug-in is a remote plug-in.

Table 4.10. Che-Theia plug-in comparison matrix: embedded vs remote

	Configure RAM per plug-in	Environment dependencies	Create separated container
<b>Remote</b>	TRUE	Plug-in uses dependencies defined in the remote container.	TRUE
<b>Embedded</b>	FALSE (users can configure RAM for the whole editor container, but not per plug-in)	Plug-in uses dependencies from the editor container; if container does not include these dependencies, the plug-in fails or does not function as expected.	FALSE

Depending on your use case and the capabilities provided by your plug-in, select one of the described running modes.

### 4.1.6. Remote plug-in endpoint

Red Hat CodeReady Workspaces has a remote plug-in endpoint service to start VS Code Extensions and Che-Theia plug-ins in separate containers. Red Hat CodeReady Workspaces injects the remote plug-in endpoint binaries into each remote plug-in container. This service starts remote extensions and plug-ins defined in the plug-in **meta.yaml** file and connects them to the Che-Theia editor container.

The remote plug-in endpoint creates a plug-in API proxy between the remote plug-in container and the Che-Theia editor container. The remote plug-in endpoint is also an interceptor for some plug-in API parts, which it launches inside a remote sidecar container rather than an editor container. Examples: terminal API, debug API.

The remote plug-in endpoint executable command is stored in the environment variable of the remote plug-in container: **PLUGIN\_REMOTE\_ENDPOINT\_EXECUTABLE**.

Red Hat CodeReady Workspaces provides two ways to start the remote plug-in endpoint with a sidecar image:

- Defining a launch remote plug-in endpoint using a Dockerfile. To use this method, patch an original image and rebuild it.
- Defining a launch remote plug-in endpoint in the plug-in **meta.yaml** file. Use this method to avoid patching an original image.

#### 4.1.6.1. Defining a launch remote plug-in endpoint using Dockerfile

To start a remote plug-in endpoint, use the **PLUGIN\_REMOTE\_ENDPOINT\_EXECUTABLE** environment variable in the Dockerfile.

##### Procedure

- Start a remote plug-in endpoint using the **CMD** command in the Dockerfile:

##### Dockerfile example

```
FROM fedora:30

RUN dnf update -y && dnf install -y nodejs htop && node -v

RUN mkdir /home/user

ENV HOME=/home/user

RUN mkdir /projects \
  && chmod -R g+rwX /projects \
  && chmod -R g+rwX "${HOME}"

CMD ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

- Start a remote plug-in endpoint using the **ENTRYPOINT** command in the Dockerfile:

##### Dockerfile example

```

FROM fedora:30

RUN dnf update -y && dnf install -y nodejs htop && node -v

RUN mkdir /home/user

ENV HOME=/home/user

RUN mkdir /projects \
  && chmod -R g+rwX /projects \
  && chmod -R g+rwX "${HOME}"

ENTRYPOINT ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}

```

#### 4.1.6.1.1. Using a wrapper script

Some images use a wrapper script to configure permissions. The script is defined in the **ENTRYPOINT** command of the Dockerfile to configure permissions inside the container, and it script executes a main process defined in the **CMD** command of the Dockerfile.

Red Hat CodeReady Workspaces uses such images with a wrapper script to provide permission configurations on different infrastructures with advanced security, for example on OpenShift.

- Example of a wrapper script:

```

#!/bin/sh

set -e

export USER_ID=$(id -u)
export GROUP_ID=$(id -g)

if ! whoami >/dev/null 2>&1; then
  echo "${USER_NAME:-user}:x:${USER_ID}:0:${USER_NAME:-user}
user:${HOME}:/bin/sh" >> /etc/passwd
fi

# Grant access to projects volume in case of non root user with sudo rights
if [ "${USER_ID}" -ne 0 ] && command -v sudo >/dev/null 2>&1 && sudo -n true > /dev/null
2>&1; then
  sudo chown "${USER_ID}:${GROUP_ID}" /projects
fi

exec "$@"

```

- Example of a Dockerfile with a wrapper script:

#### Dockerfile example

```

FROM alpine:3.10.2

ENV HOME=/home/theia

RUN mkdir /projects ${HOME} && \
  # Change permissions to let any arbitrary user

```

```
for f in "${HOME}" "/etc/passwd" "/projects"; do \
  echo "Changing permissions on ${f}" && chgrp -R 0 ${f} && \
  chmod -R g+rwX ${f}; \
done
```

```
ADD entrypoint.sh /entrypoint.sh
```

```
ENTRYPOINT [ "/entrypoint.sh" ]
CMD ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

In this example, the container launches the `/entrypoint.sh` script defined in the **ENTRYPOINT** command of the Dockerfile. The script configures the permissions and executes the command using **exec \$@**. **CMD** is the argument for **ENTRYPOINT**, and the **exec \$@** command calls **\${PLUGIN\_REMOTE\_ENDPOINT\_EXECUTABLE}**. A remote plug-in endpoint then starts in the container after permission configuration.

#### 4.1.6.2. Defining a launch remote plug-in endpoint in a `meta.yaml` file

Use this method to re-use images to start remote a plug-in endpoint without modifications.

##### Procedure

Modify the plug-in `meta.yaml` file properties **command** and **args**:

- **command** - Red Hat CodeReady Workspaces uses to override **Dockerfile#ENTRYPOINT**.
- **args** - Red Hat CodeReady Workspaces uses to override **Dockerfile#CMD**.
- Example of a YAML file with the **command** and **args** properties modified:

```
apiVersion: v2
category: Language
description: "Typescript language features"
displayName: Typescript
firstPublicationDate: "2019-10-28"
icon: "https://www.eclipse.org/che/images/logo-eclipseche.svg"
name: typescript
publisher: che-incubator
repository: "https://github.com/Microsoft/vscode"
title: "Typescript language features"
type: "VS Code extension"
version: remote-bin-with-override-entrypoint
spec:
  containers:
    - image: "example/fedora-for-ts-remote-plugin-without-endpoint:latest"
      memoryLimit: 512Mi
      name: vscode-typescript
      command:
        - sh
        - -c
      args:
        - ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
  extensions:
    - "https://github.com/che-incubator/ms-code.typescript/releases/download/v1.35.1/che-typescript-language-1.35.1.vsix"
```

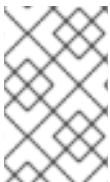
- Modify **args** instead of **command** to use an image with a wrapper script pattern and to keep a call of the **entrypoint.sh** script:

```

apiVersion: v2
category: Language
description: "Typescript language features"
displayName: Typescript
firstPublicationDate: "2019-10-28"
icon: "https://www.eclipse.org/che/images/logo-eclipseche.svg"
name: typescript
publisher: che-incubator
repository: "https://github.com/Microsoft/vscode"
title: "Typescript language features"
type: "VS Code extension"
version: remote-bin-with-override-entrypoint
spec:
  containers:
    - image: "example/fedora-for-ts-remote-plugin-without-endpoint:latest"
      memoryLimit: 512Mi
      name: vscode-typescript
      args:
        - sh
        - -c
        - ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
  extensions:
    - "https://github.com/che-incubator/ms-code.typescript/releases/download/v1.35.1/che-typescript-language-1.35.1.vsix"

```

Red Hat CodeReady Workspaces calls the **entrypoint.sh** wrapper script defined in the **ENTRYPOINT** command of the Dockerfile. The script executes [ **'sh', '-c', '\${PLUGIN\_REMOTE\_ENDPOINT\_EXECUTABLE}'** ] using the **exec "\$@"** command.



#### NOTE

To execute a service when starting the container and also to start a remote plug-in endpoint, use **meta.yaml** with modified **command** and **args** properties. Start the service, detach the process, and start the remote plug-in endpoint, and they then work in parallel.

## 4.2. USING ALTERNATIVE IDES IN CODEREADY WORKSPACES

Extending Red Hat CodeReady Workspaces developer workspaces using different IDEs (integrated development environments) enables:

- Re-purposing the environment for different use cases.
- Providing a dedicated custom IDE for specific tools.
- Providing different perspectives for individual users or groups of users.

Red Hat CodeReady Workspaces provides a default web IDE to be used with the developer workspaces. This IDE is completely decoupled. You can bring your own custom IDE for Red Hat CodeReady Workspaces:

- **Built from Eclipse Theia**, which is a framework to build web IDEs. Example: [Sirius on the web](#).



- **Completely different web IDEs**, such as Jupyter, Eclipse Dirigible, or others. Example: [Jupyter in Red Hat CodeReady Workspaces workspaces](#).

### Bringing custom IDE built from Eclipse Theia

- Creating your own custom IDE based on Eclipse Theia.
- Adding CodeReady Workspaces-specific tools to your custom IDE.
- Packaging your custom IDE into the available editors for CodeReady Workspaces.

### Bringing your completely different web IDE into CodeReady Workspaces

- Packaging your custom IDE into the available editors for CodeReady Workspaces.

## 4.3. USING A VISUAL STUDIO CODE EXTENSION IN CODEREADY WORKSPACES

Starting with Red Hat CodeReady Workspaces 2.2, Visual Studio Code (VS Code) extensions can be installed to extend the functionality of a CodeReady Workspaces workspace. VS Code extensions can run in the Che-Theia editor container, or they can be packaged in their own isolated and pre-configured containers with their prerequisites.

This document describes:

- Use of a VS Code extension in CodeReady Workspaces with workspaces.
- CodeReady Workspaces Plug-ins panel.
- How to publish a VS Code extension in the CodeReady Workspaces plug-in registry (to share the extension with other CodeReady Workspaces users).
  - The extension-hosting sidecar container and the use of the extension in a devfile are optional for this.
  - How to review the compatibility of the VS Code extensions to be informed whether a specific API is supported or has not been implemented yet.

### 4.3.1. Publishing a VS Code extension into the CodeReady Workspaces plug-in registry

The user of CodeReady Workspaces can use a workspace devfile to use any plug-in, also known as Visual Studio Code (VS Code) extension. This plug-in can be added to the plug-in registry, then easily reused by anyone in the same organization with access to that workspaces installation.

Some plug-ins need a runtime dedicated container for code compilation. This fact makes those plug-ins a combination of a runtime sidecar container and a VS Code extension.

The following section describes the portability of a plug-in configuration and associating an extension with a runtime container that the plug-in needs.

#### 4.3.1.1. Writing a meta.yaml file and adding it to a plug-in registry

The plug-in meta information is required to publish a VS Code extension in an Red Hat CodeReady Workspaces plug-in registry. This meta information is provided as a **meta.yaml** file. This section describes how to create a **meta.yaml** file for an extension.

## Procedure

1. Create a **meta.yaml** file in the following plug-in registry directory:  
**<apiVersion>/plugins/<publisher>/<plug-inName>/<plug-inVersion>/**.
2. Edit the **meta.yaml** file and provide the necessary information. The configuration file must adhere to the following structure:

```
apiVersion: v2
publisher: myorg
name: my-vscode-ext
version: 1.7.2
type: value
displayName:
title:
description:
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
repository:
category:
spec:
  containers:
    - image:
      memoryLimit:
      memoryRequest:
      cpuLimit:
      cpuRequest:
  extensions:
    - https://github.com/redhat-developer/vscode-yaml/releases/download/0.4.0/redhat.vscode-yaml-0.4.0.vsix
    - vscode:extension/SonarSource.sonarlint-vscode
```

- 1 Version of the file structure.
- 2 Name of the plug-in publisher. Must be the same as the publisher in the path.
- 3 Name of the plug-in. Must be the same as in path.
- 4 Version of the plug-in. Must be the same as in path.
- 5 Type of the plug-in. Possible values: **Che Plugin, Che Editor, Theia plugin, VS Code extension**.
- 6 A short name of the plug-in.
- 7 Title of the plug-in.
- 8 A brief explanation of the plug-in and what it does.
- 9 The link to the plug-in logo.

- 10 Optional. The link to the source-code repository of the plug-in.
- 11 Defines the category that this plug-in belongs to. Should be one of the following: **Editor**, **Debugger**, **Formatter**, **Language**, **Lint**, **Snippet**, **Theme**, or **Other**.
- 12 If this section is omitted, the VS Code extension is added into the Che-Theia IDE container.
- 13 The Docker image from which the sidecar container will be started. Example: **eclipse/che-theia-endpoint-runtime:next**.
- 14 The maximum RAM which is available for the sidecar container. Example: "512Mi". This value might be overridden by the user in the component configuration.
- 15 The RAM which is given for the sidecar container by default. Example: "256Mi". This value might be overridden by the user in the component configuration.
- 16 The maximum CPU amount in cores or millicores (suffixed with "m") which is available for the sidecar container. Examples: "500m", "2". This value might be overridden by the user in the component configuration.
- 17 The CPU amount in cores or millicores (suffixed with "m") which is given for the sidecar container by default. Example: "125m". This value might be overridden by the user in the component configuration.
- 18 A list of VS Code extensions run in this sidecar container.

### 4.3.2. Adding a plug-in registry VS Code extension to a workspace

When the required VS Code extension is added into a CodeReady Workspaces plug-in registry, the user can add it into the workspace through the **CodeReady Workspaces Plugins** panel or through the workspace configuration.

#### 4.3.2.1. Adding a VS Code extension using the CodeReady Workspaces Plugins panel

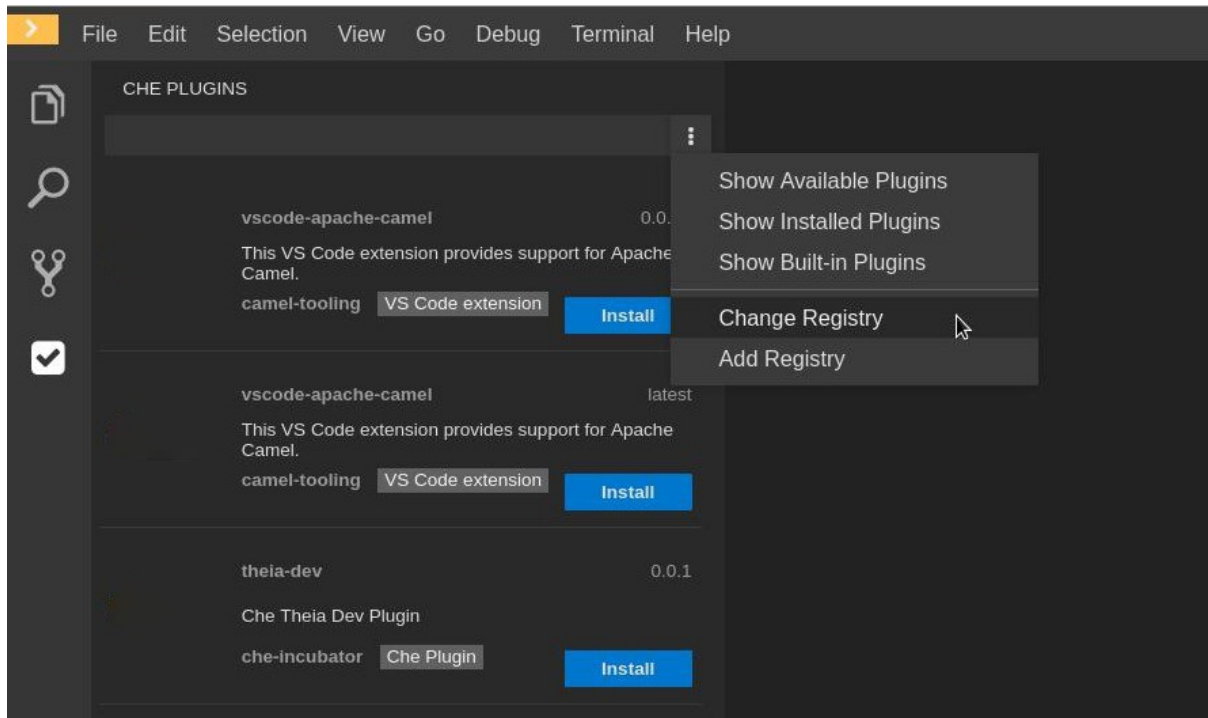
##### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#)

##### Procedure

To add a VS Code extension using the **CodeReady Workspaces Plugins** panel:

1. Open the **CodeReady Workspaces Plugins** panel by pressing **CTRL+SHIFT+J** or navigate to **View/Plugins**.
2. Change the current registry to the registry in which the VS Code extension was added.
3. In the search bar, click the **Menu** button and then click **Change Registry** to choose the registry from the list. If the required registry is not in the list, add it using the **Add Registry** menu option. The registry link points to the **plugins** segment of the registry, for example: **https://my-registry.com/v3/plugins/index.json**.



4. Search for the required plug-in using the filter, and then click the **Install** button.
5. Restart the workspace for the changes to take effect.

#### 4.3.2.2. Adding a VS Code extension using the workspace configuration

##### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) CodeReady Workspaces 'quick-starts'.
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces [Creating a workspace from user dashboard](#).

##### Procedure

To add a VS Code extension using the workspace configuration:

1. Click the **Workspaces** tab on the **Dashboard** and select the workspace in which you want to add the plug-in. The **Workspace <workspace-name>** window is opened showing the details of the workspace.
2. Click the **devfile** tab.
3. Locate the **components** section, and add a new entry with the following structure:

```
- type: chePlugin
  id: 1
```

- 1** Link to the **meta.yaml** file in your registry, for example, **<https://my-plugin-in-registry/v3/plugins/<publisher>/<plug-inName>/<plug-inVersion>/meta.yaml>**

CodeReady Workspaces automatically adds the other fields to the new component.

Alternatively, you can link to a **meta.yaml** file hosted on GitHub, using the dedicated reference field.

```
- type: chePlugin
  reference: 1
```

**1** **`https://raw.githubusercontent.com/<username>/<registryRepository>/v3/plugins/<publisher>/<plug-inName>/<plug-inVersion>/meta.yaml`**

4. Restart the workspace for the changes to take effect.

### 4.3.3. Choosing the sidecar image

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

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).

#### Procedure

To choose a sidecar image:

1. If the VS Code extension does not have any external dependencies, use **eclipse/che-theia-endpoint-runtime: next** as a sidecar container image for the extension.



## NOTE

In addition to the **eclipse/che-theia-endpoint-runtime:next** base image, the following ready-to-use sidecar images that include language-specific dependencies are available:

- `eclipse/che-remote-plugin-runner-java8`
- `eclipse/che-remote-plugin-runner-java11`
- `eclipse/che-remote-plugin-go-1.10.7`
- `eclipse/che-remote-plugin-python-3.7.3`
- `eclipse/che-remote-plugin-dotnet-2.2.105`
- `eclipse/che-remote-plugin-php7`
- `eclipse/che-remote-plugin-kubernetes-tooling-1.0.0`
- `eclipse/che-remote-plugin-kubernetes-tooling-0.1.17`
- `eclipse/che-remote-plugin-openshift-connector-0.0.17`
- `eclipse/che-remote-plugin-openshift-connector-0.0.21`

For a VS Code extension with external dependencies not found in one of the ready-to-use images, use a container image based on the **eclipse/che-theia-endpoint-runtime:next** image that contains the dependencies.

### Example

Base the **FROM** directive on **FROM eclipse/che-theia-endpoint-runtime:next**. This is required because the base image contains tools for running the remote VS Code extension and for communicating between the sidecar and the Che-Theia editor. This way, the VS Code extension operates as if it was not remote.

### 4.3.4. Verifying the VS Code extension API compatibility level

Che-Theia does not fully support the VS Code extensions API. The [vscode-theia-comparator](#) is used to analyze the compatibility between the Che-Theia plug-in API and the VS Code extension API. This tool runs in a nightly cycle, and the results are published on the [vscode-theia-comparator](#) GitHub page.

#### Prerequisites

- Personal GitHub access token. For information about creating a personal access token for the command line see [Creating a personal access token for the command line](#). A GitHub access token is required to increase the GitHub download limit for your IP address.

#### Procedure

To run the **vscode-theia comparator** manually:

1. Clone the [vscode-theia-comparator](#) repository, and build it using the **yarn** command.
2. Set the **GITHUB\_TOKEN** environment variable to your token.

3. Execute the **yarn run generate** command to generate a report.
4. Open the **out/status.html** file to view the report.

## 4.4. ADDING TOOLS TO CODEREADY WORKSPACES AFTER CREATING A WORKSPACE

When installed in the workspace, CodeReady Workspaces plug-ins bring new capabilities to the CodeReady Workspaces. Plug-ins consist of a Che-Theia plug-in, metadata, and a hosting container. These plug-ins may provide the following capabilities:

- Integrating with other systems, including OpenShift and OpenShift.
- Automating some developer tasks, such as formatting, refactoring, and running automated tests.
- Communicating with multiple databases directly from the IDE.
- Enhanced code navigation, auto-completion and error highlighting.

This chapter provides basic information about CodeReady Workspaces plug-ins installation, enabling, and use in CodeReady Workspaces workspaces.

- [Section 4.4.1, “Additional tools in the CodeReady Workspaces workspace”](#)
- [Section 4.4.2, “Adding language support plug-in to the CodeReady Workspaces workspace”](#)

### 4.4.1. Additional tools in the CodeReady Workspaces workspace

CodeReady Workspaces plug-ins are extensions to the Che-Theia IDE that come bundled with a container image that contains their native prerequisites (for example, the OpenShift Connector plug-in needs the **oc** command installed). A CodeReady Workspaces plug-in is a list of Che-Theia plug-ins together about a Linux container that the plug-in requires to run in. It can also include metadata to define the description, categorization tags, and an icon. CodeReady Workspaces provides a registry of plug-ins available for installation into the user’s workspace.

Because many VS Code extensions can run inside the Che-Theia IDE, they can be packaged as CodeReady Workspaces plug-ins by combining them with a runtime or a sidecar container. Users can choose from many more plug-ins that are provided out of the box.

From the Dashboard, plug-ins in the registry can be added from the **Plugin** tab or by adding them into a devfile. The devfile can also be used for further configuration of the plug-in, such as to define memory or CPU usage. Alternatively, plug-ins can be installed from CodeReady Workspaces by pressing **Ctrl+Shift+J** or by navigating to **View → Plugins**.

#### Additional resources

- [Adding components to a devfile](#)

### 4.4.2. Adding language support plug-in to the CodeReady Workspaces workspace

This procedure describes adding a tool to an already existing workspace, by enabling a dedicated plug-in from the Dashboard.

To add tools that are available as plug-ins into a CodeReady Workspaces workspace, use one of the following methods:

- [Enable the plug-in from the Dashboard \*\*Plugin\*\* tab.](#)
- [Edit the workspace devfile from the Dashboard \*\*Devfile\*\* tab.](#)

This procedure uses the **Java language support** plug-in as an example.

### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation GuideCodeReady Workspaces 'quick-starts'](#).
- An existing workspace defined in this instance of Red Hat CodeReady Workspaces; see:
  - [Creating and configuring a new CodeReady Workspaces workspace](#)
  - [Creating a workspace from User Dashboard](#)
- The workspace must be in a **stopped** state. To do so:
  - a. Navigate to the CodeReady Workspaces Dashboard. See [Navigating CodeReady Workspaces using the Dashboard](#).
  - b. In the **Dashboard**, click the **Workspaces** menu to open the workspaces list and locate the workspace.
  - c. On the same row with the displayed workspace, on the right side of the screen, click the **Stop** button to stop the workspace.
  - d. Wait a few seconds for the workspace to stop, then configure the workspace by clicking on it.

### Procedure

To add the plug-in from the Plug-in registry to an already existing CodeReady Workspaces workspace, use one of the following methods:

- Installing the plug-in from the **Plugin** tab.
  1. Navigate to the **Plugin** tab.  
The list of plug-ins, installed or possible to install, is displayed.
  2. Enable the desired plug-in, for example, the Language Support for Java 11, by using the \***Enable\*** slide-toggle.  
The plug-in source code is added to the workspace devfile, and the plug-in is now enabled.
  3. On the bottom right side of the screen, save the changes by clicking the **Save** button. +  
Once the changes are saved, the workspace is restarted.
- Installing the plug-in by adding content to the devfile.
  1. Navigate to the **Devfile** tab.  
The devfile structure is displayed.



2. Locate the **component** section of the devfile and add the following lines to add the Java language v8 in to the workspace:

```
- id: redhat/java8/latest  
  type: chePlugin
```

See the example of the final result:

```
components:  
- id: redhat/php/latest  
  memoryLimit: 1Gi  
  type: chePlugin  
- id: redhat/php-debugger/latest  
  memoryLimit: 256Mi  
  type: chePlugin  
- mountSources: true  
  endpoints:  
    - name: 8080/tcp  
      port: 8080  
  memoryLimit: 512Mi  
  type: dockerimage  
  volumes:  
    - name: composer  
      containerPath: /home/user/.composer  
    - name: symfony  
      containerPath: /home/user/.symfony  
  alias: php  
  image: 'quay.io/eclipse/che-php-7:nightly'  
- id: redhat/java8/latest  
  type: chePlugin
```

### Additional resources

- [Devfile specifications](#)

## CHAPTER 5. CONFIGURING OAUTH AUTHORIZATION

This section describes how to connect Red Hat CodeReady Workspaces as an OAuth application to supported OAuth providers.

- [Configuring GitHub OAuth](#)
- [Configuring OpenShift OAuth](#)

### 5.1. CONFIGURING GITHUB OAUTH

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

#### Procedure

- Set up the [GitHub OAuth client](#). The **Authorization callback URL** is filled in the next steps.
  1. Go to the RH-SSO administration console and select the **Identity Providers** tab.
  2. Select the **GitHub** identity provider in the drop-down list.
  3. Paste the **Redirect URI** to the **Authorization callback URL** of the GitHub OAuth application.
  4. Fill the **Client ID** and **Client Secret** from the GitHub oauth app.
  5. Enable **Store Tokens**.
  6. Save the changes of the Github Identity provider and click **Register application** in the GitHub oauth app page.

The screenshot shows the Keycloak administration console interface. The left sidebar contains navigation options under 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The main content area is titled 'Identity Providers > Add identity provider' and 'Add identity provider'. A red text instruction says 'Paste this to the redirect url field of the GitHub oauth client'. The form fields are:
 

- Redirect URI: `http://keycloak-che.192.168.99.253.nip.io/auth/realm/sche/broker/github/endpoint`
- Client ID: `acf8882a67e04bfa982d`
- Client Secret: `.....`
- Default Scopes: (empty)
- Store Tokens: **ON** (highlighted with a red circle)
- Stored Tokens Readable: OFF
- Enabled: ON
- Disable User Info: OFF
- Trust Email: OFF
- Account Linking Only: OFF
- Hide on Login Page: OFF
- GUI order: (empty)
- First Login Flow: first broker login
- Post Login Flow: (empty)

 At the bottom are 'Save' and 'Cancel' buttons.

### 5.2. CONFIGURING OPENSIFT OAUTH

For users to interact with OpenShift, they must first authenticate to the OpenShift cluster. OpenShift OAuth is a process in which users prove themselves to a cluster through an API with obtained OAuth access tokens.

Authentication with the [OpenShift connector plugin](#) is a possible way for CodeReady Workspaces users to authenticate with an OpenShift cluster.

The following section describes the OpenShift OAuth configuration options and its use with a CodeReady Workspaces.

## Prerequisites

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

## Procedure

To enable OpenShift OAuth automatically, deployed CodeReady Workspaces using the `crwctl` with the `-os-oauth` option. See the [crwctl server:start specification](#) chapter.

- For CodeReady Workspaces deployed in single-user mode:
  1. Register CodeReady Workspaces OAuth client in OpenShift. See the [Register an OAuth client in OpenShift](#) chapter.

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: che
secret: "<random set of symbols>"
redirectURLs:
  - "<CodeReady Workspaces api url>/oauth/callback"
grantMethod: prompt
')
```

2. Add the OpenShift SSL certificate to the CodeReady Workspaces Java trust store.
  - See [Adding self-signed SSL certificates to CodeReady Workspaces](#) .
3. Update the OpenShift deployment configuration.

```
CHE_OAUTH_OPENSHIFT_CLIENTID: <client-ID>
CHE_OAUTH_OPENSHIFT_CLIENTSECRET: <openshift-secret>
CHE_OAUTH_OPENSHIFT_OAUTH__ENDPOINT: <oauth-endpoint>
CHE_OAUTH_OPENSHIFT_VERIFY__TOKEN__URL: <verify-token-url>
```

- **<client-ID>** a name specified in the OpenShift OAuthClient.
- **<openshift-secret>** a secret specified in the OpenShift OAuthClient.
- **<oauth-endpoint>** the URL of the OpenShift OAuth service:
  - For OpenShift 3 specify the OpenShift master URL.
  - For OpenShift 4 specify the **oauth-openshift** route.
- **<verify-token-url>** request URL that is used to verify the token. **<OpenShift master url>/api** can be used for OpenShift 3 and 4.
- See [CodeReady Workspaces configMaps and their behavior](#) .

## CHAPTER 6. USING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT

This section describes how to manually configure various technology stacks to work with artifacts from in-house repositories using self-signed certificates.

- [Using Maven artifact repositories](#)
- [Using Gradle artifact repositories](#)
- [Using Python artifact repositories](#)
- [Using Go artifact repositories](#)
- [Using NuGet artifact repositories](#)

### 6.1. USING MAVEN ARTIFACT REPOSITORIES

Maven downloads artifacts that are defined in two locations:

- Artifact repositories defined in a **pom.xml** file of the project. Configuring repositories in **pom.xml** is not specific to Red Hat CodeReady Workspaces. For more information, see [the Maven documentation about the POM](#).
- Artifact repositories defined in a **settings.xml** file. By default, **settings.xml** is located at `~/m2/settings.xml`.

#### 6.1.1. Defining repositories in settings.xml

To specify your own artifact repositories at **example.server.org**, use the **settings.xml** file. To do that, ensure, that **settings.xml** is present in all the containers that use Maven tools, in particular the Maven container and the Java plug-in container.

By default, **settings.xml** is located at the `<home dir>/m2` directory which is already on persistent volume in Maven and Java plug-in containers and you don't need to re-create the file each time you restart the workspace if it isn't in ephemeral mode.

In case you have another container that uses Maven tools and you want to share `<home dir>/m2` folder with this container, you have to specify the custom volume for this specific component in the devfile:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: chePlugin
  alias: maven-tool
  id: plugin/id
  volumes:
  - name: m2
    containerPath: <home dir>/m2
```

#### Procedure

1. Configure your **settings.xml** file to use artifact repositories at **example.server.org**:

```
<settings>
  <profiles>
    <profile>
      <id>my-nexus</id>
      <pluginRepositories>
        <pluginRepository>
          <id>my-nexus-snapshots</id>
          <releases>
            <enabled>>false</enabled>
          </releases>
          <snapshots>
            <enabled>>true</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-snapshots/</url>
        </pluginRepository>
        <pluginRepository>
          <id>my-nexus-releases</id>
          <releases>
            <enabled>>true</enabled>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-releases/</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>my-nexus</activeProfile>
  </activeProfiles>
</settings>
```





```

Trust this certificate? [no]: yes
Certificate was added to keystore
Owner: CN=example.com
Issuer: CN=example.com
Serial number: 80ca0f6980c6019a
Valid from: Thu Feb 06 11:00:29 CET 2020 until: Fri Feb 05 11:00:29 CET 2021
Certificate fingerprints:
  MD5: 88:3C:EC:E1:BE:57:DD:9D:46:36:8E:DD:BF:14:04:22
  SHA1: 08:D8:79:D3:F8:6B:5C:3D:71:AA:23:CA:72:01:47:BD:9D:91:0A:AD
  SHA256:
5C:BB:66:81:44:D2:50:EE:EB:CE:D6:15:7E:63:E1:9A:71:EA:58:3F:14:01:15:4E:68:5D:71:
0A:A0:31:33:29
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: *.apps.example.com
]

Trust this certificate? [no]: yes
Certificate was added to keystore

```

- b. Upload the truststore file to **/projects/maven/truststore.jks** to make it available for all containers.

2. Add the truststore file.

- In the Maven container:
  - a. Add the **javax.net.ssl** system property to the **MAVEN\_OPTS** environment variable:

```

- mountSources: true
  alias: maven
  type: dockerimage
  ...
  env:
    -name: MAVEN_OPTS
    value: >-
      -Duser.home=/projects/maven -
      Djavax.net.ssl.trustStore=/projects/truststore.jks

```

- b. Restart the workspace.

- In the Java plug-in container:
 In the devfile, add the **javax.net.ssl** system property for the Java language server:

```

components:
- id: redhat/java11/latest
  type: chePlugin
  preferences:
    java.jdt.ls.vmargs: >-
      -noverify -Xmx1G -XX:+UseG1GC -XX:+UseStringDeduplication

```



```

-Duser.home=/projects/maven
-Djavax.net.ssl.trustStore=/projects/truststore.jks
[...]

```

## 6.2. USING GRADLE ARTIFACT REPOSITORIES

### 6.2.1. Downloading different versions of Gradle

The recommended way to download any version of Gradle is by using the Gradle Wrapper script. If your project does not have a **gradle/wrapper** directory, run **\$ gradle wrapper** to configure the Wrapper.

#### Prerequisites

- The Gradle Wrapper is present in your project.

#### Procedure

To download a Gradle version from a non-standard location, change your Wrapper settings in **/projects/<your\_project>/gradle/wrapper/gradle-wrapper.properties**:

- Change the **distributionUrl** property to point to a URL of the Gradle distribution ZIP file:

```

properties
distributionUrl=http://<url_to_gradle>/gradle-6.1-bin.zip

```

Alternatively, you may place a Gradle distribution zip file locally in **/project/gradle** in your workspace.

- Change the **distributionUrl** property to point to a local address of the Gradle distribution zip file:

```

properties
distributionUrl=file:~/projects/gradle/gradle-6.1-bin.zip

```

### 6.2.2. Configuring global Gradle repositories

Use an initialization script to configure global repositories for the workspace. Gradle performs extra configuration before projects are evaluated, and this configuration is used in each Gradle project from the workspace.

#### Procedure

To set global repositories for Gradle that could be used in each Gradle project in the workspace, create an **init.gradle** script in the **~/.gradle/** directory:

```

allprojects {
  repositories {
    mavenLocal ()
    maven {
      url "http://repo.mycompany.com/maven"
      credentials {
        username "admin"
        password "my_password"
      }
    }
  }
}

```



This file configures Gradle to use a local Maven repository with the given credentials.



#### NOTE

The `~/.gradle` directory does not persist in the current Java plug-in versions, so you must create the `init.gradle` script at each workspace start in the Java plug-in sidecar container.

### 6.2.3. Using self-signed certificates in Java projects

Internal artifact repositories often do not have a certificate signed by an authority that is trusted by default in Java. They are usually signed by an internal company authority or are self-signed. Configure your tools to accept these certificates by adding them to the Java truststore.

#### Procedure

1. Obtain a server certificate file from the repository server. It is often a file named `tls.crt`.
  - a. Create a Java truststore file:

```
$ keytool -import -file tls.crt -alias nexus -keystore truststore.jks -storepass changeit

Trust this certificate? [no]: yes
Certificate was added to keystore
Owner: CN=example.com
Issuer: CN=example.com
Serial number: 80ca0f6980c6019a
Valid from: Thu Feb 06 11:00:29 CET 2020 until: Fri Feb 05 11:00:29 CET 2021
Certificate fingerprints:
  MD5: 88:3C:EC:E1:BE:57:DD:9D:46:36:8E:DD:BF:14:04:22
  SHA1: 08:D8:79:D3:F8:6B:5C:3D:71:AA:23:CA:72:01:47:BD:9D:91:0A:AD
  SHA256:
5C:BB:66:81:44:D2:50:EE:EB:CE:D6:15:7E:63:E1:9A:71:EA:58:3F:14:01:15:4E:68:5D:71:
0A:A0:31:33:29
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: *.apps.example.com
]

Trust this certificate? [no]: yes
Certificate was added to keystore
```

- b. Upload the truststore file to `/projects/gradle/truststore.jks` to make it available for all containers.

2. Add the truststore file in the Gradle container.
  - a. Add the **javax.net.ssl** system property to the **JAVA\_OPTS** environment variable:

```

- mountSources: true
  alias: maven
  type: dockerimage
  ...
  env:
    -name: JAVA_OPTS
    value: >-
      -Duser.home=/projects/gradle -Djavax.net.ssl.trustStore=/projects/truststore.jks

```

### Additional resources

- [Gradle documentation about initialization scripts](#)
- [The Gradle Wrapper documentation](#)

## 6.3. USING PYTHON ARTIFACT REPOSITORIES

### 6.3.1. Configuring Python to use a non-standard registry

To specify a non-standard repository for use by the Python pip tool, set the **PIP\_INDEX\_URL** environment variable.

#### Procedure

- In your devfile, configure the **PIP\_INDEX\_URL** environment variable for the language support and for the development container components:

```

- id: ms-python/python/latest
  memoryLimit: 512Mi
  type: chePlugin
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
- mountSources: true
  memoryLimit: 512Mi
  type: dockerimage
  alias: python
  image: 'quay.io/eclipse/che-python-3.7:nightly'
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'

```

### 6.3.2. Using self-signed certificates in Python projects

Internal artifact repositories often do not have a self-signed (SSL) certificate signed by an authority that is trusted by default. They are usually signed by an internal company authority or are self-signed. Configure your tools to accept these certificates.

Python uses certificates from a file defined in the **PIP\_CERT** environment variable.

## Procedure

1. Obtain the certificate from the non-standard repository and place the certificate file in the `/projects/ssl` file to make it accessible from all your containers.



### NOTE

pip accepts certificates in the Privacy-Enhanced Mail (PEM) format only. Convert the certificate to the PEM format using OpenSSL if necessary.

2. Configure the devfile:

```
- id: ms-python/python/latest
  memoryLimit: 512Mi
  type: chePlugin
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
    - value: '/projects/ssl/rootCA.pem'
      name: 'PIP_CERT'
- mountSources: true
  memoryLimit: 512Mi
  type: dockerimage
  alias: python
  image: 'quay.io/eclipse/che-python-3.7:nightly'
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
    - value: '/projects/ssl/rootCA.pem'
      name: 'PIP_CERT'
```

## 6.4. USING GO ARTIFACT REPOSITORIES

To configure Go in a restricted environment, use the **GOPROXY** environment variable and the [Athens](#) module datastore and proxy.

### 6.4.1. Configuring Go to use a non-standard-registry

Athens is a Go module datastore and proxy with many configuration options. It can be configured to act only as a module datastore and not as a proxy. An administrator can upload their Go modules to the Athens datastore and have them available across their Go projects. If a project tries to access a Go module that is not in the Athens datastore, the Go build fails.

- To work with Athens, configure the **GOPROXY** environment variable in the devfile of your CLI container:

```
components:
- mountSources: true
  type: dockerimage
  alias: go-cli
  image: 'quay.io/eclipse/che-golang-1.12:7.7.0'
  ...
- value: /tmp/.cache
```

```

name: GOCACHE
- value: 'http://your.athens.host'
name: GOPROXY

```

## 6.4.2. Using self-signed certificates in Go projects

Internal artifact repositories often do not have a self-signed (SSL) certificate signed by an authority that is trusted by default. They are usually signed by an internal company authority or are self-signed. Configure your tools to accept these certificates.

Go uses certificates from a file defined in the **SSL\_CERT\_FILE** environment variable.

### Procedure

1. Obtain the certificate used by the Athens server in the Privacy-Enhanced Mail (PEM) format and place it in the **/projects/ssl** file to make it accessible from all your containers.
2. Right-click the project explorer and select **Upload files** to upload the **rootCA.crt** certificate file to your Red Hat CodeReady Workspaces workspace.
3. Add the appropriate environment variables to your devfile:

```

components:
- mountSources: true
  type: dockerimage
  alias: go-cli
  image: 'quay.io/eclipse/che-golang-1.12:7.7.0'
  ...
- value: /tmp/.cache
  name: GOCACHE
- value: 'http://your.athens.host'
  name: GOPROXY
- value: 'on'
  name: GO111MODULE
- value: '/projects/ssl/rootCA.crt'
  name: SSL_CERT_FILE

```

### Additional resources

- [GitHub - gomods/athens: A Go module datastore and proxy](#)

## 6.5. USING NUGET ARTIFACT REPOSITORIES

To configure NuGet in a restricted environment, modify the **nuget.config** file and use the **SSL\_CERT\_FILE** environment variable in the devfile to add self-signed certificates.

### 6.5.1. Configuring NuGet to use a non-standard artifact repository

NuGet searches for configuration files anywhere between the solution directory and the driver root directory. If you put the **nuget.config** file in the **/projects** directory, the **nuget.config** file defines NuGet behavior for all projects in **/projects**.

### Procedure

- Create and place the **nuget.config** file in the **/projects** directory.

### Example nuget.config with a Nexus repository hosted at **nexus.example.org**:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <packageSources>
    <add key="nexus2" value="https://nexus.example.org/repository/nuget-hosted/" />
  </packageSources>
  <packageSourceCredentials>
    <nexus2>
      <add key="Username" value="user" />
      <add key="Password" value="..." />
    </nexus2>
  </packageSourceCredentials>
</configuration>
```

## 6.5.2. Using self-signed certificates in NuGet projects

Internal artifact repositories often do not have a self-signed (SSL) certificate signed by an authority that is trusted by default. They are usually signed by an internal company authority or are self-signed. Configure your tools to accept these certificates.

### Procedure

1. Obtain the certificate file of a non-standard repository and place it in the **/projects/ssl** file to make it accessible from all your containers.
2. Specify the location of the certificate file in the **SSL\_CERT\_FILE** environment variable in your devfile for the OmniSharp plug-in and for the .NET container.

### Example of the devfile:

```
components:
  - id: redhat-developer/che-omnisharp-plugin/latest
    memoryLimit: 1024Mi
    type: chePlugin
    alias: omnisharp
    env:
      - value: /projects/ssl/rootCA.crt
        name: SSL_CERT_FILE
  - mountSources: true
  endpoints:
    - name: 5000/tcp
      port: 5000
  memoryLimit: 512Mi
  type: dockerimage
  volumes:
    - name: dotnet
      containerPath: /home/user
  alias: dotnet
  image: 'quay.io/eclipse/che-dotnet-2.2:7.7.1'
  env:
    - value: /projects/ssl/rootCA.crt
      name: SSL_CERT_FILE
```

■

## 6.6. USING NPM ARTIFACT REPOSITORIES

npm is usually configured using the **npm config** command, writing values to the **.npmrc** files. However, configuration values can also be set using the environment variables beginning with **NPM\_CONFIG\_**.

The Javascript/Typescript plug-in used in Red Hat CodeReady Workspaces does not download any artifacts. It is enough to configure npm in the dev-machine component.

Use the following environment variables for configuration:

- The URL for the artifact repository: **NPM\_CONFIG\_REGISTRY**
- For using a certificate from a file: **NODE\_EXTRA\_CA\_CERTS**

To be able to reference the certificate in a devfile, get a copy of the certificate of the npm repository server and put it inside the **/project** folder.

1. An example configuration for the use of an internal repository with a self-signed certificate:

```
- mountSources: true
endpoints:
  - name: nodejs
    port: 3000
  memoryLimit: '512Mi'
  type: 'dockerimage'
  alias: 'nodejs'
  image: 'quay.io/eclipse/che-nodejs10-ubi:nightly'
  env:
    -name: NODE_EXTRA_CA_CERTS
    value: '/projects/config/tls.crt'
  - name: NPM_CONFIG_REGISTRY
    value: 'https://snexus-airgap.apps.acme.com/repository/npm-proxy'
```

## CHAPTER 7. TROUBLESHOOTING FOR CODEREADY WORKSPACES END USERS

### 7.1. RESTARTING A CODEREADY WORKSPACES WORKSPACE IN DEBUG MODE AFTER START FAILURE

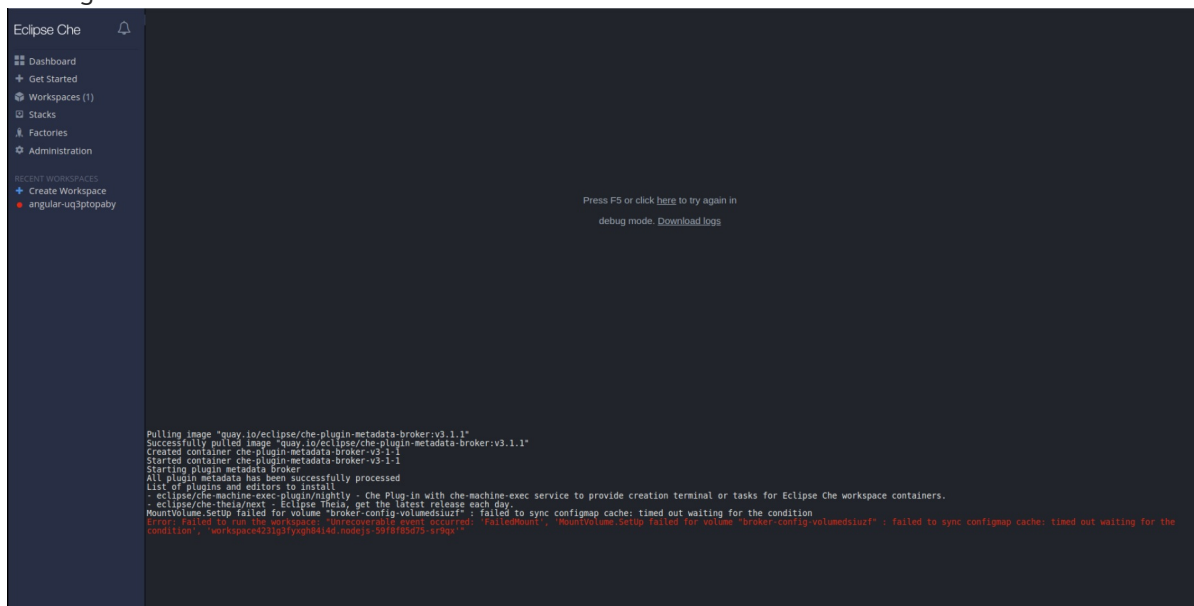
This section describes how to restart the Red Hat CodeReady Workspaces workspace in debug mode after a failure during workspace start.

#### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) CodeReady Workspaces 'quick-starts'.
- An existing workspace that failed to start.

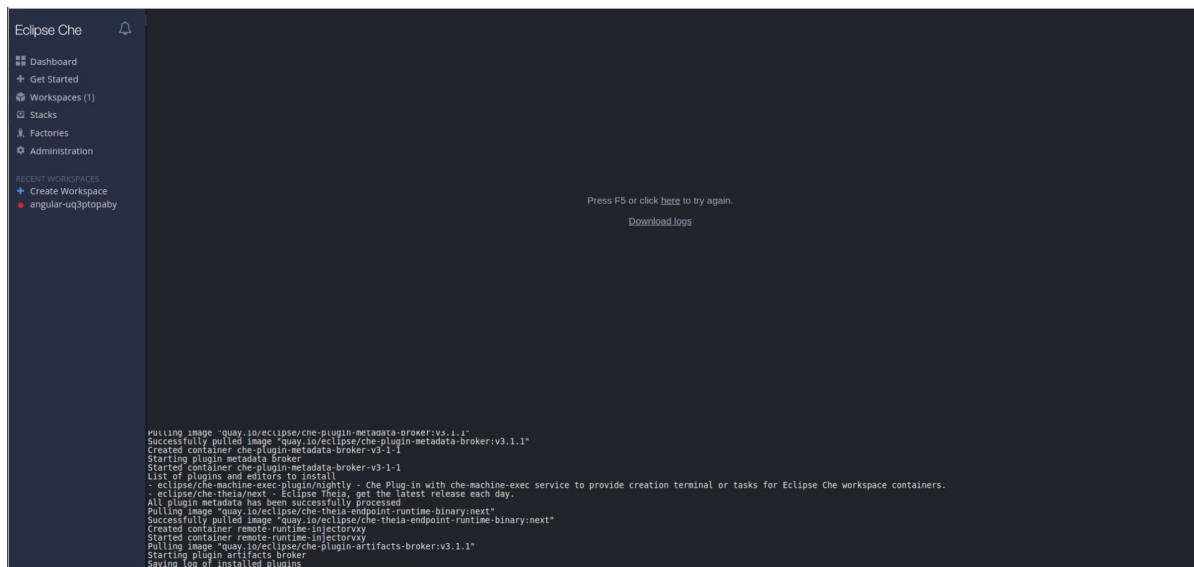
#### Procedure

1. Find the target workspace from the recent workspaces. Click on the target workspace to see the logs.



2. Click the link for restarting in debug mode.
3. Download full logs after start fail with the **Download logs** link.





## 7.2. STARTING A CODEREADY WORKSPACES WORKSPACE IN DEBUG MODE

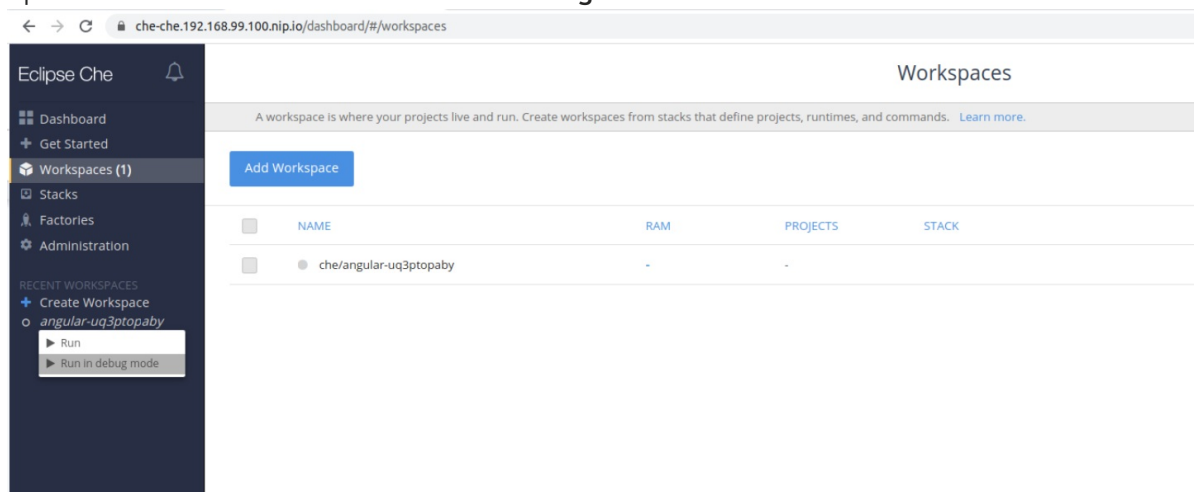
This section describes how to start the Red Hat CodeReady Workspaces workspace in debug mode.

### Prerequisites

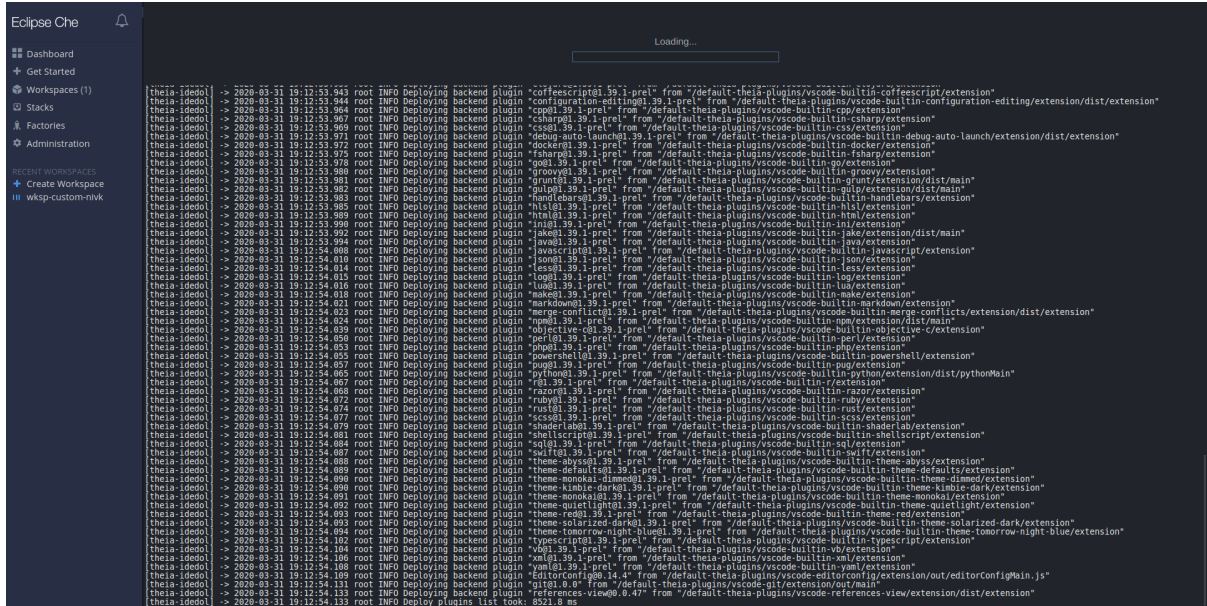
- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#).
- An existing workspace defined on this instance of Red Hat CodeReady Workspaces. See [Creating a new workspace](#).

### Procedure

1. Find the target workspace from the recent workspaces. Right-click the workspace name to open a context menu. Select the **Run in debug mode** item.



2. Click the target workspace to see the logs.
3. The workspace logs are displayed.



## CHAPTER 8. OPENSIFT CONNECTOR OVERVIEW

OpenShift Connector, also referred to as Visual Studio Code OpenShift Connector for Red Hat OpenShift, is a plug-in for CodeReady Workspaces that provides a method for interacting with Red Hat OpenShift 3 or 4 clusters.

OpenShift Connector makes it possible to create, build, and debug applications in the CodeReady Workspaces IDE and then deploy the applications directly to a running OpenShift cluster.

OpenShift Connector is a GUI for the OpenShift Do (**odo**) utility, which aggregates OpenShift CLI (**oc**) commands into compact units. As such, OpenShift Connector helps new developers who do not have OpenShift background with creating applications and running them on the cloud. Instead of using several **oc** commands, the user creates a new component or service by selecting a preconfigured template, such as a Project, an Application, or a Service, and then deploys it as an OpenShift Component to their cluster.

This section provides information about installing, enabling, and basic use of the OpenShift Connector plug-in.

- [Features of OpenShift Connector](#)
- [Installing OpenShift Connector in Red Hat CodeReady Workspaces](#)
- [Authenticating with OpenShift Connector from Red Hat CodeReady Workspaces](#)
- [Creating Components with OpenShift Connector in Red Hat CodeReady Workspaces](#)
- [Connecting source code from GitHub to an OpenShift Component using OpenShift Connector](#)

### 8.1. FEATURES OF OPENSIFT CONNECTOR

The OpenShift Connector plug-in enables the user create, deploy, and push OpenShift Components to an OpenShift Cluster in a GUI.

When used in CodeReady Workspaces, the OpenShift Connector GUI provides the following benefits to its users:

#### Cluster management

- Logging in to clusters using tokens and username and password combinations.
- Switching contexts between different **.kube/config** entries directly from the extension view.
- Viewing and managing OpenShift resources as build and deployment. configurations from the **Explorer** view.

#### Development

- Connecting to a local or hosted OpenShift cluster directly from CodeReady Workspaces.
- Quickly updating the cluster with your changes.
- Creating Components, Services, and Routes on the connected cluster.
- Adding storage directly to a component from the extension itself.

## Deployment

- Deploying to OpenShift clusters with a single click directly from CodeReady Workspaces.
- Navigating to the multiple Routes, created to access the deployed application.
- Deploying multiple interlinked Components and Services directly on the cluster.
- Pushing and watching component changes from the CodeReady Workspaces IDE.
- Streaming logs directly on the integrated terminal view of CodeReady Workspaces.

## Monitoring

- Working with OpenShift resources directly from the CodeReady Workspaces IDE.
- Starting and resuming build and deployment configurations.
- Viewing and following logs for deployments, pods, and containers.

## 8.2. INSTALLING OPENSIFT CONNECTOR IN CODEREADY WORKSPACES

OpenShift Connector is a plug-in designed to create basic OpenShift Components, using CodeReady Workspaces as the editor, and to deploy the Component to an OpenShift cluster. To visually verify that the plug-in is available in your instance, see whether the OpenShift icon is displayed in the CodeReady Workspaces left menu.

To install and enable OpenShift Connector in a CodeReady Workspaces instance, use instructions in this section.

### Prerequisites

- A running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#).

### Procedure

Install OpenShift Connector in CodeReady Workspaces by adding it as an extension in the CodeReady Workspaces **Plugins** panel.

1. Open the **CodeReady Workspaces Plugins** panel by pressing **Ctrl+Shift+J** or by navigating to **View → Plugins**.
2. Search for **vscode-openshift-connector**, and click the **Install** button.
3. Restart the workspace for the changes to take effect.
4. The dedicated OpenShift Application Explorer icon is added to the left panel.

## 8.3. AUTHENTICATING WITH OPENSIFT CONNECTOR FROM CODEREADY WORKSPACES

Before the user can develop and push Components from CodeReady Workspaces, they need to authenticate with an OpenShift cluster.

OpenShift Connector offers the following methods for logging in to the OpenShift Cluster from the CodeReady Workspaces instance:

- Using the notification that asks to log in to the OpenShift cluster where CodeReady Workspaces is deployed to.
- Using the **Log in to the cluster** button.
- Using the Command Palette.



## NOTE

### In CodeReady Workspaces 2.2, Openshift Connector plug-in requires manual connecting to the target cluster

By default, the Openshift Connector plug-in logs into the cluster as **inClusterUser**, which may not have the manage project permission. This causes an error message to be displayed when a new project is being created using Openshift Application Explorer:

```
Failed to create Project with error 'Error: Command failed: "/tmp/vscode-unpacked/redhat.vscode-openshift -connector.latest.qvkozqtkba.openshift-connector-0.1.4-523.vsix/extension/out/tools/linux/odo" project create test-project X
projectrequests.project.openshift.io is forbidden
```

To work around this temporary issue, log out from the local cluster and relog in to OpenShift cluster using the OpenShift user's credentials.

When using a local instance of OpenShift (such as CodeReady Containers or Minishift), the user's credentials are stored in the workspace `~/.kube/config` file, and may be used for automatic authentication in subsequent logins. In the context of CodeReady Workspaces, the `~/.kube/config` is stored as a part of the plug-in sidecar container.

## Prerequisites

- A running instance of CodeReady Workspaces. To install an instance of CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide CodeReady Workspaces quick-starts](#).
- A CodeReady Workspaces workspace has been created.
- The OpenShift Connector plug-in is installed.
- The OpenShift OAuth provider is configured (only for the auto-login to the OpenShift cluster where CodeReady Workspaces is deployed. See [Configuring OpenShift OAuth](#)).

## Procedure

1. In the left panel, select the **OpenShift Application Explorer** icon. The OpenShift Connector panel is displayed.
2. Log in using the OpenShift Application Explorer. Use one of the following methods:
  - Click the **Log in to cluster** button in the top left corner of the pane.

- Press **F1** to open the Command Palette, or navigate to **View → Find Command** in the top menu.  
Search for **OpenShift: Log in to cluster** and press **Enter**.
3. If a **You are already logged in a cluster.** message appears, click **Yes**.  
A selection whether to log in using **Credentials** or **Token** is displayed at the top of the screen.
  4. Select the method to log in to the cluster and follow the login instructions.

**NOTE**

For authenticating with a token, the required token information is in the top right corner of the main OpenShift Container Platform screen, under **<User name> → Copy Login Command**.

## 8.4. CREATING COMPONENTS WITH OPENSIFT CONNECTOR IN CODEREADY WORKSPACES

In the context of OpenShift, Components and Services are basic structures that need to be stored in Application, which is a part of the OpenShift project that organizes deployables into virtual folders for better readability.

This chapter describes how to create OpenShift Components in the CodeReady Workspaces using the OpenShift Connector plug-in and push them to an OpenShift cluster.

### Prerequisites

- A running instance of CodeReady Workspaces. To install an instance of CodeReady Workspaces, see [the CodeReady Workspaces 2.2 Installation Guide](#) [CodeReady Workspaces quick-starts](#).
- The user is logged in to an OpenShift cluster using the OpenShift Connector plug-in.

### Procedure

1. In the OpenShift Connector panel, right-click the row with the red OpenShift icon and select **New Project**.
2. Enter a name for your project.
3. Right-click the created project and select **New Component**.
4. When prompted, enter the name for a new OpenShift Application in which the component can be stored.

The following options of source for your component are displayed:

a. **Git Repository**

This prompts you to specify a Git repository URL and select the intended revision of the runtime.

b. **Binary File**

This prompts you to select a file from the file explorer.

c. **Workspace Directory**

This prompts you to select a folder from the file explorer.

5. Enter the name for the component.
6. Select the component type.
7. Select the component type version.
8. The component is created. Right-click the component, select **New URL**, and enter a name of your choice.
9. The component is ready to be pushed to the OpenShift cluster. To do so, right-click the component and select **Push**.  
The component is now deployed to the cluster. Right-click for selecting additional actions, such as debugging and opening in a browser (requires port **8080** to be exposed).

## 8.5. CONNECTING SOURCE CODE FROM GITHUB TO AN OPENSIFT COMPONENT USING OPENSIFT CONNECTOR

When the user has a Git-stored source code that is wanted for further development, it is more efficient to deploy it directly from the Git repository into the OpenShift Connector Component.

This chapter describes how to obtain the content from the Git repository and connect it with a CodeReady Workspaces-developed OpenShift Component.

### Prerequisites

- Have a running CodeReady Workspaces workspace.
- Be logged in to the OpenShift cluster using the OpenShift Connector.

### Procedure

To make changes to your GitHub component, clone the repository into CodeReady Workspaces to obtain this source code:

1. In the CodeReady Workspaces main screen, open the **Command Palette** by pressing **F1**.
2. Type the **Git Clone** command in the **Command Palette** and press **Enter**.
3. Provide the GitHub URL and select the destination for the deployment.
4. Add source-code files to your Project by clicking the **Add to workspace** button.

For additional information about cloning Git repository, see: [Accessing a Git repository via HTTPS](#).