Red Hat CodeReady Workspaces 2.15

End-user Guide

Using Red Hat CodeReady Workspaces 2.15

Last Updated: 2022-02-22
Legal Notice

Copyright © 2022 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® is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js® 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® 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

MAKING OPEN SOURCE MORE INCLUSIVE ......................................................... 8

CHAPTER 1. NAVIGATING CODEREADY WORKSPACES ............................................ 9

1.1. Navigating CodeReady Workspaces Using the Dashboard ............................... 9
   1.1.1. Logging in to CodeReady Workspaces on OpenShift for the first time using OAuth 9
   1.1.2. Logging in to CodeReady Workspaces on OpenShift for the first time registering as a new user 9
   1.1.3. Logging in to CodeReady Workspaces using crwctl ................................ 10
   1.1.4. Finding CodeReady Workspaces cluster URL using the OpenShift 4 CLI 10

1.2. Importing Certificates to Browsers ................................................................. 11
   1.2.1. Adding certificates to Google Chrome on Linux or Windows 11
   1.2.2. Adding certificates to Google Chrome on macOS .................................... 12
   1.2.3. Adding certificates to Mozilla Firefox .................................................... 12

1.3. Accessing CodeReady Workspaces from OpenShift Developer Perspective ...... 13
   1.3.1. OpenShift Developer Perspective integration with CodeReady Workspaces 13
   1.3.2. Editing the code of applications running in OpenShift Container Platform using CodeReady Workspaces 13
   1.3.3. Accessing CodeReady Workspaces from Red Hat Applications menu 14

CHAPTER 2. CHE-THEIA IDE BASICS ................................................................. 16

2.1. Defining Custom Commands for CHE-THEIA ........................................... 16
   2.1.1. Che-Theia task types ........................................................................... 16
   2.1.2. Running and debugging ..................................................................... 17
   2.1.3. Editing a task and launch configuration .............................................. 21

2.2. Version Control .............................................................................................. 22
   2.2.1. Managing Git configuration: identity .................................................. 22
   2.2.2. Accessing a Git repository using HTTPS ............................................. 23
   2.2.3. Accessing a Git repository using a generated SSH key pair ............... 24
     2.2.3.1. Generating an SSH key using the CodeReady Workspaces command palette 24
     2.2.3.2. Adding the associated public key to a repository or account on GitHub 24
     2.2.3.3. Adding the associated public key to a Git repository or account on GitLab 25
   2.2.4. Managing pull requests using the GitHub PR plug-in 25
     2.2.4.1. Using the GitHub Pull Requests plug-in ......................................... 25

2.3. CHE-THEIA Troubleshooting ....................................................................... 26

2.4. Differences in CHE-THEIA Webview in Single-Host Mode and Multihost Mode 27
   2.4.1. What’s a Webview .............................................................................. 27
   2.4.2. Webview in multihost mode ................................................................. 27
   2.4.3. Webview in single-host mode ............................................................ 27

CHAPTER 3. DEVELOPER WORKSPACES .............................................................. 28

3.1. Creating a Workspace from a Code Sample .............................................. 29
3.2. Creating a Workspace from a Template Devfile ......................................... 30
3.3. Creating a Workspace from Remote Devfile .............................................. 30
   3.3.1. Creating a workspace from the default branch of a Git repository 30
   3.3.2. Creating a workspace from a feature branch of a Git repository ......... 31
   3.3.3. Creating a workspace from a publicly accessible standalone devfile using HTTP 32
   3.3.4. Overriding devfile values using factory parameters 32
   3.3.5. Allowing users to define workspace deployment labels and annotations 35
   3.3.6. Allowing users to define workspace creation strategy 37

3.4. Creating a Workspace Using CRWCTL and a Local Devfile .................... 37
3.5. Importing the Source Code of a Project Into a Workspace ......................... 38
   3.5.1. Importing a project when creating a workspace .............................. 39
   3.5.2. Importing a project into a created workspace .................................... 40
3.5.2.1. Adding commands to an imported project 40
3.5.3. Importing a project with Git: Clone in the GUI 41
3.5.4. Importing a project with the Git clone command in a terminal 42
3.6. CONFIGURING A CODEREADY WORKSPACES 2.15 WORKSPACE 43
3.6.1. Changing the configuration of an existing workspace 43
3.6.2. Adding projects to your workspace 44
3.6.3. Configuring the workspace tools
   3.6.3.1. Adding plug-ins 45
   3.6.3.2. Defining the workspace editor 45
3.7. RUNNING AN EXISTING WORKSPACE FROM THE USER DASHBOARD 46
3.7.1. Running an existing workspace from the user dashboard with the Run button 46
3.7.2. Running an existing workspace from the user dashboard using the Open button 47
3.7.3. Running an existing workspace from the user dashboard using the Recent Workspaces 47
3.8. IMPORTING OPENSHIFT APPLICATIONS INTO A WORKSPACE 47
3.8.1. Including a OpenShift application in a workspace devfile definition 48
3.8.2. Adding a OpenShift application to an existing workspace using the dashboard 50
3.8.3. Generating a devfile from an existing OpenShift application 50
3.9. REMOTELY ACCESSING WORKSPACES 51
3.9.1. Remotely accessing workspaces using oc 52
3.9.2. Downloading and uploading a file to a workspace using the command-line interface 53
3.10. MOUNTING A SECRET AS A FILE OR AN ENVIRONMENT VARIABLE INTO A WORKSPACE CONTAINER 54
   3.10.1. Mounting a secret as a file into a workspace container 54
   3.10.2. Mounting a secret as an environment variable into a workspace container 57
3.11. AUTHENTICATING USERS ON PRIVATE REPOSITORIES OF SCM SERVERS 60
   3.11.1. Authenticating on Bitbucket servers 60
   3.11.2. Authenticating on GitLab servers 61
   3.11.3. Authenticating on GitHub servers 62

CHAPTER 4. AUTHORING DEVFILES 65
4.1. AUTHORING DEVFILES VERSION 1 65
   4.1.1. What is a devfile 65
   4.1.2. A minimal devfile 65
   4.1.3. Generating workspace names 66
   4.1.4. Writing a devfile for a project
      4.1.4.1. Preparing a minimal devfile 67
      4.1.4.2. Specifying multiple projects in a devfile 67
   4.1.5. Devfile reference
      4.1.5.1. Adding schema version to a devfile 68
      4.1.5.2. Adding a name to a devfile 69
      4.1.5.3. Adding projects to a devfile
         4.1.5.3.1. Project-source type: git 70
         4.1.5.3.2. Project-source type: zip 70
         4.1.5.3.3. Project clone-path parameter: clonePath 71
      4.1.5.4. Adding components to a devfile
      4.1.5.4.1. Component type: cheEditor 71
      4.1.5.4.2. Component type: chePlugin 71
      4.1.5.4.3. Specifying an alternative component registry 72
      4.1.5.4.4. Specifying a component by linking to its descriptor 72
      4.1.5.4.5. Tuning chePlugin component configuration 72
      4.1.5.4.6. Component type: kubernetes 73

2
5.5.2. Verifying the Visual Studio Code extension API compatibility level

5.6. USING ALTERNATIVE IDES IN CODEREADY WORKSPACES

5.7. CONFIGURING A WORKSPACE TO USE AN IDE BASED ON THE INTelliJ PLATFORM
   5.7.1. Configuring a workspace to use IntelliJ IDEA Community
   5.7.2. Configuring a workspace to use PyCharm Community
   5.7.3. Configuring a workspace to use a custom image with an IDE based on the InteliJ Platform
   5.7.4. Building images for IDEs based on the InteliJ Platform
      5.7.4.1. Building an image for IntelliJ IDEA Community or PyCharm Community
      5.7.4.2. Building an image for an IDE based on the InteliJ Platform
   5.7.5. Provisioning the JetBrains offline activation code

5.8. THEIA-BASED IDES

5.9. ADDING TOOLS TO CODEREADY WORKSPACES AFTER CREATING A WORKSPACE
   5.9.1. Additional tools in the CodeReady Workspaces workspace
   5.9.2. Adding a language support plug-in to a CodeReady Workspaces workspace

5.10. USING PRIVATE CONTAINER REGISTRIES

CHAPTER 6. USING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT

6.1. USING MAVEN ARTIFACT REPOSITORIES
   6.1.1. Defining repositories in settings.xml
   6.1.2. Defining Maven settings.xml file across workspaces
      6.1.2.1. OpenShift 3.11 and OpenShift <1.13
   6.1.3. Using self-signed certificates in Maven projects

6.2. USING GRADLE ARTIFACT REPOSITORIES
   6.2.1. Downloading different versions of Gradle
   6.2.2. Configuring global Gradle repositories
   6.2.3. Using self-signed certificates in Gradle projects

6.3. USING PYTHON ARTIFACT REPOSITORIES
   6.3.1. Configuring Python to use a non-standard registry
   6.3.2. Using self-signed certificates in Python projects

6.4. USING GO ARTIFACT REPOSITORIES
   6.4.1. Configuring Go to use a non-standard-registry
   6.4.2. Using self-signed certificates in Go projects

6.5. USING NUGET ARTIFACT REPOSITORIES
   6.5.1. Configuring NuGet to use a non-standard artifact repository
   6.5.2. Using self-signed certificates in NuGet projects

6.6. USING NPM ARTIFACT REPOSITORIES

CHAPTER 7. TROUBLESHOOTING CODEREADY WORKSPACES

7.1. VIEWING CODEREADY WORKSPACES WORKSPACES LOGS
   7.1.1. Viewing logs from language servers and debug adapters
      7.1.1.1. Checking important logs
      7.1.1.2. Detecting memory problems
      7.1.1.3. Logging the client-server traffic for debug adapters
      7.1.1.4. Viewing logs for Python
      7.1.1.5. Viewing logs for Go
         7.1.1.5.1. Finding the Go path
         7.1.1.5.2. Viewing the Debug Console log for Go
         7.1.1.5.3. Viewing the Go logs output in the Output panel
      7.1.1.6. Viewing logs for the NodeDebug NodeDebug2 adapter
      7.1.1.7. Viewing logs for Typescript
         7.1.1.7.1. Enabling the label switched protocol (LSP) tracing
         7.1.1.7.2. Viewing the Typescript language server log
         7.1.1.7.3. Viewing the Typescript logs output in the Output panel
7.1.8. Viewing logs for Java
  7.1.8.1. Verifying the state of the Eclipse JDT Language Server
  7.1.8.2. Verifying the Eclipse JDT Language Server features
  7.1.8.3. Viewing the Java language server log
  7.1.8.4. Logging the Java language server protocol (LSP) messages
  7.1.9. Viewing logs for Intelephense
    7.1.9.1. Logging the Intelephense client–server communication
    7.1.9.2. Viewing Intelephense events in the Output panel
  7.1.10. Viewing logs for PHP-Debug
  7.1.11. Viewing logs for XML
    7.1.11.1. Verifying the state of the XML language server
    7.1.11.2. Checking XML language server feature flags
    7.1.11.3. Enabling XML Language Server Protocol (LSP) tracing
    7.1.11.4. Viewing the XML language server log
  7.1.12. Viewing logs for YAML
    7.1.12.1. Verifying the state of the YAML language server
    7.1.12.2. Checking the YAML language server feature flags
    7.1.12.3. Enabling YAML Language Server Protocol (LSP) tracing
  7.1.13. Viewing logs for .NET with OmniSharp-Theia plug-in
    7.1.13.1. OmniSharp-Theia plug-in
    7.1.13.2. Verifying the state of the OmniSharp-Theia plug-in language server
    7.1.13.3. Checking OmniSharp Che-Theia plug-in language server features
    7.1.13.4. Viewing OmniSharp-Theia plug-in logs in the Output panel
  7.1.14. Viewing logs for .NET with NetcoredebugOutput plug-in
    7.1.14.1. NetcoredebugOutput plug-in
    7.1.14.2. Verifying the state of the NetcoredebugOutput plug-in
    7.1.14.3. Viewing NetcoredebugOutput plug-in logs in the Output panel
  7.1.15. Viewing logs for Camel
    7.1.15.1. Verifying the state of the Camel language server
    7.1.15.2. Viewing Camel logs in the Output panel
  7.2. VIEWING CHE-THEIA IDE LOGS
    7.2.1. Viewing Che-Theia editor logs using the OpenShift CLI
  7.3. TROUBLESHOOTING SLOW WORKSPACES
    7.3.1. Improving workspace start time
    7.3.2. Improving workspace runtime performance
  7.4. TROUBLESHOOTING NETWORK PROBLEMS

CHAPTER 8. OPENSFiTH CONNECTOR OVERViEW
  8.1. FEATURES OF OPENSFiTH CONNECTOR
  8.2. INSTALLiNG OPENSFiTH CONNECTOR IN CODEREADY WORKSPACES
  8.3. AUTHENTICATING WITH OPENSFiTH CONNECTOR FROM CODEREADY WORKSPACES WHEN THE OPENSFiTH OAUTH SERVICE DOES NOT AUTHENTICATE THE CODEREADY WORKSPACES INSTANCE
  8.4. CREATiNG COMPONENTS WITH OPENSFiTH CONNECTOR IN CODEREADY WORKSPACES
  8.5. CONNECTiNG SOURCE CODE FROM GITHUB TO AN OPENSFiTH COMPONENT USING OPENSFiTH CONNECTOR

CHAPTER 9. TELEMETRY OVERViEW
  9.1. USE CASES
  9.2. HOW IT WORKS
  9.3. CREATING A TELEMETRY PLUG-IN
9.3.1. Getting Started
  Optional: creating a server that receives events
9.3.2. Creating a new Maven project
9.3.3. Running the application
9.3.4. Creating a concrete implementation of AnalyticsManager and adding specialized logic
9.3.5. Implementing isEnabled()
9.3.6. Implementing onEvent()
9.3.7. Implementing increaseDuration()
9.3.8. Implementing onActivity()
9.3.9. Implementing destroy()
9.3.10. Packaging the Quarkus application
9.3.11. Creating a meta.yaml for your plug-in
9.3.12. Updating CodeReady Workspaces to reference your telemetry plug-in

9.4. THE WOOPRA TELEMETRY PLUG-IN
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
CHAPTER 1. NAVIGATING CODEREADY WORKSPACES

This chapter describes available methods to navigate Red Hat CodeReady Workspaces.

- Section 1.1, “Navigating CodeReady Workspaces using the Dashboard”
- Section 1.2, “Importing certificates to browsers”
- Section 1.3, “Accessing CodeReady Workspaces from OpenShift Developer Perspective”

1.1. NAVIGATING CODEREADY WORKSPACES USING THE DASHBOARD

The Dashboard is accessible on your cluster from a URL such as `https://codeready-<openshift_deployment_name>.<domain_name>/dashboard`. This section describes how to access this URL on OpenShift.

1.1.1. 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.

1.1.2. 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.

Self-registration is enabled. See Enabling self-registration.

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.

1.1.3. Logging in to CodeReady Workspaces using crwctl

This section describes how to log in to CodeReady Workspaces using the crwctl tool by copying login command from CodeReady Workspaces Dashboard.

Prerequisites


- Red Hat CodeReady Workspaces Dashboard is opened in a browser.

Procedure

1. Using the upper-right corner of Dashboard, open the user’s pop-up menu.

2. Select the Copy crwctl login command option.

3. Wait for the notification message The login command copied to clipboard to display.

4. Paste the login command into a terminal and observe a successful login:

   $ crwctl auth:login ...
   Successfully logged into <server> as <user>

1.1.4. Finding CodeReady Workspaces cluster URL using the OpenShift 4 CLI

This section describes how to obtain the CodeReady Workspaces cluster URL using the OpenShift 4 command line interface (CLI). 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:

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

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

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

1.2. IMPORTING CERTIFICATES TO BROWSERS

This section describes how to import a root certificate authority into a web browser to use CodeReady Workspaces with self-signed TLS certificates.

When a TLS certificate is not trusted, the error message "Your CodeReady Workspaces server may be using a self-signed certificate. To resolve the issue, import the server CA certificate in the browser." blocks the login process. To prevent this, add the public part of the self-signed CA certificate into the browser after installing CodeReady Workspaces.

1.2.1. Adding certificates to Google Chrome on Linux or Windows

Procedure

1. Navigate to URL where CodeReady Workspaces is deployed.

2. Save the certificate:
   a. Click the warning or open lock icon on the left of the address bar.
   b. Click **Certificates** and navigate to the **Details** tab.
   c. Select the top-level certificate, which is the needed Root certificate authority (do not export the unfolded certificate from the lower level), and export it:
      - On Linux, click the **Export** button.
      - On Windows, click the **Save to file** button.

3. Go to **Google Chrome Certificates settings** in the **Privacy and security** section and navigate to the **Authorities** tab.

4. Click the **Import** button and open the saved certificate file.

5. Select **Trust this certificate for identifying websites** and click the **OK** button.
6. After adding the CodeReady Workspaces certificate to the browser, the address bar displays the closed lock icon next to the URL, indicating a secure connection.

1.2.2. Adding certificates to Google Chrome on macOS

Procedure

1. Navigate to URL where CodeReady Workspaces is deployed.

2. Save the certificate:
   a. Click the lock icon on the left of the address bar.
   b. Click Certificates.
   c. Select the certificate to use and drag its displayed large icon to the desktop.

3. Open the Keychain Access application.

4. Select the System keychain and drag the saved certificate file to it.

5. Double-click the imported CA, then go to Trust and select When using this certificate: Always Trust.

6. Restart the browser for the added certificate to take effect.

1.2.3. Adding certificates to Mozilla Firefox

Procedure

1. Navigate to URL where CodeReady Workspaces is deployed.

2. Save the certificate:
   a. Click the lock icon on the left of the address bar.
   b. Click the > button next to the Connection not secure warning.
   c. Click the More information button.
   d. Click the View Certificate button on the Security tab.
   e. Select the second certificate tab. The certificate Common Name should start with ingress-operator
   f. Click the PEM (cert) link and save the certificate.

3. Navigate to about:preferences, search for certificates, and click View Certificates.

4. Go to the Authorities tab, click the Import button, and open the saved certificate file.

5. Check Trust this CA to identify websites and click OK.

6. Restart Mozilla Firefox for the added certificate to take effect.
7. After adding the CodeReady Workspaces certificate to the browser, the address bar displays the closed lock icon next to the URL, indicating a secure connection.

### 1.3. ACCESSING CODEREADY WORKSPACES FROM OPENSHIFT DEVELOPER PERSPECTIVE

The OpenShift Container Platform web console provides two perspectives; the **Administrator** perspective and the **Developer** perspective.

The Developer perspective provides workflows specific to developer use cases, such as the ability to:

- Create and deploy applications on OpenShift Container Platform by importing existing codebases, images, and Dockerfiles.
- Visually interact with applications, components, and services associated with them within a project and monitor their deployment and build status.
- Group components within an application and connect the components within and across applications.
- Integrate serverless capabilities (Technology Preview).
- Create workspaces to edit your application code using CodeReady Workspaces.

#### 1.3.1. OpenShift Developer Perspective integration with CodeReady Workspaces

This section provides information about OpenShift Developer Perspective support for CodeReady Workspaces.

When the CodeReady Workspaces Operator is deployed into OpenShift Container Platform 4.2 and later, it creates a **ConsoleLink** Custom Resource (CR). This adds an interactive link to the Red Hat Applications menu for accessing the CodeReady Workspaces installation using the OpenShift Developer Perspective console.

To access the Red Hat Applications menu, click the three-by-three matrix icon on the main screen of the OpenShift web console. The CodeReady Workspaces **Console Link**, displayed in the drop-down menu, creates a new workspace or redirects the user to an existing one.

**NOTE**

OpenShift Container Platform console links are not created when CodeReady Workspaces is used with HTTP resources

When installing CodeReady Workspaces with the **From Git** option, the OpenShift Developer Perspective console link is only created if CodeReady Workspaces is deployed with HTTPS. The console link will not be created if an HTTP resource is used.

#### 1.3.2. Editing the code of applications running in OpenShift Container Platform using CodeReady Workspaces

This section describes how to start editing the source code of applications running on OpenShift using CodeReady Workspaces.

**Prerequisites**
CodeReady Workspaces is deployed on the same OpenShift 4 cluster.

Procedure

1. Open the **Topology** view to list all projects.
2. In the **Select an Application** search field, type **workspace** to list all workspaces.
3. Click the workspace to edit.
   The deployments are displayed as graphical circles surrounded by circular buttons. One of these buttons is **Edit Source Code**.

4. To edit the code of an application using CodeReady Workspaces, click the **Edit Source Code** button. This redirects to a workspace with the cloned source code of the application component.

1.3.3. Accessing CodeReady Workspaces from Red Hat Applications menu

This section describes how to access CodeReady Workspaces workspaces from the **Red Hat Applications** menu on OpenShift Container Platform.

Prerequisites

- The CodeReady Workspaces Operator is available in OpenShift 4.

Procedure

1. Open the **Red Hat Applications** menu by using the three-by-three matrix icon in the upper right corner of the main screen.
   The drop-down menu displays the available applications.
2. Click the **CodeReady Workspaces** link to open the CodeReady Workspaces Dashboard.
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.

- Section 2.1, “Defining custom commands for Che-Theia”
- Section 2.2, “Version Control”
- Section 2.3, “Che-Theia Troubleshooting”
- Section 2.4, “Differences in Che-Theia Webview in single-host mode and multihost mode”

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.

This is useful, for example, for:

- Simplifying building, running, and debugging projects.
- Allowing lead developers to customize workspaces based on team requirements.
- Reducing time needed to onboard new team members.

See also Section 4.2, “Authoring a devfile 2”.

2.1.1. Che-Theia task types

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

```json
commands:

- name: Package Native App
  actions:
    - type: exec
      component: centos-quarkus-maven
      command: "mvn package -Dnative -Dmaven.test.skip"
      workdir: ${CHE_PROJECTS_ROOT}/quarkus-quickstarts/getting-started

- name: Start Native App
  actions:
    - type: exec
      component: ubi-minimal
      command: ./getting-started-1.0-SNAPSHOT-runner
      workdir: ${CHE_PROJECTS_ROOT}/quarkus-quickstarts/getting-started/target

- name: Attach remote debugger
  actions:
    - type: vscode-launch
      referenceContent: |
      {
        "version": "0.2.0",
        "configurations": [
```
CodeReady Workspaces commands

**Package Native App** and **Start Native App**
The CodeReady Workspaces commands are to be used to define tasks that will be executed in the workspace container.

- 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.

- The **component** field refers to the container where the command will be executed. The field contains the component **alias** where the container is defined.

Visual Studio Code launch configurations

**Attach remote debugger**
Visual Studio Code **launch** configurations are commonly used to define debugging configuration. To trigger these configurations, press F5 or choose **Start Debugging** from the **Debug** menu. The configurations provide information to the debugger, such as the port to connect to for debugging or the type of the application to debug (Node.js, Java, and others.).

- The type is **vscode-launch**.

- It contains the **launch** configurations in the Visual Studio Code format.

- For more information about Visual Studio Code **launch** configurations, see the Debugging section on the Visual Studio documentation page.

Tasks of type che, also known as exec commands, can be executed from the Terminal→Run Task menu or by selecting them in the My Workspace panel. Other tasks are only available from Terminal→Run Task. Configurations to start with are available in the Che-Theia debugger.

Additional resources

- Quarkus command mode devfile, including a theia task and a vscode-launch task

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


Procedure

To debug an application:

1. Click **Debug → Add Configuration** for debugging or adding of a **launch** configuration to the project.

2. From the pop-up menu, select the appropriate configuration for the application that you are about to debug.
3. Update the configuration by modifying or adding attributes.

```
"configurations": [
  {
    "type": "java",
    "name": "Debug (Launch)",
    "request": "launch",
    "cwd": "$\{workspaceFolder\}",
    "console": "internalConsole",
    "stopOnEntry": false,
    "mainClass": "HelloWorld",
    "args": ""
  }
]
```

4. Breakpoints can be toggled by selecting the editor margin.
5. After opening the breakpoint menu, use the **Edit Breakpoint** command to add conditions.

The IDE then displays the **Expression** input field.

6. To start debugging, click **View → Debug**.
7. 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 Tasks 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


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.

Procedure

- To configure Git identity using the CodeReady Workspaces user interface:

  1. Open File > Settings > Open Preferences or press Ctrl+,.

  2. In the opened window, navigate to the Git → User sub-section and enter the User mail and User name values.

- 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 in the current container and restores it for other containers on future workspace starts.

### 2.2.2. Accessing a Git repository using HTTPS

**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>
```

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

You can generate an SSH key by using the CodeReady Workspaces command palette. You have to create a unique SSH key for each Git provider in use because each SSH key restricts permissions for one specific Git provider.

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

Prerequisites

- An existing workspace defined on this instance of CodeReady Workspaces (Section 3.6, “Configuring a CodeReady Workspaces 2.15 workspace”).
- Personal GitHub account or other Git provider account created.

Procedure

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 in the lower-right corner and copy the public key from the editor and add it to the Git provider.
   It is possible to 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.

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.

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 Visual Studio 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**

- Sign in to GitHub, using the **Accounts** menu or the **Sign in** button in the plug-in’s view:
To sign out from GitHub use the Accounts menu in the left bottom side, or GitHub Pull Requests: Sign out of GitHub command.

Additional resources

- vscode GitHub Pull Requests plugin documentation

2.3. CHE-THEIA TROUBLESHOOTING

This section describes 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.
2.4. DIFFERENCES IN CHE-THEIA WEBVIEW IN SINGLE-HOST MODE AND MULTIHOST MODE

Depending on which Che deployment strategy is used, single-host or multihost, there are differences in how Che-Theia Webview API works.

2.4.1. What’s a Webview

Webview Plug-in API allows creating a view within Che-Theia to show an arbitrary HTML content. Internally, it’s implemented with an iframe and service worker.

2.4.2. Webview in multihost mode

When Red Hat CodeReady Workspaces is deployed in multihost mode, Webview content is served on a separate origin. That means it’s isolated from the main Che-Theia context. So, a contributed view has no access:

- to the top-level DOM
- to the Che-Theia state, like local storage, cookies, and so on.

2.4.3. Webview in single-host mode

When Red Hat CodeReady Workspaces is deployed in single-host mode, Webview content is loaded through the same origin as the main Che-Theia context. It means that nothing prevents external content from accessing the main Che-Theia in a browser. So, pay extra attention to what content may be loaded by different Plugins that contribute the Webviews.
CHAPTER 3. DEVELOPER WORKSPACES

Red Hat CodeReady Workspaces provides developer workspaces with everything you need to code, build, test, run, and debug applications:

- Project source code
- Web-based integrated development environment (IDE)
- Tool dependencies needed by developers to work on a project
- 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 easy to share.

**IMPORTANT**

By default, it is possible to run only one workspace at a time. To increase the number of concurrent workspaces a user can run, update the CheCluster:

```
$ oc patch checluster/codeready-workspaces -n openshift-workspaces --type=merge \
-p '{ "spec": { "server": { "customCheProperties": { "CHE_LIMITS_USER_WORKSPACES_RUN_COUNT": "-1" } } } }'
```

For additional information, see: https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#users-workspace-limits.

**Table 3.1. Features and benefits**

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

Start a CodeReady Workspaces workspace:

- Section 3.6, “Configuring a CodeReady Workspaces 2.15 workspace”

Use the Dashboard to discover CodeReady Workspaces 2.15:

- Section 3.1, “Creating a workspace from a code sample”
- Section 3.5, “Importing the source code of a project into a workspace”

Use a devfile as the preferred way to start a CodeReady Workspaces 2.15 workspace:

- Chapter 4, Authoring devfiles
- Section 3.8, “Importing OpenShift applications into a workspace”

Use the browser-based IDE as the preferred way to interact with a CodeReady Workspaces 2.15 workspace. For an alternative way to interact with a CodeReady Workspaces 2.15 workspace, see: Section 3.9, “Remotely accessing workspaces”.

### 3.1. CREATING A WORKSPACE FROM A CODE SAMPLE

You can create a new workspace by using one of the code samples included in CodeReady Workspaces.

**Prerequisites**


**Procedure**

2. In the left navigation panel, click Create Workspace to go to the Quick Add tab that contains code samples.
3. Select a sample to start a new workspace.
4. Wait for the initialized workspace to start and for the IDE to open.
CodeReady Workspaces generates a unique name for a new workspace from the devfile’s `metadata.generateName` property followed by four random characters. You can edit the name of the workspace when it is created.

### 3.2. CREATING A WORKSPACE FROM A TEMPLATE DEVFILE

You can create a new workspace by using one of the devfile templates included in CodeReady Workspaces.

For information about devfiles, see Section 4.2, “Authoring a devfile 2”.

#### Prerequisites


#### Procedure

1. Go to the **Dashboard** of CodeReady Workspaces.
2. In the left navigation panel, click **Create Workspace**.
3. Go to the **Custom Workspace** tab.
4. Enter a workspace name.
5. Select a devfile template from the drop-down list.
6. Click **Create & Open**.
7. Wait for the initialized workspace to start and for the IDE to open.

### 3.3. CREATING A WORKSPACE FROM REMOTE DEVFILE

For quick and easy CodeReady Workspaces workspace creation, use a factory link.

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

This section describes how to start a CodeReady Workspaces workspace using a factory URL. The factory URL is a link pointing CodeReady Workspaces to a Git source repository containing a devfile.

The factory URL exist in two forms:

- the short form `/#$URL`
- long \`f?url=$URL\` form that supports additional configuration parameters used in previous versions of CodeReady Workspaces

### Prerequisites


- The Git repository `<GIT_REPOSITORY_URL>` is available over HTTPS and contains a `devfile.yaml` or `.devfile.yaml` in the root folder. See Section 4.2, “Authoring a devfile 2”.

### Procedure

- Run the workspace by opening the factory URL. Two formats are available:

  \`
  \https://codeready-<openshift_deployment_name>.<domain_name>/#<GIT_REPOSITORY_URL>
  \`

  This is the short format.

  \`
  \https://codeready-<openshift_deployment_name>.<domain_name>/f?url=<GIT_REPOSITORY_URL>
  \`

  This long format supports additional configuration parameters.

  **Example 3.1.** Create a workspace on Eclipse Che hosted by Red Hat from the default branch of the https://github.com/eclipse-che/che-server repository using the short factory URL format.


  **Example 3.2.** Create a workspace on Eclipse Che hosted by Red Hat from the default branch of the https://github.com/eclipse-che/che-server repository using the long factory URL format.


### 3.3.2. 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


- The `devfile.yaml` or `.devfile.yaml` file is located in the root folder of a Git repository, on a specific branch of the user’s choice that is accessible over HTTPS. See Section 4.2, “Authoring a devfile 2” for detailed information about creating and using devfiles.
Procedure

Execute the workspace by opening the following URL:

`https://codeready-<openshift_deployment_name>.<domain_name>/#<GitHubBranch>`

Example

Use following URL format to open an experimental `quarkus-quickstarts` branch hosted on `workspaces.openshift.com`.


3.3.3. 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


- The publicly-accessible standalone `devfile.yaml` file. See Section 4.2, “Authoring a devfile 2” for detailed information about creating and using devfiles.

Procedure

1. Execute the workspace by opening the following URL:

`https://codeready-<openshift_deployment_name>.<domain_name>/#https://<yourhosturl>/devfile.yaml`

3.3.4. Overriding devfile values using factory parameters

You can override values in the following sections of a remote devfile:

- **apiVersion**
- **metadata**
- **projects**
- **attributes**

You can override the values by using additional factory parameters.

Prerequisites


Procedure

1. Open the workspace by navigating to the following URL:
   \texttt{https://codeready-\langle openshift\_deployment\_name\rangle.\langle domain\_name\rangle/f?url=https://\langle hostURL\rangle/devfile.yaml&override.\langle parameter\_path\rangle=\langle value\rangle} 

Example 3.3. Example of overriding the `generateName` property

Consider the following initial devfile:

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

To add or override `generateName` value, use the following factory URL:

\texttt{https://workspaces.openshift.com/f?url=\langle repository-url\rangle&override.metadata.generateName=myprefix}

The resulting workspace has the following devfile model:

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

Example 3.4. Example of overriding project source branch property

Consider the following initial devfile:

```yaml
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 the source `branch` value, use the following factory URL:

\texttt{https://workspaces.openshift.com/f?url=\langle repository-url\rangle&override.projects.web-java-spring-petclinic.source.branch=1.0.x}

The resulting workspace has the following devfile model:

```yaml
apiVersion: 1.0.0
metadata:
```
Example 3.5. Example of overriding or creating an attribute value

Consider the following initial devfile:

```yaml
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
  persistVolumes: false
projects:
```

To add or override the `persistVolumes` attribute value, use the following factory URL:

```
```

The resulting workspace has the following devfile model:

```yaml
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
  persistVolumes: true
projects:
```

When overriding attributes, everything that follows the `attributes` keyword is interpreted as an attribute name. You can use dot-separated names:

```
https://workspaces.openshift.com/f?url=<repository-url>&override.attributes.dot.name.format.attribute=true
```

The resulting workspace has the following devfile model:

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

1. In CodeReady Workspaces Dashboard, navigate to the Devfile tab of the newly created workspace and inspect the content.

3.3.5. Allowing users to define workspace deployment labels and annotations

This section describes how to customize workspace deployment labels and annotation using factory parameters.

Prerequisites


- A publicly accessible standalone devfile.yaml file. See Section 4.2, “Authoring a devfile 2” for detailed information about creating and using devfiles.

Procedure

1. Open the workspace by navigating to the following URL: \(https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<hostURL>/devfile.yaml&workspaceDeploymentLabels=<url_encoded_comma_separated_key_values>&workspaceDeploymentAnnotations=<url_encoded_comma_separated_key_values_override>\)

Example 3.6. Example of overriding the deployment labels

Consider the following labels to add:

ike.target=preference-v1
ike.session=test

To add or override labels, use the following factory URL:

https://workspaces.openshift.com/f?url=<repository-url>&workspaceDeploymentLabels=ike.target%3Dpreference-v1%2Cike.session%3Dtest

The resulting workspace has the following deployment labels:

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2020-10-27T14:03:26Z"
  generation: 1
  labels:
    che.component.name: che-docs-dev
    che.original_name: che-docs-dev
    che.workspace_id: workspacegln2g1shejufpkd
    ike.session: test
ike.target: preference-v1
name: workspacegln2g1shejjufpkd.che-docs-dev
namespace: opentlc-mgr-che
resourceVersion: "107516"
spec:
...

Example 3.7. Example of overriding the deployment annotations

Consider the following annotations to add:

ike.A1=preference-v1
ike.A=test

To add or override annotations, use the following factory URL:


The resulting workspace has the following deployment annotations:

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    ike.A: test
    ike.A1: preference-v1
  creationTimestamp: "2020-10-28T09:58:52Z"
  generation: 1
  labels:
    che.component.name: che-docs-dev
    che.original_name: che-docs-dev
    che.workspace_id: workspacextf710v64rl5ouz
  name: workspacextf710v64rl5ouz.che-docs-dev
  namespace: opentlc-mgr-che
  resourceVersion: "213191"
...

Verification steps

To display the deployment labels and annotations:

1. Get the name of the user’s namespace:

   a. Using CodeReady Workspaces Dashboard, move to the Workspaces tab and read the name of the OpenShift namespace field.

1. Log in to the cluster:

   a. Retrieve the CodeReady Workspaces cluster URL from the checluster CR (Custom Resource), run:
| $ oc get checluster --output jsonpath='{.items[0].status.cheURL}' |

b. Log in:

| $ oc login -u <username> -p <password> <cluster_URL> |

1. Display the deployment labels and annotations for all deployments in the project using the OpenShift namespace name from the first step:

| $ oc get deployment -n <NAMESPACE> -o=custom-columns="NAMESPACE:.metadata.namespace,NAME:.metadata.name,LABELS:.metadata.labels,ANNOTATIONS:.metadata.annotations" |

3.3.6. Allowing users to define workspace creation strategy

As a developer, you can configure CodeReady Workspaces to create a new workspace each time it accepts a factory URL, or to reuse the existing workspace if a user already has one.

CodeReady Workspaces supports the following options:

- **perclick**: The default strategy, which creates a new workspace each time a given factory URL is accepted.

- **peruser**: Initially, a workspace is created using a factory URL. Other user’s calls then re-use the particular workspace created by the factory URL (1 factory = 1 workspace).

**Prerequisites**


- The Git repository `<GIT_REPOSITORY_URL>` is available over HTTPS.

**Procedure**

- Run the workspace by opening the factory URL and specify the additional strategy parameter: `https://codeready-<openshift_deployment_name>.<domain_name>/f?url=<GIT_REPOSITORY_URL>&policies.create=<value>`

**Additional resources**

- Section 4.2, “Authoring a devfile 2”

3.4. 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


- The devfile is available on the local filesystem in the current working directory. See Section 4.2, “Authoring a devfile 2” for detailed information about creating and using devfiles.

- You are logged in to Red Hat CodeReady Workspaces. See How to login into CodeReady Workspaces using crwctl

Procedure

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

   ```
   $ crwctl workspace:create --name=<WORKSPACE_NAME> \
   --devfile=devfile.yaml --start \n   -n openshift-workspaces
   ```

   The workspace name to create.

   **NOTE**

   If **--devfile** flag is omitted, the crwctl looks for **devfile.yaml** or **devfile.yml** files in the current directory to create a workspace from.

3.5. IMPORTING THE SOURCE CODE OF A PROJECT INTO A WORKSPACE

With CodeReady Workspaces, you can import and work on a project’s current codebase in a workspace.

You can import a project’s codebase into a workspace by using any of the following options:

- Section 3.5.1, “Importing a project when creating a workspace”
- Section 3.5.2, "Importing a project into a created workspace"
- Section 3.5.3, “Importing a project with Git: Clone in the GUI”
- Section 3.5.4, "Importing a project with the Git clone command in a terminal"

Prerequisites


- A configured workspace with plug-ins related to your development environment defined on this instance of Red Hat CodeReady Workspaces. See Section 3.6, “Configuring a CodeReady Workspaces 2.15 workspace”.
3.5.1. Importing a project when creating a workspace

You can import a project’s codebase while creating a new workspace: by using a devfile template and the project’s URL.

**Procedure**

2. In the left navigation panel, click Create Workspace.
3. Go to the Custom Workspace tab.
4. Enter a workspace name.

**NOTE**

If you do not enter a Workspace Name for your workspace, CodeReady Workspaces generates a unique name for a new workspace from the devfile’s `metadata.generateName` property followed by four random characters.

5. Select a devfile template from the drop-down list.

6. Edit the **projects** section of the template devfile displayed in the devfile editor on the page.

---

**Example 3.8. Editing the projects section of a devfile**

```
projects:
```
3.5.2. Importing a project into a created workspace

You can import a project’s codebase into a created workspace by editing its devfile.

**Prerequisites**

- A running workspace

**Procedure**

2. Under **RECENT WORKSPACES**, hover over the name of the workspace to which you will import the project.
3. Click the three dots.
4. Select **Edit Workspace**.
5. In the **Devfile** tab, edit the **projects** section of the devfile in the editor on the page.

**Example 3.9. Editing the projects section of a devfile**

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

For more information about devfiles, see Section 4.2, ”Authoring a devfile 2“.

6. Click **Save**.

3.5.2.1. Adding commands to an imported project

You can add commands to a project that will enable running, debugging, or starting an application in a browser.

**Prerequisites**

- A workspace with a project in it
Procedure


2. Under RECENT WORKSPACES, hover over the name of the workspace to which you will import the project.

3. Click the three dots next to the workspace name.

4. Select Edit Workspace.

5. In the Devfile tab, edit the commands section of the devfile in the editor on the page.

6. Click Save.

7. Open the workspace.

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

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

3.5.3. Importing a project with Git: Clone in the GUI

You can import a project’s codebase into a running workspace with the Git: Clone command in the graphic user interface of CodeReady Workspaces.

Prerequisites
A started workspace

Procedure

1. Select the **Git: Clone** command on the **Welcome** screen or from the command palette:
   - On the **Welcome** screen, click the **Git: Clone** link.
   - To open the command palette, press F1 or **Ctrl+Shift+P** (**Cmd+Shift+P** on macOS).

2. Enter the path to the project that you want to clone.

### 3.5.4. Importing a project with the Git clone command in a terminal

You can use the command line to import a project’s codebase into a running workspace.

**Prerequisites**

- A running workspace

**Procedure**

1. Open a terminal inside the running workspace.
2. Type the `git clone` command to pull code.

```
$ 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 1189
Receiving objects: 100% (1197/1197), 203.71 KiB | 8 bytes/s, done.
Resolving deltas: 100% (665/665), done.
$ ls -la
total 16
drwxwxr-x.-  4 root  root  35 Jun 21 20:18
drwxr-xr-x  1 root  root  20 Jun 21 20:14
drwxr-xr-x  2 user  root  594 Jun 21 19:55 che
drwxr-xr-x  3 user  root  110 Jun 21 20:18 che-parent
```

**NOTE**

Importing or deleting workspace projects in the terminal does not update the workspace configuration, and the IDE does not reflect the changes in the Devfile tab in the dashboard.

Similarly, when you add a project in the Dashboard and then delete the project with the `rm -fr myproject` command, the project may still appear in the Devfile tab.

### 3.6. CONFIGURING A CODEREADY WORKSPACES 2.15 WORKSPACE

#### 3.6.1. 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 CodeReady Workspaces. See [Installing CodeReady Workspaces](#).
- An existing workspace defined on this instance of CodeReady Workspaces.

**Procedure**

1. Navigate to the CodeReady Workspaces Dashboard. See [Section 1.1, “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**.
   - Select **Storage Type**.
3.6.2. Adding projects to your workspace

Prerequisites

- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure

To add a project to your workspace:

1. Navigate to the Workspaces page and click the workspace, which is about to be updated.
2. Open the Devfile tab.
3. In the Devfile editor, add a projects section with desired project.
4. Once the project is added, click the Save button to save this workspace configuration.

For demonstration example, see below:

**Example - Adding a .git project into a workspace using a devfile**

In the following instance, the project crw acts as the example of a user’s project. A user specifies this project using the name attribute of a devfile. The location attribute defines the source repository represented by an URL to a Git repository or ZIP archive.

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

```
projects:
- name: <crw>
  source:
    type: git
    location: 'https://github.com/<github-organization>/crw.git'
```
3.6.3. Configuring the workspace tools

3.6.3.1. Adding plug-ins

**Prerequisites**

- An existing workspace defined on this instance of CodeReady Workspaces.

**Procedure**

To add plug-ins to your workspace:

1. Click the *Devfile* tab.

2. Add the desired *chePlugin* component and click the *Save* button.

**NOTE**

To see a list of available plugins, activate the completion feature by pressing *Ctrl+Space*.

3.6.3.2. Defining the workspace editor

**Prerequisites**

- An existing workspace defined on this instance of CodeReady Workspaces.

**Procedure**

To define the editor to use with the workspace:

1. Click the *Devfile* tab.
2. Add the desired cheEditor component and click the Save button.

NOTE
To see a list of available plug-ins, activate the completion feature by pressing Ctrl+Space. The recommended editor for CodeReady Workspaces 2.15 is Che-Theia.

Additional resources
- Section 4.2, “Authoring a devfile 2”

3.7. RUNNING AN EXISTING WORKSPACE FROM THE USER DASHBOARD

This section describes how to run an existing workspace from the user dashboard.

3.7.1. 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
- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure
1. Navigate to the CodeReady Workspaces Dashboard. See Section 1.1, “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.
3.7.2. 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

- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See Section 1.1, “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. The workspace is started, and a browser navigates to the workspace.

3.7.3. 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

- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure

1. Navigate to the CodeReady Workspaces Dashboard. See Section 1.1, “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.

3.8. IMPORTING OPENShift APPLICATIONS INTO A WORKSPACE

To deploy a new instance of an application in a CodeReady Workspaces workspace, use one of the following scenarios:
3.8.1. Including a OpenShift application in a workspace devfile definition

This procedure describes how to define a CodeReady Workspaces workspace devfile to include a OpenShift application.

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:

```bash
$ 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}
```

**Prerequisites**

- You are logged in to the cluster with a running instance of Red Hat CodeReady Workspaces. To install an instance of Red Hat CodeReady Workspaces, see [Installing CodeReady Workspaces](#).
- The crwctl management tool is available. See the [Using the crwctl management tool](#) section.

**Procedure**

1. Create the simplest devfile:

   apiVersion: 1.0.0
   metadata:
     name: minimal-workspace

   Specify the name **minimal-workspace**. 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, including, for example, the terminal.

2. To add OpenShift applications to a workspace, modify the devfile and add the Kubernetes component type.
   For example, to embed the NodeJS-Mongo application in the **minimal-workspace**:

   apiVersion: 1.0.0
   metadata:
     name: minimal-workspace
   components:
The `sleep infinity` command is added as the entrypoint of the Node.js application. The command prevents the application from starting at the workspace start phase. This configuration allows the user to start the application when needed for testing or debugging purposes.

3. Add the commands in the devfile to make it easier for a developer to test the application:

```yaml
apiVersion: 1.0.0
metadata:
  name: nodejs-with-db
projects:
  - name: nodejs-mongo-app
    source:
      type: git
      location: 'https://github.com/ijason/NodeJS-Sample-App.git'
      commitId: 187d468 # refers to the last commitId the project compiles (with express3)
components:
  - type: openshift
  - alias: nodejs-app
    type: openshift
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
```

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.

4. Use the devfile to create and start a workspace:

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

Additional resources

- Chapter 4, Authoring devfiles.
3.8.2. 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

- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure

1. After the creation of a workspace, use the Workspace menu and then click on the desired workspace.
2. Modify the workspace devfile, use the Devfile tab.
3. Add a OpenShift component.
4. For the changes to take effect, save the devfile and restart the CodeReady Workspaces workspace.

3.8.3. 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

- You are logged in to CodeReady Workspaces. See How to login into CodeReady Workspaces using crwctl

Procedure

1. To generate a devfile, use:

   ```
   $ crwctl devfile:generate
   ```

   It is also possible to generate a devfile from, for example, the NodeJS-MongoDB application that includes the NodeJS component, using the crwctl devfile:generate command:

   **Example:**

   ```
   $ crwctl devfile:generate --selector="app=nodejs"
   apiVersion: 1.0.0
   ```
The Node.js application YAML definition is available 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"
  ```

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

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

### 3.9. 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, `oc`. 
3.9.1. Remotely accessing workspaces using oc

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

Prerequisites

- The oc, version 1.5.0 or higher, is available. For information about installed version, use:

```bash
$ oc version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0"
```

Procedure

In the example below:

- `workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4` is the name of the Pod.
- `crw` is the project.

1. To find the name of the OpenShift project and the Pod that runs the CodeReady Workspaces workspace:
   ```bash
   $ oc get pod -l che.workspace_id --all-namespaces
   NAMESPACE NAME                                               READY STATUS    RESTARTS AGE
   crw         workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4   4/4     Running   0 6m4s
   ```

2. To find the name of the container:
   ```bash
   $ NAMESPACE=crw
   $ 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
   ```

3. When you have the project, Pod name, and the name of the container, use the `oc` command to open a remote shell prompt:
   ```bash
   $ NAMESPACE=crw
   $ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
   $ CONTAINER=maven
   $ oc exec -ti -n $NAMESPACE $POD -c $CONTAINER bash
   user@workspace7b2wemdf3hx7s3ln$
   ```

4. 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 `oc`, see the Getting started with the CLI.

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

This procedure describes how to use the `oc` tool to download or upload files remotely from or to an CodeReady Workspaces workspace.

Prerequisites

- Remote access to the CodeReady Workspaces workspace you intend to modify. See Remotely accessing workspaces.
- The `oc`, version 1.5.0 or higher, is available. For information about installed version, use:

  ```bash
  $ oc version
  Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0"}
  ...
  
  Procedure

The following procedure uses `crw` as an example of a user project.

- 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 prompt.

  ```bash
  $ REMOTE_FILE_PATH=/projects/downloadme.txt
  $ NAMESPACE=crw
  $ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
  $ CONTAINER=maven
  $ oc cp ${NAMESPACE}/${POD}:${REMOTE_FILE_PATH} ~/downloadme.txt -c ${CONTAINER}
  
  Using the preceding steps, the user can also download and upload directories.

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

  ```bash
  $ LOCAL_FILE_PATH=./uploadme.txt
  $ NAMESPACE=crw
  $ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
  $ CONTAINER=maven
  $ oc cp ${LOCAL_FILE_PATH} ${NAMESPACE}/${POD}:/projects -c ${CONTAINER}
  ```
3.10. 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.

Do not mount secrets that you define in a devfile.

The following section describes how to automatically mount a 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
- Git credentials store file

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.

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

**WARNING**

On OpenShift 3.11, secrets mounted as file overrides volume mounts defined in the devfile.
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 CodeReady Workspaces. To install an instance of CodeReady Workspaces, see [Installing CodeReady Workspaces](#).

**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:**

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: mvn-settings-secret
     labels:
       app.kubernetes.io/part-of: che.eclipse.org
       app.kubernetes.io/component: workspace-secret
   
   apiVersion: v1
   kind: Secret
   metadata:
     name: mvn-settings-secret
     annotations:
       che.eclipse.org/target-container: maven
       che.eclipse.org/mount-path: {prod-home}/.m2/
       che.eclipse.org/mount-as: file
     labels:
   
   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.
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.

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

This results in a file named `settings.xml` being mounted at the `/home/jboss/.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.
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.

3.10.2. 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


Procedure

1. In the OpenShift project where a CodeReady Workspaces workspace will be created, generate a new OpenShift secret.

   - The labels of the secret that is about to be generated 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:**

   ```yaml
   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.

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
   ```
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 the 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.

```yaml
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:

```yaml
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 OpenShift 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.

### 3.10.3. Mounting a Git credentials store into a workspace container

This section describes how to mount Git credentials store as secret from the user’s project into the file in single-workspace or multiple-workspace containers of CodeReady Workspaces.

**Prerequisites**

- A running instance of CodeReady Workspaces. To install an instance of CodeReady Workspaces, see [Installing CodeReady Workspaces](#).

**Procedure**

1. Prepare Git credential file in the Storage format.
2. Encode content of the file to the base64 format.
3. 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`

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

Kubernetes 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.

### 3.11. AUTHENTICATING USERS ON PRIVATE REPOSITORIES OF SCM SERVERS

The following section describes how to configure user authentications for SCM servers.

- Section 3.11.1, “Authenticating on Bitbucket servers”
- Section 3.11.2, “Authenticating on GitLab servers”
- Section 3.11.3, “Authenticating on GitHub servers”

#### 3.11.1. Authenticating on Bitbucket servers

The following section describes the configuration needed to enable CodeReady Workspaces to use Bitbucket (BB) repositories with Git cloning operations.

BB authentication is based on using personal access tokens. Each BB user is able to request several personal access tokens with different attributes, such as names, permissions, or expiration times. Those tokens can also be used to sign BB REST API calls and perform Git repository operations.

- CodeReady Workspaces users may use public or private repositories on the BB Source Code Management (SCM) system as a source of their projects.

**NOTE**

Before configuring the OAuth 1 authentication that is required for reading from private repositories and writing to both private and public repositories, configure the BB server with CodeReady Workspaces first. To do so, see the Prerequisites section below.

**Additional resources**


**Prerequisites**
A BB endpoint has been registered with CodeReady Workspaces. Follow the Configuring Bitbucket servers procedure and register a BB server using the CHE_INTEGRATION_BITBUCKET_SERVER__ENDPOINTS YAML secret property.

Procedure

1. Configure the OAuth 1 authentication by following the Configuring Bitbucket Server OAuth 1 procedure.

3.11.2. Authenticating on GitLab servers

Configuring authentication on the GitLab system is similar to Bitbucket.

GitLab authentication is based on using personal access tokens. Each GitLab user is able to request several personal access tokens with different names, permissions, expiration times, and so on. Those tokens can be used to sign GitLab REST API calls and perform Git repository operations.

See the GitLab documentation for more details about personal access tokens.

To allow GitLab authentication on CodeReady Workspaces side, personal tokens must be stored in the user’s project in the form of a secret. The secret must look as follows:

Example 3.10. GitLab personal access token secret

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-personal-access-token-secret
labels:
  app.kubernetes.io/part-of: che.eclipse.org
  app.kubernetes.io/component: scm-personal-access-token
annotations:
  che.eclipse.org/expired-after: '-1'
  che.eclipse.org/che-userid: '355d1ce5-990e-401e-9a8c-094bca10b5b3'
  che.eclipse.org/scm-userid: '2'
  che.eclipse.org/scm-username: 'user-foo'
  che.eclipse.org/scm-url: 'https://gitlab.apps.cluster-example.com'
data:
  token: Yzh5cEt6cURxUWVCa3FKazhtaHg=
```

The main parts of the secret are:

<table>
<thead>
<tr>
<th>Label</th>
<th>app.kubernetes.io/component</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>app.kubernetes.io/component</td>
<td>Annotation</td>
<td>Description</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/che-userid</td>
<td>che.eclipse.org/che-userid</td>
<td>Red Hat CodeReady Workspaces id of the user token belongs to</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-userid</td>
<td>che.eclipse.org/scm-userid</td>
<td>GitLab user id to which token belongs</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-username</td>
<td>GitLab user name to which token belongs</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-url</td>
<td>GitLab server URL to which this token belong</td>
<td></td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/expired-after</td>
<td>Personal access token expiration time</td>
<td></td>
</tr>
<tr>
<td>Data entry</td>
<td>token</td>
<td>Base-64 encoded value of the personal access token</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

Encoding a string into the base64 format using the `base64` tool on Linux machines leads to adding the newline character to the end of the source string and causing a value to be unusable as the authentication header value after decoding. Avoid this by using `base64 -w0`, which removes newly added lines, or strip newlines explicitly using `tr -d \n`.

1. To obtain a user ID from a secret, take a look into user profile page on GitLab web UI or make a call to a REST API URL:

   - For GitLab:
     ```
     ```
   - For CodeReady Workspaces
     ```
     https://codeready-<openshift_deployment_name>.<domain_name>/api/user
     ```
   - With the token credentials obtained from a secret, another secret is automatically created, allowing authorization to Git operations. This secret is mounted into a workspace container as a Git credentials file, and any additional configurations are not required to work with private Git repositories.


### 3.11.3. Authenticating on GitHub servers

Configuring authentication on the GitHub system is similar to GitLab.

GitHub authentication can be based on using personal access tokens. Each GitHub user is able to request several personal access tokens with different names, permissions, expiration times, and so on. Those tokens can be used to sign GitHub REST API calls and perform Git repository operations.

To allow GitHub authentication on CodeReady Workspaces side, personal tokens must be stored in the user’s project in the form of a secret. The secret must look as follows:

**Example 3.11. GitHub personal access token secret**
The main parts of the secret are:

<table>
<thead>
<tr>
<th>Label</th>
<th>app.kubernetes.io/component</th>
<th>Indicates it is an SCM personal token secret.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/che-userid</td>
<td>Red Hat CodeReady Workspaces id of the user token belongs to</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-userid</td>
<td>GitHub user id to which token belongs</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-username</td>
<td>GitHub username to which token belongs</td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/scm-url</td>
<td>GitHub server URL to which this token belongs. Typically, it is <a href="https://github.com">https://github.com</a></td>
</tr>
<tr>
<td>Annotation</td>
<td>che.eclipse.org/expired-after</td>
<td>Personal access token expiration time</td>
</tr>
<tr>
<td>Data entry</td>
<td>token</td>
<td>Base-64 encoded value of the personal access token</td>
</tr>
</tbody>
</table>

**NOTE**

Encoding a string into the base64 format using the `base64` tool on Linux machines leads to adding the newline character to the end of the source string and causing a value to be unusable as the authentication header value after decoding. Avoid this by using `base64 -w0`, which removes newly added lines, or strip newlines explicitly using `tr -d \n`.

1. To obtain a user ID from a secret, make a call to a REST API URL:

   ```
   https://api.github.com/user
   ```
For CodeReady Workspaces

\(https://codeready-<openshift_deployment_name>.<domain_name>/api/user\)

With the token credentials obtained from a secret, another secret is automatically created, allowing authorization to Git operations. This secret is mounted into a workspace container as a Git credentials file, and any additional configurations are not required to work with private Git repositories.

4.1. AUTHORING DEVFILES VERSION 1

This section explains the concept of a devfile and how to configure a CodeReady Workspaces workspace by using a devfile of the 1.0 specification.

4.1.1. What is a devfile

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

- The source code.
- The development components, such as browser IDE tools and application runtimes.
- A list of pre-defined commands.
- Projects to clone.

A devfile is a YAML file that CodeReady Workspaces consumes and transforms into a cloud workspace composed of multiple containers. It is possible to store a devfile remotely or locally, in any number of ways, such as:

- In a Git repository, in the root folder, or on a feature branch.
- On a publicly accessible web server, accessible through HTTP.
- Locally as a file, and deployed using `crwctl`.
- In a collection of devfiles, known as a devfile registry.

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:

- Elements of the project, such as the path, Git location, or branch.
- Commands to perform daily tasks such as build, run, test, and debug.
- The runtime environment with its container images needed for the application to run.
- Che-Theia plug-ins with tools, IDE features, and helpers that a developer would use in the workspace, for example, Git, Java support, SonarLint, and Pull Request.

4.1.2. A minimal devfile
The following is the minimum content required in a devfile:

- apiVersion
- metadata name

apiVersion: 1.0.0
metadata:
  name: crw-in-crw-out

For a complete devfile example, see Red Hat CodeReady Workspaces in CodeReady Workspaces devfile.yaml.

NOTE

A choice of use of the parameter `generateName` or `name` is optional, but only one of these parameters has to be chosen by a user and defined. When both attributes are specified, `generateName` is ignored. See Section 4.1.3, “Generating workspace names”.

```
metadata:
  generatedName:
```

or

```
metadata:
  name:
```

4.1.3. Generating workspace names

To specify a prefix for automatically generated workspace names, set the `generateName` parameter in the devfile:

apiVersion: 1.0.0
metadata:
  generateName: crw-

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.
4.1.4. 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.

4.1.4.1. 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

```yaml
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 Exec` plug-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

```yaml
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
```

4.1.4.2. Specifying multiple projects in a devfile

A single devfile can define multiple projects, which are cloned to the desired destination. These projects are created inside a user’s workspace after the workspace is started.

For each project, specify the following:
The type of the source repository - this can be **git** or **zip**. For additional information, see the Devfile reference section.

The location of the source repository - an URL to a Git repository or **zip** archive.

Optionally, the directory to which the project is cloned. If none is specified, the default directory is used, which is a directory that matches the project name or project Git repository.

**Example of a devfile with two projects**

In the following example, the projects **frontend** and **backend** act as examples of a user’s projects. Each project is located in a separate repository.

- The **backend** project has a specific requirement to be cloned into the `src/github.com/<github-organization>/<backend>` directory under the source root, implicitly defined by the CodeReady Workspaces runtime.

- The **frontend** project will be cloned into the `<frontend/>` directory under the source root.

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

**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.

**4.1.5. Devfile reference**

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

**4.1.5.1. Adding schema version to a devfile**
Procedure

- Define the `schemaVersion` attribute in the devfile:

Example 4.1. Adding schema version to a devfile

```yaml
schemaVersion: 1.0.0
```

4.1.5.2. Adding a name to a devfile

Adding a name to a devfile is mandatory. Both `name` and `generateName` are optional attributes, but at least one of them must be defined.

Procedure

1. To specify a static name for the workspace, define the `name` attribute.

   Adding a static name to a devfile

   ```yaml
   schemaVersion: 1.0.0
   metadata:
     name: devfile-sample
   ```

2. To specify a prefix for automatically generated workspace names, define the `generateName` attribute and do not define the `name` attribute. The workspace name will be in the `<generateName>YYYYY` format, for example, `devfile-sample-2y7kp`, where `Y` is a random [a-z0-9] character.

   Adding a generated name to a devfile

   ```yaml
   schemaVersion: 1.0.0
   metadata:
     generateName: devfile-sample-
   ```

   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.

4.1.5.3. Adding projects to a devfile

A devfile is designed to contain 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.

**4.1.5.3.1. Project-source type: git**

<table>
<thead>
<tr>
<th>source</th>
<th>type: git</th>
<th>location: <a href="https://github.com/eclipse-che/che-server.git">https://github.com/eclipse-che/che-server.git</a></th>
<th>startPoint: main</th>
<th>tag: 7.34.0</th>
<th>commitId: 36fe587</th>
<th>branch: 7.34.x</th>
<th>sparseCheckoutDir: core</th>
</tr>
</thead>
</table>

1. **startPoint**: 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 a single directory, is needed.

**Example 4.2. 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](#).

**4.1.5.3.2. Project-source type: zip**
4.1.5.3. 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

```yaml
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
```

4.1.5.4. Adding components to a devfile

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

4.1.5.4.1. 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.

```yaml
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.

4.1.5.4.2. Component type: chePlugin

Describes plug-ins in a workspace by defining their `id`. A devfile is allowed to have multiple `chePlugin` components.
Both types above use an ID, which is slash-separated publisher, name and version of plug-in from the CodeReady Workspaces Plug-in registry. Note that the CodeReady Workspaces Plug-in registry uses the latest version by default for all plug-ins.


4.1.5.4.3. 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/latest
```

4.1.5.4.4. Specifying a component by linking to its descriptor

Rather than using the editor or plug-in id to specify cheEditor or chePlugin, provide a direct link to the component descriptor, typically named as meta.yaml, using the reference field:

```
components:
  - alias: exec-plugin
    type: chePlugin
    reference: https://raw.githubusercontent.com.../plugin/1.0.1/meta.yaml
    id: redhat/java/latest
    type: chePlugin
    preferences:
      java.jdt.ls.vmargs: '-noverify -Xmx1G -XX:+UseG1GC -XX:+UseStringDeduplication'
```

The URL in the reference field must be publicly accessible and should directly point to a fetchable meta.yaml file. URLs that redirect or do not directly point to a meta.yaml file will cause the workspace startup to fail. To learn more about publishing meta.yaml files, see Section 5.4, “Publishing metadata for a Visual Studio Code extension”.

**NOTE**

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

4.1.5.4.5. 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.
Preferences may also be specified as an array:

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

### 4.1.5.4.6. 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.

```yaml
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` object can be embedded into the devfile using the `referenceContent` field:

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

### 4.1.5.4.7. Overriding container entrypoints

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 entrypoint changes to.

The entrypoints can be defined as follows:

```yaml
components:
  - alias: appDeployment
type: kubernetes
The `entrypoints` 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.

### 4.1.5.4.8. Overriding container environment variables

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

```yaml
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.

### 4.1.5.4.9. Specifying mount-source option

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

```yaml
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.

4.1.5.4.10. Component type: dockerimage

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

Example of a minimal dockerimage component

```json
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
- type: dockerimage
  image: quay.io/eclipse/che-java11-maven:nightly
  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
  memoryRequest: 256M
  command: ["tail"]
  args: ['-f', '/dev/null']
```

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.

4.1.5.4.11. Mounting project sources

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

### 4.1.5.4.12. 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, and so on.

### 4.1.5.4.13. 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:

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

Example specifying volumes for `cheEditor/chePlugin` type:

```yaml
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: cheEditor
    alias: theia-editor
    id: eclipse/che-theia/next
    env:
      - name: HOME
        value: 
```
Example specifying volumes for **kubernetes/openshift** type:

```yaml
volumes:
  - name: cache
    containerPath: /.cache
```

### 4.1.5.4.14. Specifying container memory limit for components

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

```yaml
components:
  - alias: exec-plugin
    type: chePlugin
    id: eclipse/che-machine-exec-plugin/latest
    memoryLimit: 1Gi
  - alias: maven
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    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_SIDECAR_DEFAULT__MEMORY__LIMIT__MB` system property).

### 4.1.5.4.15. Specifying container memory request for components

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

```yaml
components:
  - alias: exec-plugin
    type: chePlugin
    id: eclipse/che-machine-exec-plugin/latest
    memoryLimit: 1Gi
    memoryRequest: 512M
  - alias: maven
    type: dockerimage
```
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_SIDECAR_DEFAULT__MEMORY__REQUEST__MB system property).

4.1.5.4.16. 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/latest
    cpuLimit: 1.5
  - alias: maven
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    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).

4.1.5.4.17. 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/latest
    cpuLimit: 1.5
    cpuRequest: 0.225
  - alias: maven
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    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).

### 4.1.5.4.18. 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.

```yaml
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 Kubernetes 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 to which the workspace is deployed.

- **CHE_MACHINE_TOKEN**: The token used to authenticate the request against the CodeReady Workspaces server.

- **CHE_MACHINE_AUTH_SIGNATURE__PUBLIC__KEY**: 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 devfile might 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. The environment variables for securely accessing the CodeReady Workspaces server are out of scope for devfiles. These variables are available only to CodeReady Workspaces plug-ins, which use them for advanced use cases.

### 4.1.5.4.19. 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 Kubernetes 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 need to be able to directly interact with it yourself or you allow other components to interact with it.

Endpoints have several properties as shown in the following example:

```yaml
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
```
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 hostname within the workspace containers (in the OpenShift terminology, 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 value (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 loop-back interface, such as 127.0.0.1.

- **path**: The path portion of the URL to the endpoint. This defaults to /, meaning that the endpoint is assumed to be accessible at the web root of the server defined by the component.

- **unsecuredPaths**: A comma-separated list of endpoint paths that are to stay unsecured even if the secure attribute is set to true.

---

**WARNING**

Listening on any other interface than the local loop-back poses a security risk because such server is accessible without the JWT authentication within the cluster network on the corresponding IP addresses.
- **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 automatically detects this, and the UI offers to expose this port as a **public** port automatically. This behavior is useful for debugging a web application. It is impossible to do this for servers, such as a database server, which automatically starts at the container start. For such components, specify the endpoints explicitly.

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

```yaml
apiVersion: 1.0
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:
```

Red Hat CodeReady Workspaces 2.15 End-user Guide
To describe complex deployments, include references to OpenShift resource lists in the devfile. The OpenShift resource lists become a part of the workspace.

**IMPORTANT**

- CodeReady Workspaces merges all resources from the OpenShift resource lists into a single deployment.
- Be careful when designing such lists to avoid name conflicts and other problems.

**Table 4.1. Supported OpenShift resources**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Supported resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenShift</td>
<td>deployments, pods, services, persistent volume claims, secrets, ConfigMaps, Routes</td>
</tr>
</tbody>
</table>

```
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.

```yaml
apiVersion: v1
kind: List
items:
  - apiVersion: v1
type: Deployment
metadata:
  name: postgres
  labels:
    app: postgres
spec:
type: template
metadata:
  name: postgres
  app:
    name: postgres
spec:
  type: containers
  - name: postgres
    image: 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
type: Service
metadata:
  name: postgres
  labels:
    app: postgres
spec:
  type: ports
  - port: 5432
    targetPort: 5432
  selector:
    app: postgres
- apiVersion: v1
  kind: PersistentVolumeClaim
metadata:
  name: pg-storage
  labels:
    app: postgres
spec:
  accessModes:
  - ReadWriteOnce
```
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).

Additionally, you can modify the entrypoints (command and arguments) of the containers in the resource list.

### 4.1.5.5. 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.

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 Visual Studio Code).
4.1.5.5.1. CodeReady Workspaces-specific commands

Each CodeReady Workspaces-specific command features:

- An **actions** attribute that specifies 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.

```json
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
- name: my-go-project
c  clonePath: go/src/github.com/acme/my-go-project
c  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.

4.1.5.5.2. 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 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.
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 start configurations.

4.1.5.5.3. 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.

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 Route.

4.1.5.5.3.1. 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)

4.1.5.6. Adding attributes to a devfile

Devfile attributes can be used to configure various features.

4.1.5.6.1. 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
    reference: my-app.yaml
attributes:
  editorFree: true

4.1.5.6.2. 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
4.1.5.6.3. Attribute: asyncPersist (asynchronous storage)

When `persistVolumes` is set to `false` (see above), the additional attribute `asyncPersist` can be set to `true` to enable asynchronous storage. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-storage-types.adoc for more details.

Example of a devfile with asynchronous storage enabled

```yaml
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
  asyncPersist: true
```

4.1.5.6.4. Attribute: mergePlugins

This property can be set to manually control how plug-ins are included in the workspace. When the property `mergePlugins` is set to `true`, Che will attempt to avoid running multiple instances of the same container by combining plugins. The default value when this property is not included in a devfile is governed by the Che configuration property `che.workspace.plugin_broker.default_merge_plugins`; adding the `mergePlugins: false` attribute to a devfile will disable plug-in merging for that workspace.

Example of a devfile with plug-in merging disabled

```yaml
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:
  mergePlugins: false
```
4.1.6. Objects supported in Red Hat CodeReady Workspaces 2.15

The following table lists the objects that are partially supported in Red Hat CodeReady Workspaces 2.15:

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

4.2. AUTHORING A DEVFILE 2

When you author or edit a devfile for configuring a workspace, the devfile must meet the latest devfile 2 specification.

Prerequisites

• Follow the instructions in the Devfile User Guide.

Additional resources

• For more information about devfile object schema and object properties, see the Introduction to Devfiles.
CHAPTER 5. CUSTOMIZING DEVELOPER ENVIRONMENTS

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

You can extend Red Hat CodeReady Workspaces in three different ways:

- **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 IDE (web or desktop based). 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 5.1. CodeReady Workspaces extensibility

A user can extend CodeReady Workspaces by using **self-hosted** mode, which CodeReady Workspaces provides by default.

Section 5.1, “What is a Che-Theia plug-in”

Section 5.6, “Using alternative IDEs in CodeReady Workspaces”

Section 5.2, “Adding a Visual Studio Code extension to a workspace”

Section 5.10, “Using private container registries”

5.1. 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.
5.1.1. Features and benefits of Che-Theia plug-ins

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Loading</td>
<td>Plug-ins are loaded at runtime and are already compiled. IDE is loading the plug-in code.</td>
<td>Avoid any compilation time. Avoid post-installation steps.</td>
</tr>
<tr>
<td>Secure Loading</td>
<td>Plug-ins are loaded separately from the IDE. The IDE stays always in a usable state.</td>
<td>Plug-ins do not break the whole IDE if it has bugs. Handle network issue.</td>
</tr>
<tr>
<td>Tools Dependencies</td>
<td>Dependencies for the plug-in are packaged with the plug-in in its own container.</td>
<td>No-installation for tools. Dependencies running into container.</td>
</tr>
<tr>
<td>Code Isolation</td>
<td>Guarantee that plug-ins cannot block the main functions of the IDE like opening a file or typing</td>
<td>Plug-ins are running into separate threads. Avoid dependencies mismatch.</td>
</tr>
</tbody>
</table>

5.1.2. 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.

5.1.2.1. 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 5.2. 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).

5.1.2.2. Che-Theia plug-in APIs

To provide 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 Visual Studio 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.

5.1.2.3. Che-Theia plug-in capabilities

Che-Theia plug-ins have the following capabilities:

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CodeReady Workspaces</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Extended Tasks</strong></td>
<td>Handles the CodeReady Workspaces commands and provides the ability to start those into a specific container of the workspace.</td>
<td>Task plug-in</td>
</tr>
<tr>
<td><strong>Extended Terminal</strong></td>
<td>Allows to provide terminal for any of the containers of the workspace.</td>
<td>Extended Terminal extension</td>
</tr>
<tr>
<td><strong>Factory</strong></td>
<td>Handles the Red Hat CodeReady Workspaces Factories</td>
<td>Workspace plug-in</td>
</tr>
<tr>
<td><strong>Container</strong></td>
<td>Provides a container view that shows all the containers that are running in the workspace and allows to interact with them.</td>
<td>Containers plug-in</td>
</tr>
</tbody>
</table>
5.1.2.4. Visual Studio Code extensions and Eclipse Theia plug-ins

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

A Visual Studio Code extension

To repackage a Visual Studio 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 Section 5.2, “Adding a Visual Studio Code extension to a workspace”.

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 5.1.5, “Embedded and remote Che-Theia plug-ins”

5.1.3. 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. These files can be referenced in a devfile to include Che-Theia plug-ins in a workspace.

Here is an overview of all fields that can be available in plug-in meta YAML files. This document represents the plugin meta YAML structure (version 3).

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion</td>
<td>Version 2 and higher where version is 1 supported for backwards compatibility</td>
</tr>
<tr>
<td>category</td>
<td>Available: Category must be set to one of the followings: Editor, Debugger, Language, Linter, Snippet, Theme, Other</td>
</tr>
<tr>
<td>description</td>
<td>Short description of the plug-in purpose</td>
</tr>
<tr>
<td>displayName</td>
<td>Name shown in user dashboard</td>
</tr>
</tbody>
</table>

Table 5.1. meta.yaml
### deprecate
Optional; section for deprecating plug-ins in favor of others

* autoMigrate - boolean
* migrateTo - new org/plugin-id/version, for example redhat/vscode-apache-camel/latest

### firstPublicationDate
Not required to be in YAML; if it is not included, the plug-in registry dockerimage build generates it

### latestUpdateDate
Not required to be in YAML; if it is not included, the plug-in registry dockerimage build generates it

### icon
URL of an SVG or PNG icon

### name
Name (no spaces allowed), must match [-a-z0-9]

### publisher
Name of the publisher, must match [-a-z0-9]

### repository
URL for plug-in repository, for example, GitHub

### title
Plug-in title (long)

### type
Che Plugin, Visual Studio Code extension

### version
Version information, for example: 7.5.1, [-a-z0-9]

### spec
Specifications (see below)

#### Table 5.2. spec attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpoints</td>
<td>Optional; plug-in endpoint</td>
</tr>
<tr>
<td>containers</td>
<td>Optional; sidecar containers for the plug-in. Che Plug-in and Visual Studio Code extension supports only one container</td>
</tr>
<tr>
<td>initContainers</td>
<td>Optional; sidecar init containers for the plug-in</td>
</tr>
<tr>
<td>workspaceEnv</td>
<td>Optional; environment variables for the workspace</td>
</tr>
<tr>
<td>extensions</td>
<td>Optional; Attribute that is required for Visual Studio Code and Che-Theia plug-ins in a form list of URLs to plug-in artefacts, such as .vsix or .theia files</td>
</tr>
</tbody>
</table>

#### Table 5.3. spec.containers. Notice: spec.initContainers has absolutely the same container definition.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Sidecar container name</td>
</tr>
<tr>
<td>image</td>
<td>Absolute or relative container image URL</td>
</tr>
<tr>
<td>memoryLimit</td>
<td>OpenShift memory limit string, for example 512Mi</td>
</tr>
<tr>
<td>memoryRequest</td>
<td>OpenShift memory request string, for example 512Mi</td>
</tr>
<tr>
<td>cpuLimit</td>
<td>OpenShift CPU limit string, for example 2500m</td>
</tr>
<tr>
<td>cpuRequest</td>
<td>OpenShift CPU request string, for example 125m</td>
</tr>
<tr>
<td>env</td>
<td>List of environment variables to set in the sidecar</td>
</tr>
<tr>
<td>command</td>
<td>String array definition of the root process command in the container</td>
</tr>
<tr>
<td>args</td>
<td>String array arguments for the root process command in the container</td>
</tr>
<tr>
<td>volumes</td>
<td>Volumes required by the plug-in</td>
</tr>
<tr>
<td>ports</td>
<td>Ports exposed by the plug-in (on the container)</td>
</tr>
<tr>
<td>commands</td>
<td>Development commands available to the plug-in container</td>
</tr>
<tr>
<td>mountSources</td>
<td>Boolean flag to bind volume with source code /projects to the plug-in container</td>
</tr>
<tr>
<td>initContainers</td>
<td>Optional; init containers for sidecar plug-in</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Container lifecycle hooks. See lifecycle description</td>
</tr>
</tbody>
</table>

Table 5.4. **spec.containers.env** and **spec.initContainers.env** attributes. Notice: **workspaceEnv** has absolutely the same attributes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Environment variable name</td>
</tr>
<tr>
<td>value</td>
<td>Environment variable value</td>
</tr>
</tbody>
</table>

Table 5.5. **spec.containers.volumes** and **spec.initContainers.volumes** attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mountPath</td>
<td>Path to the volume in the container</td>
</tr>
<tr>
<td>name</td>
<td>Volume name</td>
</tr>
</tbody>
</table>
### ephemeral

| ephemeral | If true, the volume is ephemeral, otherwise the volume is persisted |

**Table 5.6.** `spec.containers.ports` and `spec.initContainers.ports` attributes

| exposedPort | Exposed port |

**Table 5.7.** `spec.containers.commands` and `spec.initContainers.commands` attributes

| name | Command name |
| workingDir | Command working directory |
| command | String array that defines the development command |

**Table 5.8.** `spec.endpoints` attributes

| name | Name (no spaces allowed), must match [-a-z0-9] |
| public | true, false |
| targetPort | Target port |
| attributes | Endpoint attributes |

**Table 5.9.** `spec.endpoints.attributes` attributes

| protocol | Protocol, example: ws |
| type | ide, ide-dev |
| discoverable | true, false |
| secure | true, false. If true, then the endpoint is assumed to listen solely on 127.0.0.1 and is exposed using a JWT proxy |
| cookiesAuthEnabled | true, false |
| requireSubdomain | true, false. If true, the endpoint is exposed on subdomain in single-host mode. |

**Table 5.10.** `spec.containers.lifecycle` and `spec.initContainers.lifecycle` attributes
The **postStart** event that runs immediately after a Container is started. See postStart and preStop handlers

- **exec**: Executes a specific command, resources consumed by the command are counted against the Container
- **command**: `"/bin/sh", "-c", "/bin/post-start.sh"`

The **preStop** event that runs before a Container is terminated. See postStart and preStop handlers

- **exec**: Executes a specific command, resources consumed by the command are counted against the Container
- **command**: `"/bin/sh", "-c", "/bin/post-start.sh"`

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

```yaml
apiVersion: v2
publisher: eclipse
name: che-machine-exec-plugin
version: 7.9.2

type: Che Plugin
displayName: CodeReady Workspaces machine-exec Service
title: Che machine-exec Service Plugin
description: CodeReady Workspaces Plug-in with che-machine-exec service to provide creation terminal or tasks for Eclipse CHE workspace containers.

icon: https://www.eclipse.org/che/images/logo-eclipseche.svg

repository: https://github.com/eclipse-che/che-machine-exec/

firstPublicationDate: "2020-03-18"
category: Other

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:7.9.2"
```
Every time a user starts a Che workspace, a Che-Theia plug-in life cycle process starts. The steps of this process are as follows:

1. CodeReady Workspaces server checks for plug-ins to start from the workspace definition.

2. CodeReady Workspaces server retrieves plug-in metadata, recognizes each plug-in type, and stores them in memory.

3. CodeReady Workspaces server selects a broker according to the plug-in type.

4. The broker processes the installation and deployment of the plug-in. The installation process of the plug-in differs for each specific broker.

**NOTE**

Plug-ins exist in various types. A broker ensures the success of a plug-in deployment by meeting all installation requirements.
Before a CodeReady Workspaces workspace is launched, CodeReady Workspaces server starts the workspace containers:

1. The Che-Theia plug-in broker extracts the information about sidecar containers that a particular plug-in needs from the .theia file.
2. The broker sends the appropriate container information to CodeReady Workspaces server.
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 server then starts all the containers of the workspace.
5. Che-Theia starts in its container and checks the correct folder to load the plug-ins.

**A user experience with Che-Theia plug-in lifecycle**

1. When a user opens a browser tab with Che-Theia, Che-Theia starts a new plug-in session with:
   - Web Worker for frontend
   - Node.js for backend
2. Che-Theia notifies all Che-Theia plug-ins with the start of the new session by calling the `start()` function for each triggered plug-in.
3. A Che-Theia plug-in session runs and interacts with the Che-Theia backend and frontend.
4. When the user closes the Che-Theia browser tab, or the session ended on a timeout limit, Che-Theia notifies all plug-ins with the `stop()` function for each triggered plug-in.

**5.1.5. 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.

Based on the required dependencies, Che-Theia plug-in can run as:

- Embedded, also know as local
- Remote

**5.1.5.1. Embedded (local) plug-ins**
The Embedded plug-ins are plug-ins without specific dependencies that are injected into the Che-Theia IDE. These plug-ins use the Node.js runtime, which runs in the IDE container.

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

To include a Che-Theia plug-in or Visual Studio Code extension, define a URL to the plug-in .theia archive binary in the `meta.yaml` file. See Section 5.2, “Adding a Visual Studio Code extension to a workspace”.

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.

### 5.1.5.2. 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 Visual Studio 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.

### 5.1.5.3. Comparison matrix

- Embedded plug-ins are those Che-Theia plug-ins or Visual Studio Code extensions that do not require extra dependencies inside its container.
- Remote plug-ins are containers that contain a plug-in with all required dependencies.

**Table 5.11. Che-Theia plug-in comparison matrix: embedded compared to remote**
Depending on your use case and the capabilities provided by your plug-in, select one of the described running modes.

### 5.1.6. Remote plug-in endpoint

Red Hat CodeReady Workspaces has a remote plug-in endpoint service to start Visual Studio 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.

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

To start a remote plug-in endpoint, set 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/jboss
ENV HOME=/home/jboss
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/jboss
ENV HOME=/home/jboss
RUN mkdir /projects 
  && chmod -R g+rwX /projects 
  && chmod -R g+rwX "${HOME}"
ENTRYPOINT ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}

5.1.6.1.1. Using a wrapper script

Some images use a wrapper script to configure permissions inside the container. The Dockerfile `ENTRYPOINT` command defines this script, which executes the main process defined in the `CMD` command of the Dockerfile.

CodeReady Workspaces uses images with a wrapper script to provide permission configurations to different infrastructures protected by advanced security. OpenShift Container Platform is an example of such an infrastructure.

- Example of a wrapper script:

```bash
#!/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}
```
Example of a Dockerfile with a wrapper script:

```
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}
```

Explanation:

- 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}`.
- The remote plug-in endpoint then starts in the container after permission configuration.

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

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

Procedure

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

- **command** - CodeReady Workspaces uses the `command` properties to override the Dockerfile#ENTRYPOINT value.
- **args** - CodeReady Workspaces uses the **args** properties to override the **Dockerfile#CMD** value.

- Example of a YAML file with the **command** and **args** properties modified:

```yaml
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: "Visual Studio 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:

```yaml
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: "Visual Studio 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
      - ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```
Red Hat CodeReady Workspaces calls the **entrypoint.sh** wrapper script defined in the **ENTRYPOINT** command of the Dockerfile. The script executes `[ 'sh', '-c", ' \n${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE} ]` using the `exec "$@"` command.

**NOTE**

By modifying the **command** and **args** properties of the **meta.yaml** file, a user can:

- Execute a service at a container start
- Start a remote plug-in endpoint

To make these actions run at the same time:

1. Start the service.
2. Detach the process.
3. Start the remote plug-in endpoint.

### 5.2. ADDING A VISUAL STUDIO CODE EXTENSION TO A WORKSPACE

This section describes how to add a Visual Studio Code extension to a workspace using the workspace configuration.

#### Prerequisites

- The Visual Studio Code extension is available in the CodeReady Workspaces plug-in registry, or metadata for the Visual Studio Code extension are available. See Section 5.4, “Publishing metadata for a Visual Studio Code extension”.

#### 5.2.1. Adding a Visual Studio Code extension using the workspace configuration

**Prerequisites**

- An existing workspace defined on this instance of CodeReady Workspaces.
- The Visual Studio Code extension is available in the CodeReady Workspaces plug-in registry, or metadata for the Visual Studio Code extension are available. See Section 5.4, “Publishing metadata for a Visual Studio Code extension”.

**Procedure**

To add a Visual Studio Code extension using the workspace configuration:

1. Click the **Workspaces** tab on the **Dashboard** and select the plug-in destination workspace. 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

   ID format: \texttt{<publisher>/<plug-inName>/<plug-inVersion>}

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

   Alternatively, you can link to a \texttt{meta.yaml} file hosted on GitHub, using the dedicated reference field.

   - type: chePlugin
     reference: 1


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

5.2.2. Adding a Visual Studio Code extension using recommendations

Prerequisites


- Featured Visual Studio Code extensions are available in the CodeReady Workspaces plug-in registry.

Procedure

Open a workspace without any existing devfile using the CodeReady Workspaces dashboard:

The recommendations plug-in will scan files, discover languages and install Visual Studio Code extensions matching these languages. Disable this feature by setting \texttt{extensions.ignoreRecommendations} to true in the devfile attributes.

The recommendations plug-in can suggest Visual Studio Code extensions to install when opening files. It suggests extensions based on the workspace content, allowing the user to work with the given files. Enable this feature by setting \texttt{extensions.openFileRecommendations} to true in the devfile attributes.

5.3. ADDING A VISUAL STUDIO CODE EXTENSION TO THE CODEREADY WORKSPACES PLUG-INS REGISTRY

To add or modify an extension in the CodeReady Workspaces plug-ins registry, edit the `che-theia-plugins.yaml` file and add relevant metadata.

**NOTE**

This article describes the steps needed to build the plug-ins registry with a custom plug-in definition. If you are looking to create a custom `meta.yaml` file that can be directly referenced in a devfile, see Section 5.4, “Publishing metadata for a Visual Studio Code extension”.

**Prerequisite**

- A working knowledge of customizing the registries, see [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#customizing-the-registries.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#customizing-the-registries.adoc)

- A link to a sidecar container image, should the Visual Studio Code extension require one.

**Procedure**

1. Edit the `che-theia-plugins.yaml` file and create a new entry.

   ```yaml
   - id: publisher/my-vscode-ext
     repository:
       url: https://github.com/publisher/my-vscode-ext
       revision: 1.7.2
     aliases:
       - publisher/my-vscode-ext-revised
     preferences:
       asciidoc.use_asciidoctopdf: true
       shellcheck.executablePath: /bin/shellcheck
       solargraph.bundlerPath: /usr/local/bin/bundle
       solargraph.commandPath: /usr/local/bundle/bin/solargraph
     sidecar:
       image: quay.io/repository/eclipse/che-plugin-sidecar:sonarlint-2fcf341
       name: my-vscode-ext-sidecar
       memoryLimit: "1500Mi"
       memoryRequest: "1000Mi"
       cpuLimit: "500m"
       cpuRequest: "125m"
       command:
         - /bin/sh
         - "/c"
         - "/entrypoint.sh"
     volumeMounts:
       - name: vscode-ext-volume
         path: "/home/theia/my-vscode-ext"
     endpoints:
       - name: "configuration-endpoint"
         public: true
         targetPort: 61436
   ```
attributes: protocol: http
extension: https://github.com/redhat-developer/vscode-yaml/releases/download/0.4.0/redhat.vscode-yaml-0.4.0.vsix

skipDependencies:
- id-of/extension1
- id-of/extension2
extraDependencies:
- id-of/extension1
- id-of/extension2
metaYaml:
skipIndex: <true|false>
skipDependencies:
- id-of/extension1
- id-of/extension2
extraDependencies:
- id-of/extension1
- id-of/extension2

1. (OPTIONAL) The ID of the plug-in, useful if a plug-in has multiple entries for one repository. For example, Java 8 and Java 11.
2. Repository information about the plug-in. If ID is specified, then this field is not a list element.
3. The URL to the Git repository of the extension.
4. Tag or SHA1 ID of the upstream repository that hosts the extension, corresponding to a version, snapshot, or release.
5. (OPTIONAL) An alias for this plug-in. For anything listed here, a meta.yaml file is generated.
6. (OPTIONAL) Plug-in preferences in freeform format.
7. (OPTIONAL) If the plug-in runs in a sidecar container, then the sidecar information is specified here.
8. A location of a container image to be used as the plug-in sidecar. This line cannot be specified concurrently with directory. See above.
9. (OPTIONAL) The name of the sidecar container.
10. (OPTIONAL) The memory limit of the sidecar container.
11. (OPTIONAL) The memory request of the sidecar container.
12. (OPTIONAL) The CPU limit of the sidecar container.
13. (OPTIONAL) The CPU request of the sidecar container.
14. (OPTIONAL) Definitions of root process commands inside the container.
15. (OPTIONAL) Arguments for root process commands inside the container.
16. (OPTIONAL) Any volume mounting information for the sidecar container.
17. The name of the mount.
The path to the mount.

(Optional) Any endpoint information for the sidecar container.

Endpoint name.

A Boolean value determining whether the endpoint is exposed publicly.

The port number.

Attributes relating to the endpoint.

Direct link or links to the vsix files included with the plug-in. The vsix built by the repository specified, such as the main extension, must be listed first.

# TODO #

(Optional) Extra dependencies in addition to the one listed in extensionDependencies field of package.json.

(Optional) Do not include this plug-in in index.json if true. Useful in case of dependencies that you do not want to expose as standalone plug-ins.

(Optional) Do not examine specified dependencies from extensionDependencies field of package.json (only for meta.yaml generation).

(Optional) Extra dependencies in addition to the one listed in extensionDependencies field of package.json (only for meta.yaml generation).

1. Run the build.sh script with the options of your choosing. The build process will generate meta.yaml files automatically, based on the entries in the che-theia-plugins.yaml file.

2. Use the resulting plug-ins registry image in CodeReady Workspaces, or copy the meta.yaml file out of the registry container and reference it directly as an HTTP resource.

5.4. PUBLISHING METADATA FOR A VISUAL STUDIO CODE EXTENSION


This article describes how to publish metadata for an additional extension, not available in the CodeReady Workspaces plug-ins registry, by using the extension configuration meta.yaml file.

For details on adding a plug-in to an existing plug-in registry, see Section 5.3, “Adding a Visual Studio Code extension to the CodeReady Workspaces plug-ins registry”

Prerequisite

- If the Visual Studio Code extension requires it, the required associated container image is available.

Procedure
1. Create a `meta.yaml` file.

2. Edit the `meta.yaml` file and provide the necessary information. The file must have the following structure:

```yaml
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  
  - https://github.com/SonarSource/sonarlint-vscode/releases/download/1.16.0/sonarlint-vscode-1.16.0.vsix
```

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, Visual Studio 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, Linter, Snippet, Theme, or Other**.
If this section is omitted, the Visual Studio Code extension is added into the Che-Theia IDE container.

The Docker image from which the sidecar container will be started. Example: codeready-workspaces/theia-endpoint-rhel8.

The maximum RAM which is available for the sidecar container. Example: "512Mi". This value might be overridden by the user in the component configuration.

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.

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.

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.

A list of Visual Studio Code extensions run in this sidecar container.

3. Publish the meta.yaml file as an HTTP resource by creating a gist on GitHub or GitLab with a file content published there.

5.5. TESTING A VISUAL STUDIO CODE EXTENSION IN CODEREADY WORKSPACES

Visual Studio Code (Visual Studio Code) extensions work in a workspace. Visual Studio Code extensions can run in the Che-Theia editor container, or in their own isolated and preconfigured containers with their prerequisites.

This section describes how to test a Visual Studio Code extension in CodeReady Workspaces with workspaces and how to review the compatibility of Visual Studio Code extensions to check whether a specific API is available.

NOTE

The extension-hosting sidecar container and the use of the extension in a devfile are optional.

5.5.1. Testing a Visual Studio Code extension using GitHub gist

Each workspace can have its own set of plug-ins. The list of plug-ins and the list of projects to clone are defined in the devfile.yaml file.

For example, to enable an AsciiDoc plug-in from the Red Hat CodeReady Workspaces dashboard, add the following snippet to the devfile:

```yaml
components:
  - id: joaopinto/vscode-asciidoctor/latest
type: chePlugin
```

To add a plug-in that is not in the default plug-in registry, build a custom plug-in registry. See
https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#customizing-the-registries.adoc, or, alternatively, use GitHub and the gist service.

Prerequisites

- A GitHub account.

Procedure

1. Go to the gist webpage and create a README.md file with the following description: Try Bracket Pair Colorizer extension in Red Hat CodeReady Workspaces and content: Example Visual Studio Code extension. (Bracket Pair Colorizer is a popular Visual Studio Code extension.)

2. Click the Create secret gist button.

3. Clone the gist repository by using the URL from the navigation bar of the browser:

   ```
   $ git clone https://gist.github.com/<your-github-username>/<gist-id>
   ```

   Example of the output of the git clone command

   ```
   git clone https://gist.github.com/benoitf/85c60c8c439177ac50141d527729b9d9
   Cloning into '85c60c8c439177ac50141d527729b9d9'...
   remote: Enumerating objects: 3, done.
   remote: Counting objects: 100% (3/3), done.
   remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
   Unpacking objects: 100% (3/3), done.
   ```

   Each gist has a unique ID.

4. Change the directory:

   ```
   $ cd <gist-directory-name>
   ```

   Directory name matching the gist ID.

5. Download the plug-in from the Visual Studio Code marketplace or from its GitHub page, and store the plug-in file in the cloned directory.

6. Create a plugin.yaml file in the cloned directory to add the definition of this plug-in.

   Example of the plugin.yaml file referencing the .vsix binary file extension

   ```
   apiVersion: v2
   publisher: CoenraadS
   name: bracket-pair-colorizer
   version: 1.0.61
   ```
This extension requires a basic Node.js runtime, so it is not necessary to add a custom runtime image in `plugin.yaml`.

`{{REPOSITORY}}` is a macro for a pre-commit hook.

7. Define a memory limit and volumes:

```yaml
spec:
  containers:
    - image: "quay.io/eclipse/che-sidecar-java:8-0cfbacb"
      name: vscode-java
      memoryLimit: "1500Mi"
      volumes:
        - mountPath: "/home/theia/.m2"
          name: m2
```

8. Create a `devfile.yaml` that references the `plugin.yaml` file:

```yaml
apiVersion: 1.0.0
metadata:
  generateName: java-maven-projects:
    - name: console-java-simple
      source:
        type: git
        location: "https://github.com/che-samples/console-java-simple.git"
        branch: java1.11
      components:
        - type: chePlugin
          id: redhat/java11/latest
          type: chePlugin
          reference: "{{REPOSITORY}}/plugin.yaml"
        - type: dockerimage
          alias: maven
          image: quay.io/eclipse/che-java11-maven:nightly
          memoryLimit: 512Mi
```
Any other devfile definition is also accepted. The important information in this devfile are the lines defining this external component. It means that an external reference defines the plug-in and not an ID, which pointing to a definition in the default plug-in registry.

9. Verify there are 4 files in the current Git directory:

```bash
$ ls -la
.git
CoenraadS.bracket-pair-colorizer-1.0.61.vsix
README.md
devfile.yaml
plugin.yaml
```

10. Before committing the files, add a pre-commit hook to update the {{REPOSITORY}} variable to the public external raw gist link:

a. Create a `.git/hooks/pre-commit` file with this content:

```bash
#!/bin/sh

# get modified files
FILES=$(git diff --cached --name-only --diff-filter=ACMR "*.yaml" | sed 's| |\ |g')

# exit fast if no files found
[ -z "$FILES" ] & & exit 0

# grab remote origin
origin=$(git config --get remote.origin.url)
url="${origin}/raw"

# iterate on files and add the good prefix pattern
for FILE in $(FILES); do
    sed -e "s#{{REPOSITORY}}#${url}#g" "$FILE" > "$FILE.back"
```
The hook replaces the \{(REPOSITORY)\} macro and adds the external raw link to the gist.

b. Make the script executable:

$ chmod u+x .git/hooks/pre-commit

11. Commit and push the files:

# Add files
$ git add *

# Commit
$ git commit -m "Initial Commit for the test of our extension"
[main 98dd370] Initial Commit for the test of our extension
3 files changed, 61 insertions(+)
create mode 100644 CoenraadS.bracket-pair-colorizer-1.0.61.vsix
create mode 100644 devfile.yaml
create mode 100644 plugin.yaml

# and push the files to the main branch
$ git push origin

12. Visit the gist website and verify that all links have the correct public URL and do not contain any \{(REPOSITORY)\} variables. To reach the devfile:

$ echo "$(git config --get remote.origin.url)/raw/devfile.yaml"

or:

$ echo "https://<che-server>/$(git config --get remote.origin.url)/raw/devfile.yaml"

5.5.2. Verifying the Visual Studio Code extension API compatibility level

Che-Theia does not fully support the Visual Studio Code extensions API. The vscode-theia-comparator is used to analyze the compatibility between the Che-Theia plug-in API and the Visual Studio Code extension API. This tool runs nightly, and the results are published on the vscode-theia-comparator GitHub page.

Prerequisites

- Personal GitHub access token. 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.

5.6. USING ALTERNATIVE IDES IN CODEREADY WORKSPACES

Red Hat CodeReady Workspaces provides a default web IDE to use in the developer workspaces. To use another editor, see:

- Section 5.7, “Configuring a workspace to use an IDE based on the IntelliJ Platform”
- Section 5.8, “Theia-based IDEs”

5.7. CONFIGURING A WORKSPACE TO USE AN IDE BASED ON THE INTELLIJ PLATFORM

This section describes how to configure a workspace to use an IDE based on the IntelliJ Platform.

NO INITIAL REPOSITORY CHECKOUT WHEN RUNNING THE CODEREADY WORKSPACES SERVER WORKSPACES ENGINE

When the CodeReady Workspaces instance is running the CodeReady Workspaces Server workspaces engine, the workspace starts without an initial checkout of the code repositories referenced in the devfile.

Workarounds

- In the IDE, click Get from VCS to checkout a repository.
- To enable the automatic initial checkout of the code repositories in the devfile, use the Dev Workspace operator.

5.7.1. Configuring a workspace to use IntelliJ IDEA Community

This section describes how to configure a workspace devfile to use IntelliJ IDEA Community.

Procedure

1. Add the following component to the workspace devfile:

   ```yaml
   components:
   - type: cheEditor
     id: registry.redhat.io/codeready-workspaces/idea-rhel8:2.15
   ```

2. Remove the plugins or commands defined for the Theia IDE from the workspace devfile.

3. Restart the workspace.

5.7.2. Configuring a workspace to use PyCharm Community
This section describes how to configure a workspace devfile to use PyCharm Community.

Procedure

1. Add the following component to the workspace devfile:

```yaml
components:
  - type: cheEditor
```

2. Remove the plugins or commands defined for the Theia IDE from the workspace devfile.

3. Restart the workspace.

5.7.3. Configuring a workspace to use a custom image with an IDE based on the IntelliJ Platform

This section describes how to configure a workspace to use an IDE based on the IntelliJ Platform.

Prerequisites

- CodeReady Workspaces has access to metadata and image with the desired IDE based on the IntelliJ Platform. See Section 5.7.4, “Building images for IDEs based on the IntelliJ Platform”.

Procedure

1. Add the following component to the workspace devfile:

```yaml
components:
  - type: cheEditor
    reference: "<URL_to_meta.yaml>"
```

<URL_to_meta.yaml>: HTTPS resource defining the IDE metadata, see Section 5.7.4, “Building images for IDEs based on the IntelliJ Platform”.

2. Remove the plugins or commands defined for the Theia IDE from the workspace devfile.

3. Restart the workspace.

5.7.4. Building images for IDEs based on the IntelliJ Platform

This section describes how to build images for IDEs based on the IntelliJ Platform version 2020.3.

5.7.4.1. Building an image for IntelliJ IDEA Community or PyCharm Community

This procedure describes how to build an image for IntelliJ IDEA Community or PyCharm Community.

Prerequisites

- The build host has at least 2 GB of available RAM.
- The following tools are installed on the build host:
- **Docker** version 18.09 or greater, supporting **BuildKit**
- **Git**
- **GNU getopt**
- **GNU wget**
- **Java Development Kit (JDK) version 11**
- **jq**

**Procedure**

1. Get a local copy of the **JetBrains Projector Editor Images repository**.

   ```
   $ git clone https://github.com/che-incubator/jetbrains-editor-images
   $ cd jetbrains-editor-images
   ```

2. Run the build script and select the IDE package and package version:

   ```
   $ ./projector.sh build
   [info] Select the IDE package to build (default is ‘IntelliJ IDEA Community’):
   1) IntelliJ IDEA Community
   2) PyCharm Community
   [info] Select the IDE package version to build (default is ‘2020.3.3’):
   1) 2020.3.3
   2) 2020.3.2
   3) 2020.3.1
   ```

3. To test the image, run it locally and go to **http://localhost:8887** to access the IDE.

   ```
   $ ./projector.sh run
   ```

4. Publish the image to a registry accessible by CodeReady Workspaces, and remember the location: `<registry>/<image>:<tag>`.

5. Create a **meta.yaml** file with the following content:

   ```yaml
   apiVersion: v2
   publisher: <publisher> 1
   name: intellij-ide
   version: latest
   type: Che Editor
displayName: IntelliJ Platform IDE
title: IDE based on the IntelliJ Platform
description: IDE based on the IntelliJ Platform running using Projector
icon: https://www.jetbrains.com/apple-touch-icon.png
category: Editor
repository: https://github.com/che-incubator/jetbrains-editor-images
firstPublicationDate: "2021-04-10"
spec:
  endpoints:
  - name: intellij
    public: true
   ```
6. Publish the `meta.yaml` file to an HTTPS resource accessible by CodeReady Workspaces and copy the resulting URL for use as `<URL_to_meta.yaml>` when configuring a workspace to use this IDE.

Next steps

- Section 5.7, “Configuring a workspace to use an IDE based on the IntelliJ Platform”

5.7.4.2. Building an image for an IDE based on the IntelliJ Platform

This procedure describes how to build an image for an IDE based on the IntelliJ Platform version 2020.3. For JetBrains IDEs, the IDE version number corresponds to the version of the IntelliJ Platform. See the list of compatible IDEs.

Prerequisites

- The build host has at least 2 GB of available RAM.
- The following tools are installed on the build host:
  - Docker version 18.09 or greater, supporting BuildKit
  - Git
  - GNU getopt
  - GNU wget
  - Java Development Kit (JDK) version 11
  - jq

Procedure

```yaml
targetPort: 8887
attributes:
  protocol: http
type: ide
path: /projector-client/index.html?backgroundColor=434343&wss
containers:
  - name: intellij-ide
    image: <registry>/<image>:<tag>  2
    mountSources: true
    volumes:
      - mountPath: "/home/projector-user"
        name: projector-user
    ports:
      - exposedPort: 8887
    memoryLimit: "4096M"
```
1. Get a local copy of the JetBrains Projector Editor Images repository.

   $ git clone https://github.com/che-incubator/jetbrains-editor-images
   $ cd jetbrains-editor-images

2. Run the build script with parameters:

   $ ./projector build --tag <tag> --url <URL>

   --tag <tag>
   The name and tag to apply to the image after build in name:tag format.

   --url <url>
   The URL pointing to an archive of the IDE based on the IntelliJ Platform version 2020.3. The archive must target the Linux platform, be in tar.gz format, and include JetBrains Runtime (JBR).

   Example 5.1. Building the image with IntelliJ IDEA Community 2020.3.3
   $ ./projector.sh build --tag che-idea:2020.3.3
     --url https://download-cdn.jetbrains.com/idea/ideaIC-2020.3.3.tar.gz

   Example 5.2. Building the image with PyCharm Community 2020.3.5
   $ ./projector.sh build --tag che-pycharm:2020.3.5
     --url https://download.jetbrains.com/python/pycharm-community-2020.3.5.tar.gz

   Example 5.3. Building the image with WebStorm 2020.3.3
   $ ./projector.sh build --tag che-webstorm:2020.3.3
     --url https://download.jetbrains.com/webstorm/WebStorm-2020.3.3.tar.gz

   Example 5.4. Building the image with IntelliJ IDEA Ultimate 2020.2.2
   $ ./projector.sh build --tag che-idea-ultimate:2020.2.2
     --url https://download.jetbrains.com/idea/ideaIU-2020.2.2.tar.gz

   Example 5.5. Building the image with Android Studio 4.2.0.22
   $ ./projector.sh build --tag che-android-studio:4.2.0.22

3. To test the image, run it locally and go to http://localhost:8887 to access the IDE.
$ ./projector.sh run <tag>

Example 5.6. Testing the image with IntelliJ IDEA Community 2020.3.3

$ ./projector.sh run che-idea:2020.3.3

Example 5.7. Testing the image with PyCharm 2020.3.5

$ ./projector.sh run che-pycharm:2020.3.5

Example 5.8. Testing the image with WebStorm 2020.3.3

$ ./projector.sh run che-webstorm:2020.3.3

Example 5.9. Testing the image with IntelliJ IDEA Ultimate 2020.2.2

$ ./projector.sh run che-idea-ultimate:2020.2.2

Example 5.10. Testing the image with Android Studio 4.2.0.22

$ ./projector.sh run che-android-studio:4.2.0.22

4. Publish the image to a registry accessible by CodeReady Workspaces, and remember the location: <registry>/<image>:<tag>.

5. Create a `meta.yaml` file containing the IDE metadata for CodeReady Workspaces:

```yaml
apiVersion: v2
publisher: <publisher> 1
name: intellij-ide
version: latest
type: Che Editor
displayName: IntelliJ Platform IDE
title: IDE based on the IntelliJ Platform
description: IDE based on the IntelliJ Platform running using Projector
icon: https://www.jetbrains.com/apple-touch-icon.png
category: Editor
repository: https://github.com/che-incubator/jetbrains-editor-images
firstPublicationDate: "2021-04-10"
spec:
  endpoints:
  - name: intellij
    public: true
targetPort: 8887
attributes:
```
Publish the `meta.yaml` file to an HTTPS resource accessible by CodeReady Workspaces and copy the resulting URL for use as `<URL_to_meta.yaml>` when configuring a workspace to use this IDE.

Next steps

- Section 5.7, “Configuring a workspace to use an IDE based on the IntelliJ Platform”

5.7.5. Provisioning the JetBrains offline activation code

Some editions of JetBrains IDEs require a paid subscription beyond the evaluation period, which means buying a license from JetBrains. To register a license, you need to provision to CodeReady Workspaces the JetBrains activation code for offline usage. When you renew your subscription, you will need to generate and provision a new offline activation code.

Prerequisites

- An active JetBrains subscription associated to an active JetBrains account.
- The OpenSSL and oc tools are installed.
- An image containing the IDE. See Section 5.7.4, “Building images for IDEs based on the IntelliJ Platform”.
- A workspace running with the IDE. See Section 5.7, “Configuring a workspace to use an IDE based on the IntelliJ Platform”.

Procedure

1. Log in to your JetBrains account, choose the desired subscription, and click on the Download activation code for offline usage link.

2. Extract from the downloaded zip archive the file named `<License ID> - for 2018.1 or later.txt`.

```yaml
protocol: http
type: ide
path: /projector-client/index.html?backgroundColor=434343&wss
containers:
  - name: intellij-ide
    image: <registry>/<image>:<tag>
    mountSources: true
    volumes:
      - mountPath: "/home/projector-user"
        name: projector-user
    ports:
      - exposedPort: 8887
    memoryLimit: "4096M"
```

1 `<publisher>`: Your publisher name.
2 `<registry>/<image>:<tag>`: Location of the IDE image in a registry accessible by CodeReady Workspaces.
3. Convert the activation code to a base64 encoded single line for use in the next step as `<base64_encoded_activation_code>`.

```bash
$ openssl base64 -e -A -in '&License ID> - for 2018.1 or later.txt'
```

4. Create a `secret.yaml` file defining the OpenShift Secret to provision the activation code to CodeReady Workspaces.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: jetbrains-offline-activation-code
  labels:
    app.kubernetes.io/component: workspace-secret
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/automount-workspace-secret: 'false'
    che.eclipse.org/mount-path: /tmp/
    che.eclipse.org/mount-as: file
data:
  idea.key: <base64_encoded_activation_code>
  pycharm.key: <base64_encoded_activation_code>
  webstorm.key: <base64_encoded_activation_code>
  phpstorm.key: <base64_encoded_activation_code>
  goland.key: <base64_encoded_activation_code>

1 che.eclipse.org/automount-workspace-secret: 'false': disables the mounting process until a workspace component explicitly requests it with the automountWorkspaceSecrets: true property.

5. Apply the OpenShift Secret to the OpenShift project running the workspace.

```bash
$ oc apply -f secret.yaml
```

6. To mount the activation codes into a workspace, update the workspace devfile configuration to set automountWorkspaceSecrets: true.

```yaml
components:
  - type: cheEditor
    automountWorkspaceSecrets: true
    reference: "<URL_to_meta.yaml>"
```

7. Restart the workspace.

### 5.8. THEIA-BASED IDES

This section describes how to provide a custom IDE, based on Eclipse Theia framework.

To use a Theia-based IDE in Red Hat CodeReady Workspaces as an editor, you need to prepare two main components:

- a Docker image containing your IDE
• the Che editor descriptor file - **meta.yaml**

Procedure

1. Describe the IDE with an editor descriptor - **meta.yaml** file:

   ```yaml
   version: 1.0.0
   editors:
     - id: eclipse/theia/next
       title: Eclipse Theia development version.
       displayName: theia-ide
       description: Eclipse Theia, get the latest release each day.
       repository: https://github.com/eclipse-che/che-theia
       firstPublicationDate: "2021-01-01"
   endpoints:
     - name: "theia"
       public: true
       targetPort: 3100
       attributes:
         protocol: http
         type: ide
         secure: true
         cookiesAuthEnabled: true
         discoverable: false
   containers:
     - name: theia-ide
       image: "<your-ide-image>"
       mountSources: true
       ports:
         - exposedPort: 3100
       memoryLimit: "512M"
   ``

   **targetPort** and **exposedPort** must be the same as the Theia-based IDE running inside the container. Replace `<your-ide-image>` with the name of the IDE image. The **meta.yaml** file should be publicly accessible through an HTTP(S) link.

2. Add your editor to a Devfile:

   ```yaml
   apiVersion: 1.0.0
   metadata:
     name: che-theia-based-ide
   components:
     - type: cheEditor
       reference: '<meta.yaml URL>'
   ``

   `<meta.yaml URL>` should point to the publicly hosted **meta.yaml** file described in the previous step.

### 5.9. Adding Tools to CodeReady Workspaces After Creating a Workspace
When installed in a workspace, CodeReady Workspaces plug-ins bring new capabilities to 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.
- 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 installing, enabling, and using CodeReady Workspaces plug-ins in workspaces.

- Section 5.9.1, “Additional tools in the CodeReady Workspaces workspace”
- Section 5.9.2, “Adding a language support plug-in to a CodeReady Workspaces workspace”

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

CodeReady Workspaces plug-ins are extensions to the Che-Theia IDE that come bundled with container images. These images contain the native prerequisites of their respective extensions. For example, the OpenShift command-line tool is bundled with a command to install it, which ensures the proper functionality of the OpenShift Connector plug-in, all available in the dedicated image.

Plug-ins can also include metadata to define a description, categorization tags, and an icon. CodeReady Workspaces provides a registry of plug-ins available for installation into the user’s workspace.

The Che-Theia IDE is generally compatible with the Visual Studio Code extensions API and Visual Studio Code extensions are automatically compatible with Che-Theia. These extensions are possible to package as CodeReady Workspaces plug-ins by combining them with their dependencies. By default, CodeReady Workspaces includes a plug-in registry containing common plug-ins.

#### Adding a plug-in

- Using the Dashboard:
  - Add a plug-in directly into a devfile using the Devfile tab. The devfile can also further the plug-in configuration, such as defining memory or CPU consumption.

- Using the Che-Theia IDE:
  - By pressing Ctrl-Shift-J or by navigating to View → Plugins.

#### Additional resources

- Adding components to a devfile

### 5.9.2. Adding a language support plug-in to a CodeReady Workspaces workspace

This procedure describes adding a tool to a created workspace by enabling a dedicated plug-in from the Dashboard.
• Edit the workspace devfile from the Dashboard Devfile tab.

Prerequisites

• A running instance of CodeReady Workspaces.
  See Installing CodeReady Workspaces.

• A created workspace that is defined in this instance of Red Hat CodeReady Workspaces.
  See ] and xref:creating-a-workspace-from-a-code-sample_crw[.

• The workspace must be in a stopped state.
  The steps to stop a workspace:
  a. Navigate to the CodeReady Workspaces Dashboard, as explained in Section 1.1, “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 Stop to stop the workspace.
  d. Wait a few seconds for the workspace to stop, and then configure the workspace by selecting it.

Procedure

To add a plug-in from the plug-in registry to a created CodeReady Workspaces workspace, install the plug-in as follows by adding content to the devfile:

1. Navigate to the Devfile tab, where the devfile YAML is displayed.

2. In the components devfile section, add the following lines: id and type.

   Example: Adding the Java 8 language plug-in

   ```yaml
   - id: redhat/java8/latest
     type: chePlugin
   ```

   Example: The end result

   ```yaml
   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
   ```
3. Click **Save** to save the changes.

4. Restart the workspace.

5. Verify that the workspace includes the new plug-in.

Additional resources

- Devfile specifications

### 5.10. USING PRIVATE CONTAINER REGISTRIES

This section describes the necessary steps to use container images from private container registries.

**Prerequisites**


**Procedure**

1. Navigate to the CodeReady Workspaces Dashboard. See Section 1.1, “Navigating CodeReady Workspaces using the Dashboard”.

2. Navigate to **User Preferences**.
   
   a. Click on your username in the top right corner.
   
   b. Click the **User Preferences** tab.

3. Click the **Add Container Registry** button in **Container Registries** tab and execute following actions:
   
   - Enter the container registry domain name in the **Registry** field.
   
   - Optionally, enter the username of your account at this registry in the **Username** field.
   
   - Enter the password in the **Password** field to authenticate in the container registry.

4. Click the **Add** button.

**Verification**

1. See that there is a new entry in the **Container Registries** tab.

2. Create a workspace that uses a container image from the specified container registry. See Section 4.2, “Authoring a devfile 2”.

```
containerPath: (prod-home)/.composer
- name: symfony
containerPath: (prod-home)/.symfony
alias: php
image: 'quay.io/eclipse/che-php-7:nightly'
- id: redhat/java8/latest
type: chePlugin
```
Additional resources

- Kubernetes documentation: Pull an Image from a Private Registry
CHAPTER 6. USING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT

By configuring technology stacks, you can work with artifacts from in-house repositories using self-signed certificates.

- Section 6.1, “Using Maven artifact repositories”
- Section 6.2, “Using Gradle artifact repositories”
- Section 6.3, “Using Python artifact repositories”
- Section 6.4, “Using Go artifact repositories”
- Section 6.5, “Using NuGet artifact repositories”
- Section 6.6, “Using npm 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. Ensure that settings.xml is in all the containers that use Maven tools. In particular, ensure that it is in 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 are about to share `<home dir>/.m2` folder with this container, you have to specify the custom volume for this specific component in the devfile:

```yaml
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:

```xml
<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>
      <repositories>
        <repository>
          <id>my-nexus-snapshots</id>
          <releases>
            <enabled>false</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-snapshots/</url>
        </repository>
        <repository>
          <id>my-nexus-releases</id>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-releases/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```
6.1.2. Defining Maven settings.xml file across workspaces

To use your own settings.xml file across all your workspaces, create a Secret object (with a name of your choice) in the same project as the workspace. Put the contents of the required settings.xml in the data section of the Secret (possibly along with other files that should reside in the same directory). Labelling and annotating this Secret according to Section 3.10.1, “Mounting a secret as a file into a workspace container” ensures that the contents of the Secret is mounted into the workspace Pod. Note that you need to restart any previously running workspaces for them to use this Secret.

Prerequisites

This is required to set your private credentials to a Maven repository. See the Maven documentation Settings.xml#Servers for additional information.

To mount this settings.xml:

```xml
<settings
xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">

<activeProfile>my-nexus</activeProfile>
</activeProfiles>

<settings
xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">

<servers>
    <server>
        <id>repository-id</id>
        <username>username</username>
        <password>password123</password>
    </server>
</servers>
</settings>
```

Procedure

1. Convert settings.xml to base64:

   ```bash
   $ cat settings.xml | base64
   ```

2. Copy the output to a new file, secret.yaml, which also defines needed annotations and labels:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: maven-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/jboss/.m2
       che.eclipse.org/mount-as: file
   type: Opaque
   data:
   ```
3. Create this secret in the cluster:

$ oc apply -f secret.yaml

4. Start a new workspace. The maven container contains a file `/home/jboss/.m2/settings.xml` with your original content.

### 6.1.2.1. OpenShift 3.11 and OpenShift <1.13

On OpenShift 3.11, it’s impossible to have multiple VolumeMounts at same path so having devfile with volume `/home/jboss/.m2` and secret at `/home/jboss/.m2/settings.xml` would resolve into the conflict. On these clusters use `/home/jboss/.m2/repository` as a volume for maven repository in the devfile:

apiVersion: 1.0.0
metadata: ...
components:
- type: dockerimage
  alias: maven
  image: maven:3.11
  volumes:
  - name: m2
    containerPath: /home/jboss/.m2/repository

### 6.1.3. Using self-signed certificates in Maven projects

Internal artifact repositories often do not have a certificate signed by an authority that is trusted by default in Java. They are mainly 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 customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see https://access.redhat.com/documentation/en-us/red_hat_coderedey_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

   a. Copy the original Java truststore file:
b. Import the certificate into the Java truststore file

```
$ keytool -import -noprompt -file /public-certs/nexus.cer -alias nexus -keystore
/projects/maven/truststore.jks -storepass changeit
Certificate was added to keystore
```

2. Add the truststore file.

- In the Maven container:
  a. Add the `javax.net.ssl` system property to the `MAVEN_OPTS` environment variable:

```
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/maven/truststore.jks
  -Djavax.net.ssl.trustStorePassword=changeit
```

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/maven/truststore.jks
  -Djavax.net.ssl.trustStorePassword=changeit
```

6.2. USING GRADLE ARTIFACT REPOSITORIES

This section describes how to download and configure Gradle.

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 available 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 archive file locally in `/project/gradle` in your workspace.

- Change the `distributionUrl` property to point to a local address of the Gradle distribution archive 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:

```gradle
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 Gradle projects
Internal artifact repositories often do not have a certificate signed by an authority that is trusted by default in Java. They are mainly 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 customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index# importing-untrusted-tls-certificates.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index# importing-untrusted-tls-certificates.adoc)). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

   a. Copy the original Java truststore file:

   ```
   $ mkdir /projects/maven
   $ cp $JAVA_HOME/lib/security/cacerts /projects/maven/truststore.jks
   $ chmod +w /projects/maven/truststore.jks
   ```

   b. Import the certificate into the Java truststore file

   ```
   Certificate was added to keystore
   ```

   c. 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/maven/truststore.jks
         -Djavax.net.ssl.trustStorePassword=changeit
   ```

**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.
6.3.2. Using self-signed certificates in Python projects

Internal artifact repositories often do not have a self-signed TLS certificate signed by an authority that is trusted by default. They are mainly 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 used by the pip server in the Privacy-Enhanced Mail (PEM) format. It is customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

   **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'
     - mountSources: true
     memoryLimit: 512Mi
     type: dockerimage
     alias: python
     image: 'registry.redhat.io/codeready-workspaces/plugin-java8-rhel8:2.5'
     env:
       - name: 'PIP_INDEX_URL'
         value: 'https://<username>:<password>@pypi.company.com/simple'

     - name: 'PIP_INDEX_URL'
       value: '/projects/tls/rootCA.pem'
     name: 'PIP_CERT'
     - mountSources: true
     memoryLimit: 512Mi
6.4. USING GO ARTIFACT REPOSITORIES

To configure Go in a restricted environment, use the **GOPROXY** environment variable and the Athens module data store and proxy.

6.4.1. Configuring Go to use a non-standard-registry

Athens is a Go module data store and proxy with many configuration options. It can be configured to act only as a module data store and not as a proxy. An administrator can upload their Go modules to the Athens data store and have them available across their Go projects. If a project tries to access a Go module that is not in the Athens data store, the Go build fails.

- To work with Athens, configure the **GOPROXY** environment variable in the devfile of your CLI container:

  ```yaml
  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 TLS certificate signed by an authority that is trusted by default. They are typically 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. It is customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

2. Add the appropriate environment variables to your devfile:
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 file with a Nexus repository hosted at nexus.example.org:**

```xml
<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

**Procedure**

- Modify the `nuget.config` file and use the `SSL_CERT_FILE` environment variable in the devfile to add self-signed certificates.
Internal artifact repositories often do not have a self-signed TLS certificate signed by an authority that is trusted by default. They are mainly signed by an internal company authority or are self-signed. Configure your tools to accept these certificates.

**Procedure**

1. Obtain the certificate used by the .NET server in the Privacy-Enhanced Mail (PEM) format. It is customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

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:**

```yaml
components:
- id: redhat-developer/che-omnisharp-plugin/latest
  memoryLimit: 1024Mi
  type: chePlugin
  alias: omnisharp
  env:
    - value: /public-certs/nuget.cer
      name: SSL_CERT_FILE
    - mountSources: true
      endpoints:
        - name: 5000/tcp
          port: 5000
          memoryLimit: 512Mi
          type: dockerimage
          volumes:
            - name: dotnet
              containerPath: /home/jboss
              alias: dotnet
              image: 'quay.io/eclipse/che-dotnet-2.2:7.7.1'
              env:
                - value: /projects/tls/rootCA.crt
                  name: SSL_CERT_FILE
```

**6.6. USING NPM ARTIFACT REPOSITORIES**

The npm (Node Package Manager) package manager for the JavaScript programming language is configured using the `npm config` command, by writing values to the `.npmrc` files. However, configuration values can also be set using the environment variables beginning with `NPM_CONFIG_`.

The 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`
Obtain a server certificate file from the repository server. It is customary for administrators to provide certificates of internal artifact servers as OpenShift secrets (see https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc). The relevant server certificates will be mounted in `/public-certs` in every container in the workspace.

1. An example configuration for the use of an internal repository with a self-signed certificate:

```yaml
- 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: '/public-certs/nexus.cer'
    - name: NPM_CONFIG_REGISTRY
      value: 'https://snexus-airgap.apps.acme.com/repository/npm-proxy/
```
CHAPTER 7. TROUBLESHOOTING CODEREADY WORKSPACES

This section provides troubleshooting procedures for the most frequent issues a user can come in conflict with.

Additional resources

- Section 7.1, “Viewing CodeReady Workspaces workspaces logs”
- Section 7.2, “Investigating failures at a workspace start using the Verbose mode”
- Section 7.3, “Troubleshooting slow workspaces”
- Section 7.4, “Troubleshooting network problems”

7.1. VIEWING CODEREADY WORKSPACES WORKSPACES LOGS

This section describes how to view CodeReady Workspaces workspaces logs.

7.1.1. Viewing logs from language servers and debug adapters

7.1.1.1. Checking important logs

This section describes how to check important logs.

Procedure

1. In the OpenShift web console, click Applications → Pods to see a list of all the active workspaces.
2. Click on the name of the running Pod where the workspace is running. The Pod screen contains the list of all containers with additional information.
3. Choose a container and click the container name.
   
   **NOTE**
   
   The most important logs are the theia-ide container and the plug-ins container logs.
4. On the container screen, navigate to the Logs section.

7.1.1.2. Detecting memory problems

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

The plug-in container runs out of memory

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

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

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

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

### 7.1.1.3. Logging the client-server traffic for debug adapters

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

**Prerequisites**

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

**Procedure**

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

### 7.1.1.4. Viewing logs for Python

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

**Procedure**

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

![Output View](image)

### 7.1.1.5. Viewing logs for Go

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

#### 7.1.1.5.1. Finding the Go path

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

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

```
> Go: Current GOPATH
Go: Current GOPATH

/go:/projects is the current GOPATH.
```

7.1.1.5.2. Viewing the Debug Console log for Go

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

Procedure

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

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

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

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

3. The debug console prints the additional information in the debug console.
7.1.1.5.3. Viewing the Go logs output in the Output panel

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

**Procedure**

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

7.1.1.6. Viewing logs for the NodeDebug NodeDebug2 adapter

**NOTE**

No specific diagnostics exist other than the general ones.

7.1.1.7. Viewing logs for Typescript

7.1.1.7.1. Enabling the label switched protocol (LSP) tracing

**Procedure**

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

7.1.1.7.2. Viewing the Typescript language server log

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

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

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

### 7.1.1.7.3. Viewing the Typescript logs output in the Output panel

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

**Procedure**

1. Navigate to the **Output** view

2. Select **TypeScript** in the drop-down list.

### 7.1.1.8. Viewing logs for Java

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

#### 7.1.1.8.1. Verifying the state of the Eclipse JDT Language Server

**Procedure**

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

1. Open a terminal in the container that is running the Eclipse JDT Language Server plug-in (an example name for the container: **vscode-javaxxx**).

2. Inside the terminal, run the **ps aux | grep jdt** command to check if the Eclipse JDT Language Server process is running in the container. If the process is running, the output is:

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

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

3. Check all logs described in **Checking important logs**

#### 7.1.1.8.2. Verifying the Eclipse JDT Language Server features
Procedure

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

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

7.1.1.8.3. Viewing the Java language server log

Procedure

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

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

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

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

Procedure

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

Additional resources

For troubleshooting instructions, see the Visual Studio Code Java GitHub repository.

7.1.1.9. Viewing logs for Intelephense

7.1.1.9.1. Logging the Intelephense client-server communication

Procedure

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

1. Click `File → Settings`.

2. Open the **Preferences** view.

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

7.1.1.9.2. Viewing Intelephense events in the Output panel

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

Procedure
1. Click **View → Output**

2. Select **Intelephense** in the drop-down list for the **Output** view.

### 7.1.1.10. Viewing logs for PHP-Debug

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

**Procedure**

1. In the **launch.json** file, add the **"log": true** attribute to the **php** configuration.

2. Start the debug session.

3. The diagnostic messages are printed into the **Debug Console** view along with the application output.

### 7.1.11. Viewing logs for XML

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

#### 7.1.1.11.1. Verifying the state of the XML language server

**Procedure**

1. Open a terminal in the container named **vscode-xml-<xxx>**.

2. Run **ps aux | grep java** to verify that the XML language server has started. If the process is running, the output is:

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

   If it is not, see the **Checking important logs** chapter.

#### 7.1.1.11.2. Checking XML language server feature flags

**Procedure**

1. Check if the features are enabled. The XML plug-in provides multiple settings that can enable and disable features:

   - **xml.format.enabled**: Enable the formatter
   - **xml.validation.enabled**: Enable the validation
   - **xml.documentSymbols.enabled**: Enable the document symbols

2. To diagnose whether the XML language server is working, create a simple XML element, such as `<hello></hello>`, and confirm that it appears in the **Outline** panel on the right.

3. If the document symbols do not show, ensure that the **xml.documentSymbols.enabled** attribute is set to **true**. If it is **true**, and there are no symbols, the language server may not be hooked to the editor. If there are document symbols, then the language server is connected to...
4. Ensure that the features that the user needs, are set to true in the settings (they are set to true by default). If any of the features are not working, or not working as expected, file an issue against the Language Server.

### 7.1.11.3. Enabling XML Language Server Protocol (LSP) tracing

**Procedure**

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

### 7.1.11.4. Viewing the XML language server log

**Procedure**

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

### 7.1.12. Viewing logs for YAML

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

#### 7.1.12.1. Verifying the state of the YAML language server

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

**Procedure**

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

1. In the editor, open a terminal in the container that is running the YAML plug-in (an example name of the container: vscode-yaml-<xxx>).

2. In the terminal, run the ps aux | grep node command. This command searches all the node processes running in the current container.

3. Verify that a command node **/server.js is running.**
7.1.12.2. Checking the YAML language server feature flags

Procedure

To check the feature flags:

1. Check if the features are enabled. The YAML plug-in provides multiple settings that can enable and disable features, such as:

   - `yaml.format.enable`: Enables the formatter
   - `yaml.validate`: Enables validation
   - `yaml.hover`: Enables the hover function
   - `yaml.completion`: Enables the completion function

2. To check if the plug-in is working, type the simplest YAML, such as `hello: world`, and then open the Outline panel on the right side of the editor.

3. Verify if there are any document symbols. If yes, the language server is connected to the editor.

4. If any feature is not working, verify that the settings listed above are set to `true` (they are set to `true` by default). If a feature is not working, file an issue against the Language Server.

7.1.12.3. Enabling YAML Language Server Protocol (LSP) tracing

Procedure

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

The `node **/server.js` running in the container indicates that the language server is running. If it is not running, the language server has not started inside the container. In this case, see Checking important logs.
7.1.1.13. Viewing logs for .NET with OmniSharp-Theia plug-in

7.1.1.13.1. OmniSharp-Theia plug-in

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

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

The language server runs on .NET SDK 2.2.105.

7.1.1.13.2. Verifying the state of the OmniSharp-Theia plug-in language server

Procedure

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

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

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

7.1.1.13.3. Checking OmniSharp Che-Theia plug-in language server features

Procedure

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

7.1.1.13.4. Viewing OmniSharp-Theia plug-in logs in the Output panel

Procedure

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

7.1.1.14. Viewing logs for .NET with NetcoredebugOutput plug-in

7.1.1.14.1. NetcoredebugOutput plug-in

The NetcoredebugOutput plug-in provides the netcoredbg tool. This tool implements the Visual Studio Code Debug Adapter protocol and allows users to debug .NET applications under the .NET Core runtime.

The container where the NetcoredebugOutput plug-in is running contains .NET SDK v.2.2.105.
7.1.14.2. Verifying the state of the NetcoredebugOutput plug-in

Procedure

1. Search for a netcoredbg debug configuration in the launch.json file.
   
   Example 7.1. Sample debug configuration
   
   ```json
   {
      "type": "netcoredbg",
      "request": "launch",
      "program": "${workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>",
      "args": [],
      "name": ".NET Core Launch (console)",
      "stopAtEntry": false,
      "console": "internalConsole"
   }
   ```

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

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

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

Procedure

- Open the Debug console.

7.1.15. Viewing logs for Camel

7.1.15.1. Verifying the state of the Camel language server

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

To verify the state of the language server:

1. Open a terminal inside the `vscode-apache-camel<xxx>` container.

2. Run the `ps aux | grep java` command. The following is an example language server process:

   ```
   ```

3. If you cannot find it, see [Checking important logs](#).

### 7.1.15.2. Viewing Camel logs in the Output panel

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

**Procedure**

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

**NOTE**

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

### 7.1.2. Viewing Che-Theia IDE logs

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

#### 7.1.2.1. Viewing Che-Theia editor logs using the OpenShift CLI

Observing Che-Theia editor logs helps to get a better understanding and insight over the plug-ins loaded by the editor. This section describes how to access the Che-Theia editor logs using the OpenShift CLI (command-line interface).
Prerequisites
- CodeReady Workspaces is deployed in an OpenShift cluster.
- A workspace is created.
- User is located in a CodeReady Workspaces installation project.

Procedure
1. Obtain the list of the available Pods:

   $ oc get pods

   Example

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

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

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

   Example:

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

3. Get logs from the theia/ide container:

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

   Example:

   $ oc logs --follow workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -container theia-idexzb
   >root INFO unzipping the plug-in 'task_plugin.theia' to directory: /tmp/theia-unpacked/task_plugin.theia
   root INFO unzipping the plug-in 'theia_yeoman_plugin.theia' to directory: /tmp/theia-unpacked/theia_yeoman_plugin.theia
   root WARN A handler with prefix term is already registered.
   root INFO [nsfw-watcher: 75] Started watching: /home/theia/.theia
   root WARN e.onStart is slow, took: 367.4600000013015 ms
   root INFO [nsfw-watcher: 75] Started watching: /projects
   root INFO [4f9590c5-e1c5-40d1-b9f8-ec31ec3bdac5] Sync of 9 plugins took: 62.26000000242493 ms
   root INFO [nsfw-watcher: 75] Started watching: /projects
   root INFO [hosted-plugin: 88] PLUGIN_HOST(88) starting instance
7.2. INVESTIGATING FAILURES AT A WORKSPACE START USING THE VERBOSE MODE

Verbose mode allows users to reach an enlarged log output, investigating failures at a workspace start.

In addition to usual log entries, the Verbose mode also lists the container logs of each workspace.

7.2.1. Restarting a CodeReady Workspaces workspace in Verbose mode after start failure

This section describes how to restart a CodeReady Workspaces workspace in the Verbose mode after a failure during the workspace start. Dashboard proposes the restart of a workspace in the Verbose mode once the workspace fails at its start.

Prerequisites

- An existing workspace that fails to start.

Procedure

1. Using Dashboard, try to start a workspace.
2. When it fails to start, click on the displayed Open in Verbose mode link.
3. Check the Logs tab to find a reason for the workspace failure.

7.2.2. Starting a CodeReady Workspaces workspace in Verbose mode

This section describes how to start the Red Hat CodeReady Workspaces workspace in Verbose mode.

Prerequisites

- An existing workspace defined on this instance of CodeReady Workspaces.

Procedure

1. Open the Workspaces tab.
2. On the left side of a row dedicated to the workspace, access the drop-down menu displayed as three horizontal dots and select the Open in Verbose mode option. Alternatively, this option is also available in the workspace details, under the Actions drop-down menu.
3. Check the Logs tab to find a reason for the workspace failure.
7.3. TROUBLESHOOTING SLOW WORKSPACES

Sometimes, workspaces can take a long time to start. Tuning can reduce this start time. Depending on the options, administrators or users can do the tuning.

This section includes several tuning options for starting workspaces faster or improving workspace runtime performance.

7.3.1. Improving workspace start time

Caching images with Image Puller

*Role: Administrator*

When starting a workspace, OpenShift pulls the images from the registry. A workspace can include many containers meaning that OpenShift pulls Pod’s images (one per container). Depending on the size of the image and the bandwidth, it can take a long time.

Image Puller is a tool that can cache images on each of OpenShift nodes. As such, pre-pulling images can improve start times. See [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#caching-images-for-faster-workspace-start.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#caching-images-for-faster-workspace-start.adoc).

Choosing better storage type

*Role: Administrator and user*

Every workspace has a shared volume attached. This volume stores the project files, so that when restarting a workspace, changes are still available. Depending on the storage, attach time can take up to a few minutes, and I/O can be slow.

To avoid this problem, use ephemeral or asynchronous storage. See [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-storage-types.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-storage-types.adoc).

Installing offline

*Role: Administrator*

Components of CodeReady Workspaces are OCI images. Set up Red Hat CodeReady Workspaces in offline mode (air-gap scenario) to reduce any extra download at runtime because everything needs to be available from the beginning. See [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#installing-che-in-a-restricted-environment.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#installing-che-in-a-restricted-environment.adoc).

Optimizing workspace plug-ins

*Role: User*

When selecting various plug-ins, each plug-in can bring its own sidecar container, which is an OCI image. OpenShift pulls the images of these sidecar containers.

Reduce the number of plug-ins, or disable them to see if start time is faster. See also [https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#caching-images-for-faster-workspace-start.adoc](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#caching-images-for-faster-workspace-start.adoc).

Reducing the number of public endpoints

*Role: Administrator*

For each endpoint, OpenShift is creating OpenShift Route objects. Depending on the underlying configuration, this creation can be slow.
To avoid this problem, reduce the exposure. For example, to automatically detect a new port listening inside containers and redirect traffic for the processes using a local IP address (127.0.0.1), the Che-Theia IDE plug-in has three optional routes.

By reducing the number of endpoints and checking endpoints of all plug-ins, workspace start can be faster.

CDN configuration

The IDE editor uses a CDN (Content Delivery Network) to serve content. Check that the content uses a CDN to the client (or a local route for offline setup). To check that, open Developer Tools in the browser and check for vendors in the Network tab. vendors.<random-id>.js or editor.main.* should come from CDN URLs.

7.3.2. Improving workspace runtime performance

Providing enough CPU resources

Plug-ins consume CPU resources. For example, when a plug-in provides IntelliSense features, adding more CPU resources may lead to better performance. Ensure the CPU settings in the devfile definition, devfile.yaml, are correct:

```
apiVersion: 1.0.0

components:
  - type: chePlugin
    id: id/of/plug-in
    cpuLimit: 1360Mi
    cpuRequest: 100m

1. Specifies the CPU limit for the plug-in.
2. Specifies the CPU request for the plug-in.
```

Providing enough memory

Plug-ins consume CPU and memory resources. For example, when a plug-in provides IntelliSense features, collecting data can consume all the memory allocated to the container. Providing more memory to the plug-in can increase performance. Ensure about the correctness of memory settings:

- in the plug-in definition - meta.yaml file
- in the devfile definition - devfile.yaml file

```
apiVersion: v2

spec:
  containers:
    - image: "quay.io/my-image"
      name: "vscode-plugin"
```
Specifies the memory limit for the plug-in.

In the devfile definition (devfile.yaml):

```
memoryLimit: "512Mi"
```

1. Specifies the memory limit for the plug-in.

```
apiVersion: 1.0.0
components:
  - type: chePlugin
    id: id/of/plug-in
    memoryLimit: 1048M
    memoryRequest: 256M
```

1. Specifies the memory limit for this plug-in.

Choosing better storage type

Use ephemeral or asynchronous storage for faster I/O. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/installation_guide/index#configuring-storage-types.adoc.

7.4. TROUBLESHOOTING NETWORK PROBLEMS

This section describes how to prevent or resolve issues related to network policies. CodeReady Workspaces requires the availability of the WebSocket Secure (WSS) connections. Secure WebSocket connections improve confidentiality and also reliability because they reduce the risk of interference by bad proxies.

Prerequisites

- The WebSocket Secure (WSS) connections on port 443 must be available on the network. Firewall and proxy may need additional configuration.
- Use a supported web browser:
  - Google Chrome
  - Mozilla Firefox

Procedure

1. Verify the browser supports the WebSocket protocol. See: Searching a websocket test.
2. Verify firewalls settings: WebSocket Secure (WSS) connections on port 443 must be available.
3. Verify proxy servers settings: The proxy transmits and intercepts WebSocket Secure (WSS) connections on port 443.
CHAPTER 8. OPENSShift 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. Rather than 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.

- Section 8.1, “Features of OpenShift Connector”
- Section 8.2, “Installing OpenShift Connector in CodeReady Workspaces”
- Section 8.3, “Authenticating with OpenShift Connector from CodeReady Workspaces when the OpenShift OAuth service does not authenticate the CodeReady Workspaces instance”
- Section 8.4, “Creating Components with OpenShift Connector in CodeReady Workspaces”
- Section 8.5, “Connecting source code from GitHub to an OpenShift Component using OpenShift Connector”

8.1. FEATURES OF OPENSShift 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:
  - Authentication tokens
  - Username and password
  - Auto-login feature when CodeReady Workspaces is authenticated with the OpenShift OAuth service
- 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.
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 OPENSHIFT 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


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 OPENSOURCE CONNECTOR FROM CODEREADY WORKSPACES WHEN THE OPENSOURCE OAUTH SERVICE DOES NOT AUTHENTICATE THE CODEREADY WORKSPACES INSTANCE

This section describes how to authenticate with an OpenShift cluster when the OpenShift OAuth service does not authenticate the CodeReady Workspaces instance. It enables the user to develop and push Components from CodeReady Workspaces to the OpenShift instance that contains CodeReady Workspaces.

NOTE

When the OpenShift OAuth service authenticates the CodeReady Workspaces instance, the OpenShift Connector plug-in automatically establishes the authentication with the OpenShift instance containing CodeReady Workspaces.

OpenShift Connector offers the following methods for logging in to the OpenShift Cluster from the CodeReady Workspaces instance:

- Using the notification asking to log in to the OpenShift instance containing CodeReady Workspaces.
- Using the Log in to the cluster button.
- Using the Command Palette.

NOTE

OpenShift Connector plug-in requires manual connecting to the target cluster.

The OpenShift Connector plug-in logs in to the cluster as inClusterUser. If this user does not have manage project permission, this error message appears when creating a project 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 issue:

1. Log out from the local cluster.
2. Log in to OpenShift cluster using the OpenShift user’s credentials.

Prerequisites

- A running instance of CodeReady Workspaces. See https://access.redhat.com/documentation/en-
A CodeReady Workspaces workspace is available. See Chapter 3, *Developer workspaces*.

The OpenShift Connector plug-in is available. See Section 8.2, “Installing OpenShift Connector in CodeReady Workspaces”.

The OpenShift OAuth provider is available only for the auto-login to the OpenShift instance containing CodeReady Workspaces. See https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.15/html-single/administration_guide/index#configuring-openshift-oauth.adoc.

### Procedure

1. In the left panel, click the OpenShift Application Explorer icon.

2. In the OpenShift Connector panel, 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.

4. Select the method to log in to the cluster: Credentials or Token, and follow the login instructions.

   **NOTE**
   
   To authenticate with a token, the required token information is in the upper right corner of the main OpenShift Container Platform screen, under `<User name>` > Copy Login Command.

### 8.4. CREATING COMPONENTS WITH OPENSHIFT 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 deployable assets 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

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 deployed to the cluster. Use a right-click for selecting additional actions, such as debugging and opening in a browser, which requires the exposure of the port 8080.

8.5. CONNECTING SOURCE CODE FROM GITHUB TO AN OPENShift COMPONENT USING OPENSFit 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 using the Add to workspace button.

For additional information about cloning Git repository, see: Section 2.2.2, “Accessing a Git repository using HTTPS”.
CHAPTER 9. TELEMETRY OVERVIEW

Telemetry is the explicit and ethical collection of operation data. By default, telemetry is not available in Red Hat CodeReady Workspaces, but there is an abstract API that allows enabling telemetry using the plug-in mechanism. This approach is used in the Eclipse Che hosted by Red Hat service where telemetry is enabled for every workspace.

This documentation includes a guide describing how to make your own telemetry client for Red Hat CodeReady Workspaces, followed by an overview of the Red Hat CodeReady Workspaces Woopra Telemetry Plugin.

9.1. USE CASES

Red Hat CodeReady Workspaces telemetry API allows tracking:

- Duration of a workspace utilization
- User-driven actions such as file editing, committing, and pushing to remote repositories.
- The list of plug-ins enabled in a workspace
- Programming languages and devfiles used in workspaces. See Section 4.2, "Authoring a devfile 2"

9.2. HOW IT WORKS

When a CodeReady Workspaces workspace starts, the che-theia container starts the telemetry plug-in, which is responsible for sending telemetry events to a back-end. If the $CHE_WORKSPACE_TELEMETRY_BACKEND_PORT environment variable was set in the workspace Pod, the telemetry plug-in will send events to a back-end listening at that port.

If the CodeReady Workspaces workspace has a telemetry back-end container running, and it is listening on $CHE_WORKSPACE_TELEMETRY_BACKEND_PORT, it takes the events sent from the telemetry plug-in, turns them into the back-end-specific representation of events, and sends them to the configured analytics back-end (for example, Segment or Woopra).
9.3. CREATING A TELEMETRY PLUG-IN

This section shows how to create an AnalyticsManager class that extends AbstractAnalyticsManager and implements the following methods:

- **isEnabled()** - determines whether the telemetry back-end is functioning correctly. This could mean always returning `true`, or have more complex checks, for example, returning `false` when a connection property is missing.

- **destroy()** - cleanup method that is run before shutting down the telemetry back-end. This method sends the WORKSPACE_STOPPED event.
onActivity() - notifies that some activity is still happening for a given user. This is mainly used to send WORKSPACE_INACTIVE events.

onEvent() - submits telemetry events to the telemetry server, such as WORKSPACEUSED or WORKSPACE_STARTED.

increaseDuration() - increases the duration of a current event rather than sending many events in a small frame of time.

The following sections cover:

- Creation of a telemetry server to echo events to standard output.
- Extending the CodeReady Workspaces telemetry client and implementing a user’s custom back-end.
- Creating a meta.yaml file representing a CodeReady Workspaces workspace plug-in for a user’s custom back-end.
- Specifying of a location of a custom plug-in to CodeReady Workspaces by setting the CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS environment variable.

9.3.1. Getting Started

This document describes the steps required to extend the CodeReady Workspaces telemetry system to connect to a custom back-end:

1. Creating a server process that receives events
2. Extending CodeReady Workspaces libraries to create a back-end that send events to the server
3. Packaging the telemetry back-end in a container and deploying it to an image registry
4. Adding a plug-in for your back-end and instructing CodeReady Workspaces to load the plug-in in your workspaces

Optional: creating a server that receives events
This example shows how to create a server that receives events from CodeReady Workspaces and writes them to standard output.

For production use cases, consider integrating with a third-party telemetry system (for example, Segment, Woopra) rather than creating your own telemetry server. In this case, use your provider’s APIs to send events from your custom back-end to their system.

The following Go code starts a server on port 8080 and writes events to standard output:

Example 9.1. main.go

```go
package main

import (
    "io/ioutil"
    "net/http"
    "go.uber.org/zap"
)
```
var logger *zap.SugaredLogger

func event(w http.ResponseWriter, req *http.Request) {
    switch req.Method {
    case "GET":
        logger.Info("GET /event")
    case "POST":
        logger.Info("POST /event")
    }
    body, err := req.GetBody()
    if err != nil {
        logger.With("err", err).Info("error getting body")
        return
    }
    responseBody, err := ioutil.ReadAll(body)
    if err != nil {
        logger.With("error", err).Info("error reading response body")
        return
    }
    logger.With("body", string(responseBody)).Info("got event")
}

func activity(w http.ResponseWriter, req *http.Request) {
    switch req.Method {
    case "GET":
        logger.Info("GET /activity, doing nothing")
    case "POST":
        logger.Info("POST /activity")
    body, err := req.GetBody()
    if err != nil {
        logger.With("error", err).Info("error getting body")
        return
    }
    responseBody, err := ioutil.ReadAll(body)
    if err != nil {
        logger.With("error", err).Info("error reading response body")
        return
    }
    logger.With("body", string(responseBody)).Info("got activity")
}

func main() {
    log, _ := zap.NewProduction()
    logger = log.Sugar()

    http.HandleFunc("/event", event)
    http.HandleFunc("/activity", activity)
    logger.Info("Added Handlers")
    logger.Info("Starting to serve")
    http.ListenAndServe(":8080", nil)
}
Create a container image based on this code and expose it as a deployment in OpenShift in the openshift-workspaces project. The code for the example telemetry server is available at che-workspace-telemetry-example. To deploy the telemetry server, clone the repository and build the container:

```bash
$ git clone https://github.com/che-incubator/che-workspace-telemetry-example
$ cd che-workspace-telemetry-example
$ docker build -t registry/organization/che-workspace-telemetry-example:latest .
$ docker push registry/organization/che-workspace-telemetry-example:latest
```

In `manifest.yaml`, replace the `image` and `host` fields to match the image you pushed, and the public hostname of your OpenShift cluster. Then run:

```bash
$ oc apply -f manifest.yaml -n {prod-namespace}
```

### 9.3.2. Creating a new Maven project

**NOTE**

For fast feedback when developing, it is recommended to do development inside a CodeReady Workspaces workspace. This way, you can run the application in a cluster and connect to the workspaces front-end telemetry plug-in to send events to your custom back-end.

1. Create a new Maven Quarkus project scaffolding:

```bash
$ mvn io.quarkus:quarkus-maven-plugin:1.2.1.Final:create \
   -DprojectGroupId=mygroup -DprojectArtifactId=telemetryback-end \
   -DprojectVersion=my-version -DclassName="org.my.group.MyResource"
```

2. Add a dependency to `org.eclipse.che.incubator.workspace-telemetry.back-end-base` in your `pom.xml`:

   **Example 9.2. pom.xml**

   ```xml
   <dependency>
   <groupId>org.eclipse.che.incubator.workspace-telemetry-telemetry-back-end</groupId>
   <artifactId>backend-base</artifactId>
   <version>0.0.11</version>
   </dependency>
   </dependency>

   <dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpclient</artifactId>
   <version>4.5.12</version>
   </dependency>

3. Add the Apache HTTP components library to send HTTP requests.

4. Consult the GitHub packages for the latest version and Maven coordinates of `back-end-base`. GitHub packages require a personal access token with `read:packages` permissions to
4. Create a settings.xml file in the repository root and add the coordinates and token to the che-incubator packages:

Example 9.3. settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>che-incubator</id>
      <username>${env.GITHUB_USERNAME}</username>
      <password>${env.GITHUB_TOKEN}</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>github</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>false</enabled></snapshots>
        </repository>
        <repository>
          <id>che-incubator</id>
          <name>GitHub navikt Apache Maven Packages</name>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

This file is used when packaging the application in a container. When running locally, add the information to your personal settings.xml file.

9.3.3. Running the application

Run and test the application is in a CodeReady Workspaces workspace:

```
$ mvn quarkus:dev -Dquarkus.http.port=${CHE_WORKSPACE_TELEMETRY_BACKEND_PORT}
```

If CodeReady Workspaces is secured using a self-signed certificate, add the certificate to a trust
store and mount it into the workspace. Also add the Java system property, `-Djavax.net.ssl.trustStore=/path/to/trustStore`, to the `mvn` command. For example, assuming the trust store is located in `$JAVA_HOME/jre/lib/security/cacerts`:

```
$ keytool -import -alias self-signed-certificate \
   -file <path/to/self-signed-certificate> -keystore $JAVA_HOME/jre/lib/security/cacerts
```

Followed by:

```
$ mvn quarkus:dev -Dquarkus.http.port=${CHE_WORKSPACE_TELEMETRY_BACKEND_PORT} \
   -Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security/cacerts
```

9.3.4. Creating a concrete implementation of AnalyticsManager and adding specialized logic

Create two new files in your project:

- **AnalyticsManager.java** - contains the logic specific to the telemetry system.
- **MainConfiguration.java** - is the main entrypoint that creates an instance of AnalyticsManager and starts listening for events.

Example 9.4. AnalyticsManager.java

```java
package org.my.group;

import java.util.Map;
import org.eclipse.che.api.core.rest.HttpJsonRequestFactory;
import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.AnalyticsEvent;

public class AnalyticsManager extends AbstractAnalyticsManager {
    public AnalyticsManager(String apiEndpoint, String workspaceId, String machineToken,
                            HttpJsonRequestFactory requestFactory) {
        super(apiEndpoint, workspaceId, machineToken, requestFactory);
    }

    @Override
    public boolean isEnabled() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent,
                        String resolution,
                        Map<String, Object> properties) {
```
Example 9.5. MainConfiguration.java

```java
package org.my.group;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Produces;

import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.BaseConfiguration;

@Dependent
public class MainConfiguration extends BaseConfiguration {
    @Produces
    public AbstractAnalyticsManager analyticsManager() {
        return new AnalyticsManager(apiEndpoint, workspaceId, machineToken, requestFactory());
    }
}
```

9.3.5. Implementing isEnabled()

For the purposes of the example, this method just returns `true` whenever it is called. Whenever the server is running, it is enabled and operational.

Example 9.6. AnalyticsManager.java

```java
@Override
public boolean isEnabled() {
    return true;
}
```

It is possible to put more a complex login in `isEnabled()`. For example, the service should not be considered operational in certain cases. The hosted CodeReady Workspaces woopra back-end checks that a configuration property exists before determining if the back-end is enabled.
9.3.6. Implementing onEvent()

onEvent() sends the event passed to the back-end to the telemetry system. For the example application, it sends an HTTP POST payload to the telemetry server. The example telemetry server application is deployed to OpenShift at the following URL: http://little-telemetry-back-end-che.apps-crc.testing.

Example 9.7. AnalyticsManager.java

```java
@override
public void onEvent(AnalyticsEvent event, String ownerid, String ip, String userAgent,
String resolution, Map<String, Object> properties) {
    HttpClient httpClient = HttpClients.createDefault();
   HttpPost httpPost = new HttpPost("http://little-telemetry-backend-che.apps-
   crc.testing/event");
    HashMap<String, Object> eventPayload = new HashMap<String, Object>(properties);
    eventPayload.put("event", event);
    StringEntity requestEntity = new StringEntity(new
        JsonObject(eventPayload).toString(),
        ContentType.APPLICATION_JSON);
    httpPost.setEntity(requestEntity);
    try {
        HttpResponse response = httpClient.execute(httpPost);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

This sends an HTTP request to the telemetry server and automatically delays identical events for a small period of time. The default duration is 1500 milliseconds. You can modify the duration by setting subclasses.

9.3.7. Implementing increaseDuration()

Many telemetry systems recognize event duration. The AbstractAnalyticsManager merges similar events that happen in the same frame of time into one event. This implementation of increaseDuration() is a no-op. This method uses the APIs of the user’s telemetry provider to alter the event or event properties to reflect the increased duration of an event.

Example 9.8. AnalyticsManager.java

```java
@override
public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) {}
```

9.3.8. Implementing onActivity()

Set an inactive timeout limit, and use onActivity() to send a WORKSPACE_INACTIVE event if the last event time is longer than the inactivity timeout.

Example 9.9. AnalyticsManager.java

```java
public class AnalyticsManager extends AbstractAnalyticsManager {
```
9.3.9. Implementing destroy()

When `destroy()` is called, send a `WORKSPACE_STOPPED` event and shutdown any resources, such as connection pools.

Example 9.10. AnalyticsManager.java

```java
@Override
public void onActivity() {
    if (System.currentTimeMillis() - lastEventTime >= inactiveTimeLimit) {
        onEvent(WORKSPACE_INACTIVE, lastOwnerId, lastIp, lastUserAgent, lastResolution, commonProperties);
    }
}

@Override
public void destroy() {
    onEvent(WORKSPACE_STOPPED, lastOwnerId, lastIp, lastUserAgent, lastResolution, commonProperties);
}
```

Running `mvn quarkus:dev` as described in Section 9.3.3, “Running the application” displays the `WORKSPACE_STOPPED` event, sent to the server when the Quarkus application is terminated.

9.3.10. Packaging the Quarkus application

See the quarkus documentation for the best instructions to package the application in a container. Build and push the container to a container registry of your choice.

9.3.11. Creating a `meta.yaml` for your plug-in

Create a `meta.yaml` definition representing a CodeReady Workspaces plug-in that runs your custom back-end in a workspace Pod. For more information about `meta.yaml`, see Section 5.1, “What is a Che-Theia plug-in”.

Example 9.11. `meta.yaml`

```yaml
apiVersion: v2
publisher: demo-publisher
name: little-telemetry-backend
version: 0.0.1
type: Che Plugin
displayName: Little Telemetry Backend
```
Typically, the user deploys this file to a corporate web server. This guide demonstrates how to create an Apache web server on OpenShift and host the plug-in there.

Create a ConfigMap referencing the new `meta.yaml` file.

```bash
$ oc create configmap --from-file=meta.yaml -n openshift-workspaces telemetry-plugin-meta
```

Create a deployment, a service, and a route to expose the web server. The deployment references this ConfigMap and places it in the `/var/www/html` directory.

Example 9.12. manifests.yaml

```yaml
description: A Demo telemetry backend
title: Little Telemetry Backend
category: Other
spec:
  workspaceEnv:
  - name: CHE_WORKSPACE_TELEMETRY_BACKEND_PORT
    value: '4167'
  containers:
  - name: YOUR BACKEND NAME
    image: YOUR IMAGE NAME
    env:
      - name: CHE_API
        value: $(CHE_API_INTERNAL)

kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      volumes:
      - name: plugin-meta-yaml
        configMap:
          name: telemetry-plugin-meta
          defaultMode: 420
      containers:
      - name: apache
        image: 'registry.redhat.io/rhscl/httpd-24-rhel7:latest'
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
        volumeMounts:
Wait a few minutes for the image to pull and the deployment to start, and then confirm that `meta.yaml` is available in the web server:

```bash
$ curl apache-che.apps-crc.testing/meta.yaml
```

This command should return the `meta.yaml` file.

### 9.3.12. Updating CodeReady Workspaces to reference your telemetry plug-in

Update the `CheCluster` Custom Resource, and add the `CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS` environment variable to `spec.server.customCheProperties`. The value of the environment variable must be the URL of the

```
- name: plugin-meta-yaml
  mountPath: /var/www/html

strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 25%
    maxSurge: 25%
    revisionHistoryLimit: 10
    progressDeadlineSeconds: 600

---
kind: Service
apiVersion: v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: apache
  type: ClusterIP

---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  host: apache-che.apps-crc.testing
to:
  kind: Service
  name: apache
  weight: 100
  port:
  targetPort: 8080
  wildcardPolicy: None
```

```bash
$ oc apply -f manifests.yaml
```
location of the meta.yaml file on your web server. This can be accomplished by running `oc edit checluster -n openshift-workspaces` and typing in the change at the terminal, or by editing the CR in the OpenShift console (Installed Operators → Red Hat CodeReady Workspaces → Red Hat CodeReady Workspaces Cluster → codeready-workspaces → YAML).

Example 9.13. Example of a YAML file

```yaml
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  creationTimestamp: '2020-05-14T13:21:51Z'
finalizers:
  - oauthclients.finalizers.che.eclipse.org
generation: 18
name: codeready-workspaces
namespace: <openshift-workspaces>
resourceVersion: '5108404'
selfLink: /apis/org.eclipse.che/v1/namespaces/che/checlusters/eclipse-che
uid: bae08db2-104d-4e44-a001-c9affc07528d
spec:
  auth:
    identityProviderURL: 'https://keycloak-che.apps-crc.testing'
    identityProviderRealm: che
    updateAdminPassword: false
    oAuthSecret: ZMmNPbgOJJQ
    oAuthClientName: eclipse-che-openshift-identity-provider-yrlcxs
    identityProviderClientId: che-public
    identityProviderPostgresSecret: che-identity-postgres-secret
    externalIdentityProvider: false
    identityProviderSecret: che-identity-secret
    openShiftToAuth: true
  database:
    chePostgresDb: dbche
    chePostgresHostName: postgres
    chePostgresPort: '5432'
    chePostgresSecret: che-postgres-secret
    externalDb: false
  k8s: {}
  metrics:
    enable: false
  server:
    cheLogLevel: INFO
    customCheProperties:
      CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS: 'http://apache-che.apps-crc.testing/meta.yaml'
    externalDevfileRegistry: false
    cheHost: che-che.apps-crc.testing
    selfSignedCert: true
    cheDebug: 'false'
    tlsSupport: true
    allowUserDefinedWorkspaceNamespaces: false
    externalPluginRegistry: false
    gitSelfSignedCert: false
    cheFlavor: che
    storage:
      preCreateSubPaths: true
```
Wait for the CodeReady Workspaces server to restart, and create a new workspace. See a new message stating that the plug-in is being installed into the workspace.

- **WOOPRA_DOMAIN** - The Woopra domain to send events to.
- **SEGMENT_WRITE_KEY** - The write key to send events to Segment and Woopra.
- **WOOPRA_DOMAIN_ENDPOINT** - If you prefer not to pass in the Woopra domain directly, the plug-in will get it from a supplied HTTP endpoint that returns the Woopra Domain.
- **SEGMENT_WRITE_KEY_ENDPOINT** - If you prefer not to pass in the Segment write key directly, the plug-in will get it from a supplied HTTP endpoint that returns the Segment write key.

To enable the Woopra plug-in on the Red Hat CodeReady Workspaces installation, deploy the *meta.yaml* file to an HTTP server with the environment variables set correctly. Then, edit the CheCluster Custom Resource, and set the **spec.server.customCheProperties.CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS** field:

```
spec:
  server:
    customCheProperties:
```
CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS: 'eclipse/che-machine-exec-plugin/7.20.0,https://your-web-server/meta.yaml'
CHAPTER 10. JAVA LOMBOK

This section shows how to enable Lombok support in your Java projects. By default, the lombok.jar file is available in all Java plug-ins provided by CodeReady Workspaces.

To enable Lombok in a CodeReady Workspaces workspace, see the instructions below.

Prerequisites

- A workspace or a devfile with:
  - One of the Java-based plug-ins enabled (redhat/java, redhat/java11, redhat/java8, redhat/quarkus-java8 or redhat/quarkus-java11)
  - A valid Lombok project to import

Procedure

1. Open the workspace devfile.
2. Edit the existing Java plug-in section, adding the preference:

   ```
   - id: redhat/java/latest
     preferences:
       java.jdt.ls.vmargs: '-javaagent:/lombok.jar'
   ```

Verification

1. Start or restart the workspace.
2. Open a file containing Lombok annotations.
3. Verify that the Class outline contains the Lombok generated methods.
Additional resources

- For more details, see the Project Lombok website.