



# Red Hat OpenShift Dev Spaces 3.0

## User guide

Using Red Hat OpenShift Dev Spaces 3.0



# Red Hat OpenShift Dev Spaces 3.0 User guide

---

Using Red Hat OpenShift Dev Spaces 3.0

Robert Kratky  
rkratky@redhat.com

Fabrice Flore-Thébault  
ffloreth@redhat.com

Jana Vrbkova  
jvrbkova@redhat.com

Max Leonov  
mleonov@redhat.com

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

Information for users using Red Hat OpenShift Dev Spaces.

## Table of Contents

<b>CHAPTER 1. ADOPTING OPENSIFT DEV SPACES</b> .....	<b>5</b>
1.1. DEVELOPER WORKSPACES	5
1.2. USING A BADGE WITH A LINK TO ENABLE A FIRST-TIME CONTRIBUTOR TO START A WORKSPACE	5
1.3. BENEFITS OF REVIEWING PULL AND MERGE REQUESTS IN RED HAT OPENSIFT DEV SPACES	6
1.4. SUPPORTED LANGUAGES	7
<b>CHAPTER 2. USER ONBOARDING</b> .....	<b>9</b>
2.1. STARTING A NEW WORKSPACE WITH A CLONE OF A GIT REPOSITORY	9
2.2. OPTIONAL PARAMETERS FOR THE URLS FOR STARTING A NEW WORKSPACE	11
2.2.1. URL parameter concatenation	11
2.2.2. URL parameter for the workspace IDE	12
2.2.3. URL parameter for starting duplicate workspaces	12
2.2.4. URL parameter for the devfile file name	13
2.2.5. URL parameter for the devfile file path	13
2.3. BASIC ACTIONS YOU CAN PERFORM ON A WORKSPACE	13
2.4. AUTHENTICATING YOURSELF TO A GIT SERVER FROM A WORKSPACE	14
<b>CHAPTER 3. CUSTOMIZING WORKSPACE COMPONENTS</b> .....	<b>15</b>
<b>CHAPTER 4. SELECTING A WORKSPACE IDE</b> .....	<b>16</b>
4.1. SELECTING AN IN-BROWSER IDE FOR A NEW WORKSPACE BY USING A URL PARAMETER	16
4.2. SPECIFYING AN IN-BROWSER IDE FOR A GIT REPOSITORY BY USING CHE-EDITOR.YAML	16
4.2.1. Using the OpenShift Dev Spaces editor file to select an IDE	16
4.2.2. Customizing IDE selection with the che-editor.yaml file	17
4.2.3. Using a custom plug-in registry for your IDE	17
4.2.4. Using a web reference for your IDE	17
4.2.5. Using an embedded editor definition for your IDE	17
<b>CHAPTER 5. USING CREDENTIALS AND CONFIGURATIONS IN WORKSPACES</b> .....	<b>19</b>
5.1. USING A GIT CREDENTIALS STORE	19
5.2. ENABLING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT	21
5.2.1. Enabling Maven artifact repositories	21
5.2.2. Enabling Gradle artifact repositories	23
5.2.3. Enabling npm artifact repositories	24
5.2.4. Enabling Python artifact repositories	25
5.2.5. Enabling Go artifact repositories	26
5.2.6. Enabling NuGet artifact repositories	28
5.3. CREATING IMAGE PULL SECRETS	29
5.3.1. Creating an image pull Secret with oc	29
5.3.2. Creating an image pull Secret from a .dockercfg file	30
5.3.3. Creating an image pull Secret from a config.json file	30
5.4. MOUNTING SECRETS	31
5.5. MOUNTING CONFIGMAPS	32
<b>CHAPTER 6. REQUESTING PERSISTENT STORAGE FOR WORKSPACES</b> .....	<b>35</b>
6.1. REQUESTING PERSISTENT STORAGE IN A DEVFILE	35
6.2. REQUESTING PERSISTENT STORAGE IN A PVC	36
<b>CHAPTER 7. INTEGRATING WITH OPENSIFT</b> .....	<b>38</b>
7.1. AUTOMATIC OPENSIFT TOKEN INJECTION	38
7.2. NAVIGATING OPENSIFT DEV SPACES FROM OPENSIFT DEVELOPER PERSPECTIVE	38
7.2.1. OpenShift Developer Perspective integration with OpenShift Dev Spaces	39
7.2.2. Editing the code of applications running in OpenShift Container Platform using OpenShift Dev Spaces	

	39
7.2.3. Accessing OpenShift Dev Spaces from Red Hat Applications menu	40
7.3. NAVIGATING OPENSIFT WEB CONSOLE FROM OPENSIFT DEV SPACES	41
<b>CHAPTER 8. TROUBLESHOOTING OPENSIFT DEV SPACES</b>	<b>42</b>
8.1. VIEWING OPENSIFT DEV SPACES WORKSPACES LOGS	42
8.1.1. Viewing logs from language servers and debug adapters	42
8.1.1.1. Checking important logs	42
8.1.1.2. Detecting memory problems	42
8.1.1.3. Logging the client-server traffic for debug adapters	43
8.1.1.4. Viewing logs for Python	43
8.1.1.5. Viewing logs for Go	43
8.1.1.5.1. Finding the Go path	43
8.1.1.5.2. Viewing the Debug Console log for Go	44
8.1.1.5.3. Viewing the Go logs output in the Output panel	45
8.1.1.6. Viewing logs for the NodeDebug NodeDebug2 adapter	45
8.1.1.7. Viewing logs for Typescript	45
8.1.1.7.1. Enabling the label switched protocol (LSP) tracing	45
8.1.1.7.2. Viewing the Typescript language server log	45
8.1.1.7.3. Viewing the Typescript logs output in the Output panel	46
8.1.1.8. Viewing logs for Java	46
8.1.1.8.1. Verifying the state of the Eclipse JDT Language Server	46
8.1.1.8.2. Verifying the Eclipse JDT Language Server features	46
8.1.1.8.3. Viewing the Java language server log	47
8.1.1.8.4. Logging the Java language server protocol (LSP) messages	47
8.1.1.9. Viewing logs for Intelephense	47
8.1.1.9.1. Logging the Intelephense client-server communication	47
8.1.1.9.2. Viewing Intelephense events in the Output panel	47
8.1.1.10. Viewing logs for PHP-Debug	48
8.1.1.11. Viewing logs for XML	48
8.1.1.11.1. Verifying the state of the XML language server	48
8.1.1.11.2. Checking XML language server feature flags	48
8.1.1.11.3. Enabling XML Language Server Protocol (LSP) tracing	49
8.1.1.11.4. Viewing the XML language server log	49
8.1.1.12. Viewing logs for YAML	49
8.1.1.12.1. Verifying the state of the YAML language server	49
8.1.1.12.2. Checking the YAML language server feature flags	50
8.1.1.12.3. Enabling YAML Language Server Protocol (LSP) tracing	50
8.1.1.13. Viewing logs for .NET with OmniSharp-Theia plug-in	51
8.1.1.13.1. OmniSharp-Theia plug-in	51
8.1.1.13.2. Verifying the state of the OmniSharp-Theia plug-in language server	51
8.1.1.13.3. Checking OmniSharp Che-Theia plug-in language server features	51
8.1.1.13.4. Viewing OmniSharp-Theia plug-in logs in the Output panel	51
8.1.1.14. Viewing logs for .NET with NetcoredebugOutput plug-in	51
8.1.1.14.1. NetcoredebugOutput plug-in	51
8.1.1.14.2. Verifying the state of the NetcoredebugOutput plug-in	52
8.1.1.14.3. Viewing NetcoredebugOutput plug-in logs in the Output panel	52
8.1.1.15. Viewing logs for Camel	52
8.1.1.15.1. Verifying the state of the Camel language server	52
8.1.1.15.2. Viewing Camel logs in the Output panel	53
8.1.2. Viewing Che-Theia IDE logs	53
8.1.2.1. Viewing Che-Theia editor logs using the OpenShift CLI	53
8.2. INVESTIGATING FAILURES AT A WORKSPACE START USING THE VERBOSE MODE	55

---

8.2.1. Restarting a OpenShift Dev Spaces workspace in Verbose mode after start failure	55
8.2.2. Starting a OpenShift Dev Spaces workspace in Verbose mode	55
8.3. TROUBLESHOOTING SLOW WORKSPACES	55
8.3.1. Improving workspace start time	56
8.3.2. Improving workspace runtime performance	57
8.4. TROUBLESHOOTING NETWORK PROBLEMS	58
<b>CHAPTER 9. ADDING A VISUAL STUDIO CODE EXTENSION TO A WORKSPACE</b> .....	<b>59</b>
9.1. OPENSIFT DEV SPACES PLUG-IN REGISTRIES OVERVIEW	59
9.2. ADDING AN EXTENSION TO .VSCODE/EXTENSIONS.JSON	59
9.3. ADDING PLUG-IN PARAMETERS TO .CHE/CHE-THEIA-PLUGINS.YAML	59
9.3.1. Defining the plug-ins for workspace installation	60
9.3.2. Changing the default memory limit	60
9.3.3. Overriding default preferences	60
9.4. DEFINING VISUAL STUDIO CODE EXTENSION ATTRIBUTES IN THE DEVFILE	61
9.4.1. Inlining .vscode/extensions.json file	61
9.4.2. Inlining .che/che-theia-plugins.yaml file	62





# CHAPTER 1. ADOPTING OPENSIFT DEV SPACES

To get started with adopting OpenShift Dev Spaces for your organization, you can read the following:

- [Section 1.1, “Developer workspaces”](#)
- [Section 1.2, “Using a badge with a link to enable a first-time contributor to start a workspace”](#)
- [Section 1.3, “Benefits of reviewing pull and merge requests in Red Hat OpenShift Dev Spaces”](#)
- [Section 1.4, “Supported languages”](#)

## 1.1. DEVELOPER WORKSPACES

Red Hat OpenShift Dev Spaces 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 OpenShift Dev Spaces workspace. Therefore, everything running in a OpenShift Dev Spaces workspace is running inside containers. This makes a OpenShift Dev Spaces workspace highly portable.

The embedded browser-based IDE is the point of access for everything running in a OpenShift Dev Spaces workspace. This makes a OpenShift Dev Spaces workspace easy to share.

## 1.2. USING A BADGE WITH A LINK TO ENABLE A FIRST-TIME CONTRIBUTOR TO START A WORKSPACE

To enable a first-time contributor to start a workspace with a project, add a badge with a link to your OpenShift Dev Spaces instance.

Figure 1.1. Factory badge



### Procedure

1. Substitute your OpenShift Dev Spaces URL ([```
\[\[Contribute\]\(https://www.eclipse.org/che/contribute.svg\)\]  
\(https://devspaces-<openshift\_deployment\_name>.<domain\_name>/#https://<your-repository-url>\)
```](https://devspaces-<i><openshift_deployment_name>.<domain_name></a>) and repository URL (<i><your-repository-url></i>), and add the link to your repository in the project <b>README.md</b> file.</li></ol></div><div data-bbox=)

2. The **README.md** file in your Git provider web interface displays the



factory badge. Click the badge to open a workspace with your project in your OpenShift Dev Spaces instance.

### 1.3. BENEFITS OF REVIEWING PULL AND MERGE REQUESTS IN RED HAT OPENSIFT DEV SPACES

Red Hat OpenShift Dev Spaces workspace contains all tools you need to review pull and merge requests from start to finish. By clicking a OpenShift Dev Spaces link, you get access to Red Hat OpenShift Dev Spaces-supported web IDE with a ready-to-use workspace where you can run a linter, unit tests, the build and more.

#### Prerequisites

- You have access to the repository hosted by your Git provider.
- You use a Red Hat OpenShift Dev Spaces-supported browser: Google Chrome or Mozilla Firefox.
- You have access to a OpenShift Dev Spaces instance.

#### Procedure

1. Open the feature branch to review in OpenShift Dev Spaces. A clone of the branch opens in a workspace with tools for debugging and testing.
2. Check the pull or merge request changes.
3. Run your desired debugging and testing tools:
  - Run a linter.
  - Run unit tests.
  - Run the build.
  - Run the application to check for problems.

4. Navigate to UI of your Git provider to leave comment and pull or merge your assigned request.

### Verification

- (optional) Open a second workspace using the main branch of the repository to reproduce a problem.

## 1.4. SUPPORTED LANGUAGES

### Java 11 with JBoss EAP 7.4

Java stack with OpenJDK 11, Maven 3.6 and JBoss EAP 7.4

### Java 11 with JBoss EAP XP 3.0 Bootable Jar

Java stack with OpenJDK 11, Maven 3.6 and JBoss EAP XP 3.0 Bootable Jar

### Java 11 with JBoss EAP XP 3.0 Microprofile

Java stack with OpenJDK 11, Maven 3.6 and JBoss EAP XP 3.0

### Red Hat Fuse

Red Hat Fuse stack with OpenJDK 11 and Maven 3.6.3

### Tooling for Apache Camel K

Tooling to develop Integration projects with Apache Camel K

### Java 11 with Gradle

Java stack with OpenJDK 11, Maven 3.6.3, and Gradle 6.1

### Java 11 with Lombok

Java stack with OpenJDK 11, Maven 3.6.3 and Lombok 1.18.18

### Java 11 with Quarkus

Java stack with OpenJDK 11, Maven 3.6.3, Gradle 6.1 and Quarkus Tools

### Java 11 with Vert.x

Java stack with OpenJDK 11, Maven 3.6.3 and Vert.x booster

### Java 11 with Maven

Java stack with OpenJDK 11, Maven 3.6.3 and Vert.x demo

### Java 8 with Spring Boot

Java stack with OpenJDK 8, Maven 3.6.3 and Spring Boot Petclinic demo application

### NodeJS ConfigMap Express

NodeJS stack with NPM 8, NodeJS 16 and ConfigMap Web Application

### NodeJS MongoDB

NodeJS stack with NPM 8, NodeJS 16 and MongoDB 3.6

### NodeJS Express

NodeJS stack with NPM 8, NodeJS 16 and Express Web Application

### Python

Python Stack with Python 3.8 and pip 19.3

### C/C++

C and C++ Developer Tools stack with GCC, cmake and make (Technology Preview)

### .NET

.NET stack with .NET Core SDK 6 and 3.1, Runtime, C# Language Support and Debugger  
(Technology Preview)

### **Go**

Stack with Go (Technology Preview)

### **PHP CakePHP**

PHP Stack with PHP, Apache Web Server, Composer and a quickstart CakePHP application for  
OpenShift (Technology Preview)

### **PHP-DI**

PHP Stack with PHP, Apache Web Server and Composer (Technology Preview)

## CHAPTER 2. USER ONBOARDING

If your organization is already running a OpenShift Dev Spaces instance, you can get started as a new user by learning how to start a new workspace, manage your workspaces, and authenticate yourself to a Git server from a workspace:

1. [Section 2.1, “Starting a new workspace with a clone of a Git repository”](#)
2. [Section 2.2, “Optional parameters for the URLs for starting a new workspace”](#)
3. [Section 2.3, “Basic actions you can perform on a workspace”](#)
4. [Section 2.4, “Authenticating yourself to a Git server from a workspace”](#)

### 2.1. STARTING A NEW WORKSPACE WITH A CLONE OF A GIT REPOSITORY

Working with OpenShift Dev Spaces in your browser involves multiple URLs:

- The URL of your organization’s OpenShift Dev Spaces instance, used as part of all the following URLs
- The URL of the **Workspaces** page of your OpenShift Dev Spaces dashboard with the workspace control panel
- The URLs for starting a new workspace
- The URLs of your workspaces in use

With OpenShift Dev Spaces, you can visit a URL in your browser to start a new workspace that contains a clone of a Git repository. This way, you can clone a Git repository that is hosted on GitHub, a GitLab instance, or a Bitbucket server.

#### TIP

You can also use the **Git Repo URL** \*field on the **Create Workspace** page of your OpenShift Dev Spaces dashboard to enter the URL of a Git repository to start a new workspace.

#### Prerequisites

- Your organization has a running instance of OpenShift Dev Spaces.
- You know the FQDN URL of your organization’s OpenShift Dev Spaces instance: **`https://devspaces-<openshift_deployment_name>.<domain_name>`**.
- Your Git repository maintainer keeps the **`devfile.yaml`** or **`.devfile.yaml`** file in the root directory of the Git repository. (For alternative file names and file paths, see [Section 2.2, “Optional parameters for the URLs for starting a new workspace”](#).)

#### TIP

You can also start a new workspace by supplying the URL of a Git repository that contains no devfile. Doing so results in a workspace with the Che-Theia IDE and the Universal Developer Image.

## Procedure

To start a new workspace with a clone of a Git repository:

1. Optional: Visit your OpenShift Dev Spaces dashboard pages to authenticate to your organization's instance of OpenShift Dev Spaces.
2. Visit the URL to start a new workspace using the basic syntax:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>
```

### TIP

You can extend this URL with optional parameters:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?<optional_parameters> 1
```

- 1** See [Section 2.2, "Optional parameters for the URLs for starting a new workspace"](#) .

#### Example 2.1. A URL for starting a new workspace

```
https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/<che-samples/cpp-hello-world>
```

#### Example 2.2. The URL syntax for starting a new workspace with a clone of a GitHub-hosted repository

With GitHub and GitLab, you can even use the URL of a specific branch of the repository to be cloned:

- **https://devspaces-*<openshift\_deployment\_name>*.*<domain\_name>*#https://github.com/*<user\_or\_org>*/*<repository>*** starts a new workspace with a clone of the default branch.
- **https://devspaces-*<openshift\_deployment\_name>*.*<domain\_name>*#https://github.com/*<user\_or\_org>*/*<repository>*/tree/*<branch\_name>*** starts a new workspace with a clone of the specified branch.
- **https://devspaces-*<openshift\_deployment\_name>*.*<domain\_name>*#https://github.com/*<user\_or\_org>*/*<repository>*/pull/*<pull\_request\_id>*** starts a new workspace with a clone of the branch of the pull request.

After you enter the URL to start a new workspace in a browser tab, it renders the workspace-starting page.

When the new workspace is ready, the workspace IDE loads in the browser tab.

A clone of the Git repository is present in the filesystem of the new workspace.

The workspace has a unique URL:

**`https://devspaces-<openshift_deployment_name>.<domain_name>#workspace<unique_url>`**.

## TIP

Although this is not possible in the address bar, you can add a URL for starting a new workspace as a bookmark by using the browser bookmark manager:

- In Mozilla Firefox, go to **☰ > Bookmarks > Manage bookmarks Ctrl+Shift+O > Bookmarks Toolbar > Organize > Add bookmark**.
- In Google Chrome, go to **⋮ > Bookmarks > Bookmark manager > Bookmarks bar > ⋮ > Add new bookmark**.

## Additional resources

- [Section 2.2, “Optional parameters for the URLs for starting a new workspace”](#)
- [Section 2.3, “Basic actions you can perform on a workspace”](#)

## 2.2. OPTIONAL PARAMETERS FOR THE URLS FOR STARTING A NEW WORKSPACE

When you start a new workspace, OpenShift Dev Spaces configures the workspace according to the instructions in the devfile. When you use a URL to start a new workspace, you can append optional parameters to the URL that further configure the workspace. You can use these parameters to specify a workspace IDE, start duplicate workspaces, and specify a devfile file name or path.

- [Section 2.2.1, “URL parameter concatenation”](#)
- [Section 2.2.2, “URL parameter for the workspace IDE”](#)
- [Section 2.2.3, “URL parameter for starting duplicate workspaces”](#)
- [Section 2.2.4, “URL parameter for the devfile file name”](#)
- [Section 2.2.5, “URL parameter for the devfile file path”](#)

### 2.2.1. URL parameter concatenation

The URL for starting a new workspace supports concatenation of multiple optional URL parameters by using **&** with the following URL syntax:

**`https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?<url_parameter_1>&<url_parameter_2>&<url_parameter_3>`**

**Example 2.3. A URL for starting a new workspace with the URL of a Git repository and optional URL parameters**

The complete URL for the browser:

**`https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/che-samples/cpp-hello-world?new&che-editor=che-incubator/intellij-community/latest&devfilePath=tests/testdevfile.yaml`**

Explanation of the parts of the URL:

```
https://devspaces-<openshift_deployment_name>.<domain_name> 1
#https://github.com/che-samples/cpp-hello-world 2
?new&che-editor=che-incubator/intellij-community/latest&devfilePath=tests/testdevfile.yaml 3
```

- 1** OpenShift Dev Spaces URL.
- 2** The URL of the Git repository to be cloned into the new workspace.
- 3** The concatenated optional URL parameters.

### 2.2.2. URL parameter for the workspace IDE

If the URL for starting a new workspace doesn't contain a URL parameter specifying the integrated development environment (IDE), the workspace loads with the default IDE: Che Theia.

The URL parameter for specifying another supported IDE is **che-editor=<editor\_key>**:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?che-
editor=<editor_key>
```

Table 2.1. The URL parameter **<editor\_key>** values for supported IDEs

| IDE           | <b>&lt;editor_key&gt;</b> value      | Note                                                                             |
|---------------|--------------------------------------|----------------------------------------------------------------------------------|
| Che-Theia     | <b>eclipse/che-theia/latest</b>      | This is the default IDE that loads in a new workspace without the URL parameter. |
| IntelliJ IDEA | <b>che-incubator/che-idea/latest</b> | Community Edition - stable version                                               |

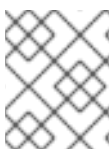
### 2.2.3. URL parameter for starting duplicate workspaces

Visiting a URL for starting a new workspace results in a new workspace according to the devfile and with a clone of the linked Git repository.

In some situations, you may need to have multiple workspaces that are duplicates in terms of the devfile and the linked Git repository. You can do this by visiting the same URL for starting a new workspace with a URL parameter.

The URL parameter for starting a duplicate workspace is **new**:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?new
```



#### NOTE

If you currently have a workspace that you started using a URL, then visiting the URL again without the **new** URL parameter results in an error message.



## 2.2.4. URL parameter for the devfile file name

When you visit a URL for starting a new workspace, OpenShift Dev Spaces searches the linked Git repository for a devfile with the file name **.devfile.yaml** or **devfile.yaml**. The devfile in the linked Git repository must follow this file-naming convention.

In some situations, you may need to specify a different, unconventional file name for the devfile.

The URL parameter for specifying an unconventional file name of the devfile is **df=<filename>.yaml**:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?
df=<filename>.yaml 1
```

1 **<filename>.yaml** is an unconventional file name of the devfile in the linked Git repository.

### TIP

The **df=<filename>.yaml** parameter also has a long version: **devfilePath=<filename>.yaml**.

## 2.2.5. URL parameter for the devfile file path

When you visit a URL for starting a new workspace, OpenShift Dev Spaces searches the root directory of the linked Git repository for a devfile with the file name **.devfile.yaml** or **devfile.yaml**. The file path of the devfile in the linked Git repository must follow this path convention.

In some situations, you may need to specify a different, unconventional file path for the devfile in the linked Git repository.

The URL parameter for specifying an unconventional file path of the devfile is **devfilePath=<relative\_file\_path>**:

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?
devfilePath=<relative_file_path> 1
```

1 **<relative\_file\_path>** is an unconventional file path of the devfile in the linked Git repository.

## 2.3. BASIC ACTIONS YOU CAN PERFORM ON A WORKSPACE

You manage your workspaces and verify their current states in the **Workspaces** page ([https://devspaces-<openshift\\_deployment\\_name>.<domain\\_name>/dashboard/#/workspaces](https://devspaces-<openshift_deployment_name>.<domain_name>/dashboard/#/workspaces)) of your OpenShift Dev Spaces dashboard.

After you start a new workspace, you can perform the following actions on it in the **Workspaces** page:

Table 2.2. Basic actions you can perform on a workspace

| Action                      | GUI steps in the Workspaces page        |
|-----------------------------|-----------------------------------------|
| Reopen a running workspace  | Click <b>Open</b> .                     |
| Restart a running workspace | Go to <b>⋮ &gt; Restart Workspace</b> . |

| Action                           | GUI steps in the Workspaces page           |
|----------------------------------|--------------------------------------------|
| <i>Stop a running workspace</i>  | Go to <b>⋮</b> > <b>Stop Workspace</b> .   |
| <i>Start a stopped workspace</i> | Click <b>Open</b> .                        |
| <i>Delete a workspace</i>        | Go to <b>⋮</b> > <b>Delete Workspace</b> . |

## 2.4. AUTHENTICATING YOURSELF TO A GIT SERVER FROM A WORKSPACE

In a workspace, you can run Git commands that require user authentication like cloning a remote private Git repository or pushing to a remote public or private Git repository.

To configure user authentication to a Git server from a workspace, OpenShift Dev Spaces provides two options:

- Your administrator sets up an [OAuth application on GitHub, GitLab, or Bitbucket](#) for your organization's Red Hat OpenShift Dev Spaces instance.
- You create your own, user [Kubernetes Secret for a Git credentials store](#) .

### Additional resources

- [Administration Guide: OAuth for GitHub, GitLab, or Bitbucket](#)
- [User Guide: Using a Git credentials store](#)

## CHAPTER 3. CUSTOMIZING WORKSPACE COMPONENTS

To customize workspace components:

- [Choose a Git repository for your workspace](#) .
- Use a devfile that meets the latest devfile 2 specification. See [Devfile User Guide](#) .
- [Select and customize your in-browser IDE](#) .
- Add OpenShift Dev Spaces specific attributes in addition to the generic devfile specification.

## CHAPTER 4. SELECTING A WORKSPACE IDE

The default in-browser IDE in a new workspace is [Che Theia](#).

You can select another supported in-browser IDE by either method:

- When you start a new workspace by visiting a URL, you can choose an IDE for that workspace by adding the **che-editor** parameter to the URL. See [Section 4.1, “Selecting an in-browser IDE for a new workspace by using a URL parameter”](#).
- You can specify an IDE in the **.che/che-editor.yaml** file of the Git repository for all new workspaces that will feature a clone of that repository. See [Section 4.2, “Specifying an in-browser IDE for a Git repository by using che-editor.yaml”](#).

Table 4.1. Supported in-browser IDEs

| IDE                           | id                                   | Note                                                                             |
|-------------------------------|--------------------------------------|----------------------------------------------------------------------------------|
| <a href="#">Che-Theia</a>     | <b>eclipse/che-theia/latest</b>      | This is the default IDE that loads in a new workspace without the URL parameter. |
| <a href="#">IntelliJ IDEA</a> | <b>che-incubator/che-idea/latest</b> | Community Edition - stable version                                               |

### 4.1. SELECTING AN IN-BROWSER IDE FOR A NEW WORKSPACE BY USING A URL PARAMETER

You can select your preferred in-browser IDE when starting a new workspace. This is the easiest way and it doesn't affect your other workspaces or other users.

#### Procedure

1. Include the URL parameter for the workspace IDE in the URL for starting a new workspace. See [Section 2.2.2, “URL parameter for the workspace IDE”](#).
2. Visit the URL in the browser. See [Section 2.1, “Starting a new workspace with a clone of a Git repository”](#).

### 4.2. SPECIFYING AN IN-BROWSER IDE FOR A GIT REPOSITORY BY USING CHE-EDITOR.YAML

#### 4.2.1. Using the OpenShift Dev Spaces editor file to select an IDE

Use the **che-editor.yaml** file to define a default IDE for the project users. For a list of supported IDs, see [Optional parameters for the URLs for starting a new workspace](#).

#### Procedure

1. Place the **che-editor.yaml** file in the **.che** folder in the root directory of your project.
2. In the **che-editor.yaml** file, specify the ID of the IDE you are selecting. For example:

```
id: che-incubator/che-idea/latest
```

### Additional resources

- Check the sample file sample [here](#).
- Load experimental new IDEs from the default plug-in registry by using the IDs shown in [che-editors.yaml](#).

### 4.2.2. Customizing IDE selection with the `che-editor.yaml` file

You can further customize your IDE selection to suit the specific needs of the project by adding various directives to the **che-editor.yaml** file. These customization options include following directives:

- Custom plug-in registry
- Web reference
- Embedded editor definition

### 4.2.3. Using a custom plug-in registry for your IDE

To include different IDEs than the default list in the OpenShift Dev Spaces plug-in registry, use an optional **registryUrl** directive.

#### Procedure

- Set an optional **registryUrl** directive in your **che-editor.yaml** file. For example:

```
id: eclipse/che-theia/next      # mandatory
registryUrl: https://my-registry.com # optional
override:                      # optional
containers:
  - name: theia-ide
    memoryLimit: 1280Mi
```

### 4.2.4. Using a web reference for your IDE

Use a web reference for your IDE by pointing at a YAML file with the **reference** directive.

#### Procedure

- Set a **reference** directive in your **che-editor.yaml** file. For example:

```
reference: https://gist.github.com/.../che-editor.yaml # mandatory
override:  # optional
containers:
  - name: theia-ide
    memoryLimit: 1280Mi
```

### 4.2.5. Using an embedded editor definition for your IDE

If you have specific requirements for your project that aren't addressed by standard IDE behaviour, you can customize the project IDE by using the **inline** directive to put a complete IDE definition in the **che-editor.yaml** file.

## Procedure

- Set an **inline** directive in your **che-editor.yaml** file. For example:

```
inline:
  endpoints:
    - name: "theia"
      public: true
      targetPort: 3100
      attributes:
        protocol: http
        type: ide
        secure: true
        cookiesAuthEnabled: true
        discoverable: false
    (...)
  containers:
    - name: theia-ide
      image: "quay.io/eclipse/che-theia:next"
      env:
        - name: THEIA_PLUGINS
          value: local-dir:///plugins
      volumeMounts:
        - name: plugins
          path: "/plugins"
        - name: theia-local
          path: "/home/theia/.theia"
      mountSources: true
      ports:
        - exposedPort: 3100
      memoryLimit: "512M"
      cpuLimit: 1000m
      cpuRequest: 100m
  initContainers:
    - name: remote-runtime-injector
      image: "quay.io/eclipse/che-theia-endpoint-runtime-binary:next"
      volumeMounts:
        - name: remote-endpoint
          path: "/remote-endpoint"
          ephemeral: true
      env:
        - name: PLUGIN_REMOTE_ENDPOINT_EXECUTABLE
          value: /remote-endpoint/plugin-remote-endpoint
        - name: REMOTE_ENDPOINT_VOLUME_NAME
          value: remote-endpoint
```

## CHAPTER 5. USING CREDENTIALS AND CONFIGURATIONS IN WORKSPACES

You can use your credentials and configurations in your workspaces.

To do so, mount your credentials and configurations to the **DevWorkspace** containers in the OpenShift cluster of your organization's OpenShift Dev Spaces instance:

- Mount your credentials and sensitive configurations as Kubernetes [Secrets](#). One example is a [Git credentials store](#).
- Mount your non-sensitive configurations as Kubernetes [ConfigMaps](#).

If you need to allow the **DevWorkspace** Pods in the cluster to access container registries that require authentication, create an [image pull Secret](#) for the **DevWorkspace** Pods.

The mounting process uses the standard Kubernetes mounting mechanism and requires applying additional labels and annotations to your existing resources. Resources are mounted when starting a new workspace or restarting an existing one.

You can create permanent mount points for various components:

- Maven configuration, such as the **settings.xml** file
- SSH key pairs
- AWS authorization tokens
- Configuration files
- Persistent storage
- Git credentials store files

### Additional resources

- [Kubernetes Documentation: Secrets](#)
- [Kubernetes Documentation: ConfigMaps](#)

## 5.1. USING A GIT CREDENTIALS STORE

As an alternative to the [OAuth for GitHub, GitLab, or Bitbucket](#) that is configured by the administrator of your organization's OpenShift Dev Spaces instance, you can apply your Git credentials store as a Kubernetes Secret.

Apply the Kubernetes Secret in your user project of the OpenShift cluster of your organization's OpenShift Dev Spaces instance.

When you apply the Secret, a Git configuration file with the path to the mounted Git credentials store is automatically configured and mounted to the **DevWorkspace** containers in the cluster at **/etc/gitconfig**. This makes your Git credentials store available in your workspaces.

### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- **base64** command line tools are installed in the operating system you are using.

## Procedure

1. In your home directory, locate and open your **.git-credentials** file if you already have it. Alternatively, if you do not have this file, save a new **.git-credentials** file, using the [Git credentials storage format](#). Each credential is stored on its own line in the file:

```
https://<username>:<token>@<git_server_hostname>
```

### Example 5.1. A line in a .git-credentials file

```
https://trailblazer:ghp_WjtiOi5KRNL5OHJif0Mzy09mqlbd9X4BrF7y@github.com
```

2. Select credentials from your **.git-credentials** file for the Secret. Encode the selected credentials to Base64 for the next step.

## TIP

- To encode all lines in the file:  
**\$ cat .git-credentials | base64 | tr -d '\n'**
- To encode a selected line:  
**\$ echo -n '<copied\_and\_pasted\_line\_from\_.git-credentials>' | base64**

3. Create a new OpenShift Secret in your user project.

```
apiVersion: v1
kind: Secret
metadata:
  name: git-credentials-secret
labels:
  controller.devfile.io/git-credential: 'true' ❶
  controller.devfile.io/watch-secret: 'true'
annotations:
  controller.devfile.io/mount-path: /etc/secret ❷
data:
  credentials: <Base64_content_of_.git-credentials> ❸
```

- ❶ The **controller.devfile.io/git-credential** label marks the Secret as containing Git credentials.
- ❷ A custom absolute path in the **DevWorkspace** containers. The Secret is mounted as the **credentials** file at this path. The default path is `/`.
- ❸ The selected content from **.git-credentials** that you encoded to Base64 in the previous step.



**TIP**

You can create and apply multiple Git credentials Secrets in your user project. All of them will be copied into one Secret that will be mounted to the **DevWorkspace** containers. For example, if you set the mount path to **/etc/secret**, then the one Secret with all of your Git credentials will be mounted at **/etc/secret/credentials**. You must set all Git credentials Secrets in your user project to the same mount path. You can set the mount path to an arbitrary path because the mount path will be automatically set in the Git configuration file configured at **/etc/gitconfig**.

4. Apply the Secret:

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

## 5.2. ENABLING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT

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

- [Maven](#)
- [Gradle](#)
- [npm](#)
- [Python](#)
- [Go](#)
- [NuGet](#)

### 5.2.1. Enabling Maven artifact repositories

You can enable a Maven artifact repository in Maven workspaces that run in a restricted environment.

#### Prerequisites

- You are not running any Maven workspace.

#### Procedure

1. Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
```

```

controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1

```

- 1 Base64 encoding with disabled line wrapping.

2. Apply the ConfigMap to create the **settings.xml** file:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: settings-xml
annotations:
  controller.devfile.io/mount-as: subpath
  controller.devfile.io/mount-path: /home/user/.m2
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
  controller.devfile.io/watch-configmap: 'true'
data:
  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
  https://maven.apache.org/xsd/settings-1.0.0.xsd">
      <localRepository/>
      <interactiveMode/>
      <offline/>
      <pluginGroups/>
      <servers/>
      <mirrors>
        <mirror>
          <id>redhat-ga-mirror</id>
          <name>Red Hat GA</name>
          <url>https://<maven_artifact_repository_route>/repository/redhat-ga/</url>
          <mirrorOf>redhat-ga</mirrorOf>
        </mirror>
        <mirror>
          <id>maven-central-mirror</id>
          <name>Maven Central</name>
          <url>https://<maven_artifact_repository_route>/repository/maven-central/</url>
          <mirrorOf>maven-central</mirrorOf>
        </mirror>
        <mirror>
          <id>jboss-public-repository-mirror</id>
          <name>JBoss Public Maven Repository</name>
          <url>https://<maven_artifact_repository_route>/repository/jboss-public/</url>
          <mirrorOf>jboss-public-repository</mirrorOf>
        </mirror>
      </mirrors>
      <proxies/>
      <profiles/>
      <activeProfiles/>
    </settings>

```

3. Apply the ConfigMap for the TrustStore initialization script:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: init-truststore
annotations:
  controller.devfile.io/mount-as: subpath
  controller.devfile.io/mount-path: /home/user/
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
  controller.devfile.io/watch-configmap: 'true'
data:
  init-truststore.sh: |
    #!/usr/bin/env bash

    keytool -importcert -noprompt -file /home/user/certs/tls.cer -cacerts -storepass changeit

```

4. Start a Maven workspace.
5. Open a new terminal in the **tools** container.
6. Run `~/init-truststore.sh`.

## 5.2.2. Enabling Gradle artifact repositories

You can enable a Gradle artifact repository in Gradle workspaces that run in a restricted environment.

### Prerequisites

- You are not running any Gradle workspace.

### Procedure

1. Apply the Secret for the TLS certificate:

```

kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
annotations:
  controller.devfile.io/mount-path: /home/user/certs
  controller.devfile.io/mount-as: file
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
  controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1

```

- 1** Base64 encoding with disabled line wrapping.

2. Apply the ConfigMap for the TrustStore initialization script:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: init-truststore
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  init-truststore.sh: |
    #!/usr/bin/env bash

    keytool -importcert -noprompt -file /home/user/certs/tls.cer -cacerts -storepass changeit

```

3. Apply the ConfigMap for the Gradle init script:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: init-gradle
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/.gradle
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  init.gradle: |
    allprojects {
      repositories {
        mavenLocal ()
        maven {
          url "https://<gradle_artifact_repository_route>/repository/maven-public/"
          credentials {
            username "admin"
            password "passwd"
          }
        }
      }
    }

```

4. Start a Gradle workspace.
5. Open a new terminal in the **tools** container.
6. Run `~/init-truststore.sh`.

### 5.2.3. Enabling npm artifact repositories

You can enable an npm artifact repository in npm workspaces that run in a restricted environment.

#### Prerequisites

- You are not running any npm workspace.



### WARNING

Applying a ConfigMap that sets environment variables might cause a workspace boot loop.

If you encounter this behavior, remove the **ConfigMap** and edit the devfile directly.

### Procedure

1. Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1
```

- 1 Base64 encoding with disabled line wrapping.

2. Apply the ConfigMap to set the following environment variables in the **tools** container:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  NODE_EXTRA_CA_CERTS: /home/user/certs/tls.cer
  NPM_CONFIG_REGISTRY: >-
    https://<npm_artifact_repository_route>/repository/npm-all/
```

### 5.2.4. Enabling Python artifact repositories

You can enable a Python artifact repository in Python workspaces that run in a restricted environment.

## Prerequisites

- You are not running any Python workspace.



### WARNING

Applying a ConfigMap that sets environment variables might cause a workspace boot loop.

If you encounter this behavior, remove the **ConfigMap** and edit the devfile directly.

## Procedure

- Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1
```

- 1 Base64 encoding with disabled line wrapping.

- Apply the ConfigMap to set the following environment variables in the **tools** container:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  PIP_INDEX_URL: >-
    https://<python_artifact_repository_route>/repository/pypi-all/
  PIP_CERT: /home/user/certs/tls.cer
```

## 5.2.5. Enabling Go artifact repositories

You can enable a Go artifact repository in Go workspaces that run in a restricted environment.

## Prerequisites

- You are not running any Go workspace.



### WARNING

Applying a ConfigMap that sets environment variables might cause a workspace boot loop.

If you encounter this behavior, remove the **ConfigMap** and edit the devfile directly.

## Procedure

1. Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1
```

- 1 Base64 encoding with disabled line wrapping.

2. Apply the ConfigMap to set the following environment variables in the **tools** container:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  GOPROXY: >-
    http://<athens_proxy_route>
  SSL_CERT_FILE: /home/user/certs/tls.cer
```

## 5.2.6. Enabling NuGet artifact repositories

You can enable a NuGet artifact repository in NuGet workspaces that run in a restricted environment.

### Prerequisites

- You are not running any NuGet workspace.



#### WARNING

Applying a ConfigMap that sets environment variables might cause a workspace boot loop.

If you encounter this behavior, remove the **ConfigMap** and edit the devfile directly.

### Procedure

- Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> 1
```

- 1 Base64 encoding with disabled line wrapping.

- Apply the ConfigMap to set the environment variable for the path of the TLS certificate file in the **tools** container:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  SSL_CERT_FILE: /home/user/certs/tls.cer
```



3. Apply the ConfigMap to create the **nuget.config** file:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: init-nuget
annotations:
  controller.devfile.io/mount-as: subpath
  controller.devfile.io/mount-path: /projects
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
  controller.devfile.io/watch-configmap: 'true'
data:
  nuget.config: |
    <?xml version="1.0" encoding="UTF-8"?>
    <configuration>
      <packageSources>
        <add key="nexus2" value="https://<nuget_artifact_repository_route>/repository/nuget-
group/" />
      </packageSources>
      <packageSourceCredentials>
        <nexus2>
          <add key="Username" value="admin" />
          <add key="Password" value="passwd" />
        </nexus2>
      </packageSourceCredentials>
    </configuration>

```

## 5.3. CREATING IMAGE PULL SECRETS

To allow the **DevWorkspace** Pods in the OpenShift cluster of your organization's OpenShift Dev Spaces instance to access container registries that require authentication, create an image pull Secret.

You can create image pull Secrets by using **oc** or a **.dockercfg** file or a **config.json** file.

### 5.3.1. Creating an image pull Secret with oc

#### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).

#### Procedure

1. In your user project, create an image pull Secret with your private container registry details and credentials:

```

$ oc create secret docker-registry <Secret_name> \
  --docker-server=<registry_server> \
  --docker-username=<username> \
  --docker-password=<password> \
  --docker-email=<email_address>

```

2. Add the following label to the image pull Secret:

```
$ oc label secret <Secret_name> controller.devfile.io/devworkspace_pullsecret=true
controller.devfile.io/watch-secret=true
```

### 5.3.2. Creating an image pull Secret from a `.dockercfg` file

If you already store the credentials for the private container registry in a `.dockercfg` file, you can use that file to create an image pull Secret.

#### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- **base64** command line tools are installed in the operating system you are using.

#### Procedure

1. Encode the `.dockercfg` file to Base64:

```
$ cat .dockercfg | base64 | tr -d '\n'
```

2. Create a new OpenShift Secret in your user project:

```
apiVersion: v1
kind: Secret
metadata:
  name: <Secret_name>
labels:
  controller.devfile.io/devworkspace_pullsecret: 'true'
  controller.devfile.io/watch-secret: 'true'
data:
  .dockercfg: <Base64_content_of_.dockercfg>
type: kubernetes.io/dockercfg
```

3. Apply the Secret:

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

### 5.3.3. Creating an image pull Secret from a `config.json` file

If you already store the credentials for the private container registry in a `$HOME/.docker/config.json` file, you can use that file to create an image pull Secret.

#### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- **base64** command line tools are installed in the operating system you are using.

## Procedure

1. Encode the `$HOME/.docker/config.json` file to Base64.

```
$ cat config.json | base64 | tr -d '\n'
```

2. Create a new OpenShift Secret in your user project:

```
apiVersion: v1
kind: Secret
metadata:
  name: <Secret_name>
labels:
  controller.devfile.io/devworkspace_pullsecret: 'true'
  controller.devfile.io/watch-secret: 'true'
data:
  .dockerconfigjson: <Base64_content_of_config.json>
type: kubernetes.io/dockerconfigjson
```

3. Apply the Secret:

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

## 5.4. MOUNTING SECRETS

To mount confidential data into your workspaces, use Kubernetes Secrets.

Using Kubernetes Secrets, you can mount usernames, passwords, SSH key pairs, authentication tokens (for example, for AWS), and sensitive configurations.

Mount Kubernetes Secrets to the **DevWorkspace** containers in the OpenShift cluster of your organization's OpenShift Dev Spaces instance.

### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- You have created a new Secret or determined an existing one in your user project to mount to all **DevWorkspace** containers.

### Procedure

1. Determine an existing ConfigMap or Secret in your user project to mount to all workspace containers.
2. Set the required labels for mounting.

```
$ oc label secret <Secret_name> \
  controller.devfile.io/mount-to-devworkspace=true \
  controller.devfile.io/watch-secret=true
```

- Optional: Use the annotations to configure how the Secret is mounted.

Table 5.1. Optional annotations

| Annotation                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>controller.devfile.io/mount-path:</b> | Specifies the mount path.<br><br>Defaults to <code>/etc/secret/&lt;Secret_name&gt;</code> .                                                                                                                                                                                                                                                                                                                                                                   |
| <b>controller.devfile.io/mount-as:</b>   | Specifies how the resource should be mounted:<br><b>file</b> , <b>subpath</b> , or <b>env</b> .<br><br>Defaults to <b>file</b> .<br><br><b>mount-as: file</b> mounts the keys and values as files within the mount path.<br><br><b>mount-as: subpath</b> mounts the keys and values within the mount path using subpath volume mounts.<br><br><b>mount-as: env</b> mounts the keys and values as environment variables in all <b>DevWorkspace</b> containers. |

### Example 5.2. Mounting a Secret as a file

```

apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
  controller.devfile.io/watch-secret: 'true'
annotations:
  controller.devfile.io/mount-path: '/home/user/.m2'
data:
  settings.xml: <Base64_encoded_content>

```

When you start a workspace, the `/home/user/.m2/settings.xml` file will be available in the **DevWorkspace** containers.

With Maven, you can set a custom path for the `settings.xml` file. For example:

```
$ mvn --settings /home/user/.m2/settings.xml clean install
```

## 5.5. MOUNTING CONFIGMAPS

To mount non-confidential configuration data into your workspaces, use Kubernetes ConfigMaps.

Using Kubernetes ConfigMaps, you can mount non-sensitive data such as configuration values for an application.

Mount Kubernetes ConfigMaps to the **DevWorkspace** containers in the OpenShift cluster of your organization's OpenShift Dev Spaces instance.

### Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- You have created a new ConfigMap or determined an existing one in your user project to mount to all **DevWorkspace** containers.

### Procedure

1. Determine an existing ConfigMap in your user project to mount to all workspace containers.
2. Set the required labels for mounting.

```
$ oc label configmap <ConfigMap_name> \
  controller.devfile.io/mount-to-devworkspace=true \
  controller.devfile.io/watch-configmap=true
```

3. Optional: Use the annotations to configure how the ConfigMap is mounted.

Table 5.2. Optional annotations

| Annotation                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>controller.devfile.io/mount-path:</b> | Specifies the mount path.<br><br>Defaults to <code>/etc/config/&lt;ConfigMap_name&gt;</code> .                                                                                                                                                                                                                                                                                                                                                             |
| <b>controller.devfile.io/mount-as:</b>   | Specifies how the resource should be mounted:<br><b>file</b> , <b>subpath</b> , or <b>env</b> .<br><br>Defaults to <b>file</b> .<br><br><b>mount-as:file</b> mounts the keys and values as files within the mount path.<br><br><b>mount-as:subpath</b> mounts the keys and values within the mount path using subpath volume mounts.<br><br><b>mount-as:env</b> mounts the keys and values as environment variables in all <b>DevWorkspace</b> containers. |

### Example 5.3. Mounting a ConfigMap as environment variables

```
kind: ConfigMap
apiVersion: v1
```

```
metadata:  
  name: my-settings  
  labels:  
    controller.devfile.io/mount-to-devworkspace: 'true'  
    controller.devfile.io/watch-configmap: 'true'  
  annotations:  
    controller.devfile.io/mount-as: env  
data:  
  <env_var_1>: <value_1>  
  <env_var_2>: <value_2>
```

When you start a workspace, the **<env\_var\_1>** and **<env\_var\_2>** environment variables will be available in the **DevWorkspace** containers.

## CHAPTER 6. REQUESTING PERSISTENT STORAGE FOR WORKSPACES

OpenShift Dev Spaces workspaces and workspace data are ephemeral and are lost when the workspace stops.

To preserve the workspace state in persistent storage while the workspace is stopped, request a Kubernetes PersistentVolume (PV) for the **DevWorkspace** containers in the OpenShift cluster of your organization's OpenShift Dev Spaces instance.

You can request a PV by using the devfile or a Kubernetes PersistentVolumeClaim (PVC).

An example of a PV is the **/projects/** directory of a workspace, which is mounted by default for non-ephemeral workspaces.

Persistent Volumes come at a cost: attaching a persistent volume slows workspace startup.



### WARNING

Starting another, concurrently running workspace with a **ReadWriteOnce PV** may fail.

### Additional resources

- [Red Hat OpenShift Documentation: Understanding persistent storage](#)
- [Kubernetes Documentation: Persistent Volumes](#)

## 6.1. REQUESTING PERSISTENT STORAGE IN A DEVFILE

When a workspace requires its own persistent storage, request a PersistentVolume (PV) in the devfile, and OpenShift Dev Spaces will automatically manage the necessary PersistentVolumeClaims.

### Prerequisites

- You have not started the workspace.

### Procedure

1. Add a **volume** component in the devfile:

```
...
components:
...
- name: <chosen_volume_name>
  volume:
    size: <requested_volume_size>G
...
```

2. Add a **volumeMount** for the relevant **container** in the devfile:

```

...
components:
  - name: ...
    container:
      ...
      volumeMounts:
        - name: <chosen_volume_name_from_previous_step>
          path: <path_where_to_mount_the_PV>
      ...

```

### Example 6.1. A devfile that provisions a PV for a workspace to a container

When a workspace is started with the following devfile, the **cache** PV is provisioned to the **golang** container in the **./cache** container path:

```

schemaVersion: 2.1.0
metadata:
  name: mydevfile
components:
  - name: golang
    container:
      image: golang
      memoryLimit: 512Mi
      mountSources: true
      command: ['sleep', 'infinity']
      volumeMounts:
        - name: cache
          path: ./cache
  - name: cache
    volume:
      size: 2Gi

```

## 6.2. REQUESTING PERSISTENT STORAGE IN A PVC

You may opt to apply a PersistentVolumeClaim (PVC) to request a PersistentVolume (PV) for your workspaces in the following cases:

- Not all developers of the project need the PV.
- The PV lifecycle goes beyond the lifecycle of a single workspace.
- The data included in the PV are shared across workspaces.

### TIP

You can apply a PVC to the **DevWorkspace** containers even if the workspace is ephemeral and its devfile contains the **controller.devfile.io/storage-type: ephemeral** attribute.

### Prerequisites

- You have not started the workspace.



- An active **oc** session with administrative permissions to the destination OpenShift cluster. See [Getting started with the CLI](#).
- A PVC is created in your user project to mount to all **DevWorkspace** containers.

## Procedure

1. Add the **controller.devfile.io/mount-to-devworkspace: true** label to the PVC.

```
$ oc label persistentvolumeclaim <PVC_name> \ controller.devfile.io/mount-to-devworkspace=true
```

2. Optional: Use the annotations to configure how the PVC is mounted:

**Table 6.1. Optional annotations**

| Annotation                               | Description                                                                                                                                                                  |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>controller.devfile.io/mount-path:</b> | The mount path for the PVC.<br>Defaults to <b>/tmp/&lt;PVC_name&gt;</b> .                                                                                                    |
| <b>controller.devfile.io/read-only:</b>  | Set to <b>'true'</b> or <b>'false'</b> to specify whether the PVC is to be mounted as read-only.<br>Defaults to <b>'false'</b> , resulting in the PVC mounted as read-write. |

## Example 6.2. Mounting a read-only PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <pvc_name>
labels:
  controller.devfile.io/mount-to-devworkspace: 'true'
annotations:
  controller.devfile.io/mount-path: </example/directory> 1
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi 2
  volumeName: <pv_name>
  storageClassName: manual
  volumeMode: Filesystem
```

**1** The mounted PV is available at **</example/directory>** in the workspace.

**2** Example size value of the requested storage.

## CHAPTER 7. INTEGRATING WITH OPENSHIFT

- [Section 7.1, “Automatic OpenShift token injection”](#)
- [Section 7.2, “Navigating OpenShift Dev Spaces from OpenShift Developer Perspective”](#)
- [Section 7.3, “Navigating OpenShift web console from OpenShift Dev Spaces”](#)

### 7.1. AUTOMATIC OPENSHIFT TOKEN INJECTION

This section describes how to use the OpenShift user token that is automatically injected into workspace containers which allows running OpenShift Dev Spaces CLI commands against OpenShift cluster.

#### Procedure

1. Open the OpenShift Dev Spaces dashboard and start a workspace.
2. Once the workspace is started, open a terminal in the container that contains the OpenShift Dev Spaces CLI.
3. Execute OpenShift Dev Spaces CLI commands which allow you to run commands against OpenShift cluster. CLI can be used for deploying applications, inspecting and managing cluster resources, and viewing logs. OpenShift user token will be used during the execution of the commands.

```

File Edit Selection View Go Run Terminal Help
EXPLORER
  OPEN EDITORS
  CHE (WORKSPACE)
  golang-example
    .vscode
    appengine-hello
    gotypes
    defsuses
    doc
    hello
    hugeparam
      main.go
    implements
      main.go
    lookup
    nilfunc
    pkginfo
    skeleton
    typeandvalue
  main.go x
  golang-example > gotypes > implements > main.go > ...
1 package main
2
3 import (
4     "fmt"
5     "go/ast"
6     "go/importer"
7     "go/parser"
8     "go/token"
9     "go/types"
10    "log"
11 )
12
tools terminal 1 x
bash-4.4$ oc whoami
ibuziuk
bash-4.4$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
workspace4b49b911486945df-6d4c5ccddc-p59fm   5/5     Running   0           5m19s
bash-4.4$

```



#### WARNING

The automatic token injection currently works only on the OpenShift infrastructure.

### 7.2. NAVIGATING OPENSHIFT DEV SPACES 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:

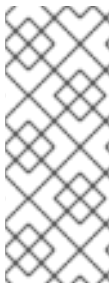
- Create and deploy applications on the OpenShift Container Platform by importing existing codebases, images, and Dockerfiles.
- Visually interact with applications, components, and services associated with them within a project and monitor their deployment and build status.
- Group components within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

### 7.2.1. OpenShift Developer Perspective integration with OpenShift Dev Spaces

This section provides information about OpenShift Developer Perspective support for OpenShift Dev Spaces.

When the OpenShift Dev Spaces 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 OpenShift Dev Spaces 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 OpenShift Dev Spaces **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 OpenShift Dev Spaces is used with HTTP resources

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

### 7.2.2. Editing the code of applications running in OpenShift Container Platform using OpenShift Dev Spaces

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

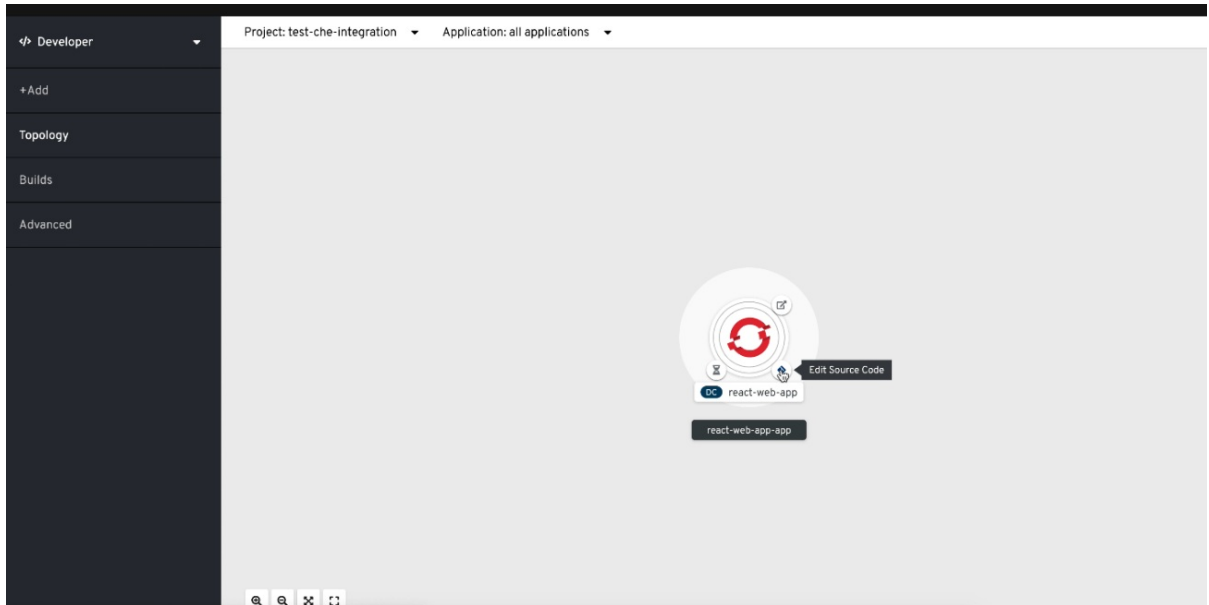
#### Prerequisites

- OpenShift Dev Spaces 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 OpenShift Dev Spaces, click the **Edit Source Code** button. This redirects to a workspace with the cloned source code of the application component.

### 7.2.3. Accessing OpenShift Dev Spaces from Red Hat Applications menu

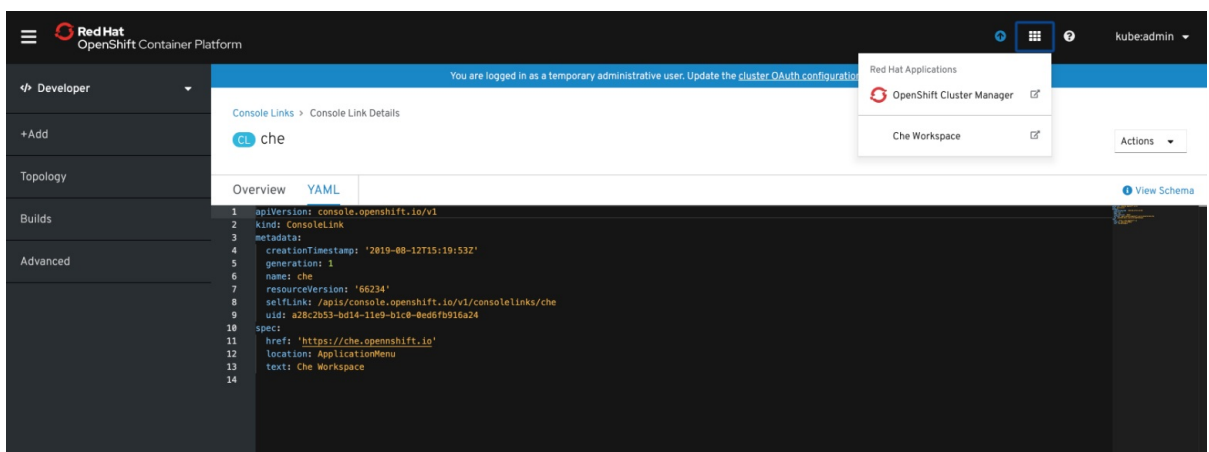
This section describes how to access OpenShift Dev Spaces workspaces from the **Red Hat Applications** menu on the OpenShift Container Platform.

#### Prerequisites

- The OpenShift Dev Spaces 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 **OpenShift Dev Spaces** link to open the Dev Spaces Dashboard.

## 7.3. NAVIGATING OPENSHIFT WEB CONSOLE FROM OPENSHIFT DEV SPACES

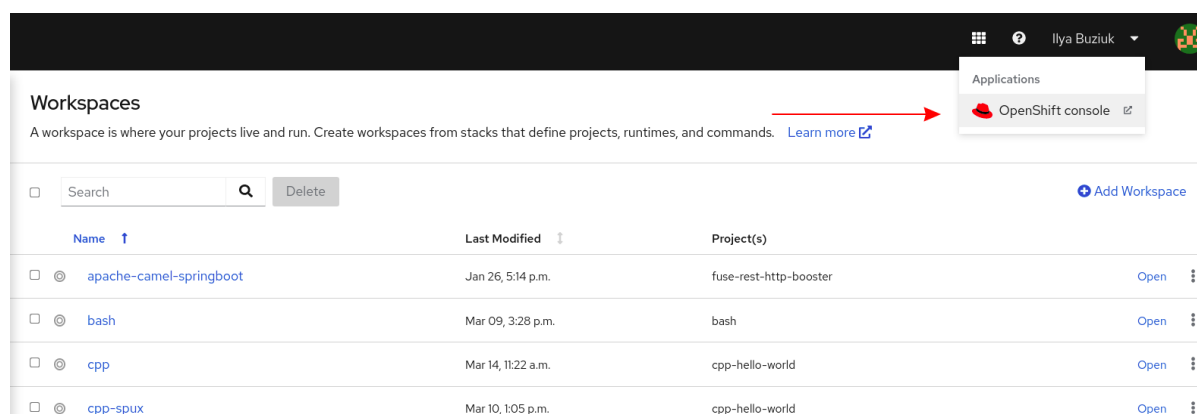
This section describes how to access OpenShift web console from OpenShift Dev Spaces.

### Prerequisites

- The OpenShift Dev Spaces Operator is available in OpenShift 4.

### Procedure

1. Open the OpenShift Dev Spaces dashboard and click 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 **OpenShift console** link to open the OpenShift web console.

## CHAPTER 8. TROUBLESHOOTING OPENSIFT DEV SPACES

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

### Additional resources

- [Section 8.1, “Viewing OpenShift Dev Spaces workspaces logs”](#)
- [Section 8.2, “Investigating failures at a workspace start using the Verbose mode”](#)
- [Section 8.3, “Troubleshooting slow workspaces”](#)
- [Section 8.4, “Troubleshooting network problems”](#)

## 8.1. VIEWING OPENSIFT DEV SPACES WORKSPACES LOGS

This section describes how to view OpenShift Dev Spaces workspaces logs.

### 8.1.1. Viewing logs from language servers and debug adapters

#### 8.1.1.1. 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.

#### 8.1.1.2. 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.

### 8.1.1.3. 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.

### 8.1.1.4. 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 x
Python
Starting Microsoft Python language server.
Downloading https://pvsc.azureedge.net/python-language-server-stable/Python-Language-Server-linux-x64.0.2.96.nupkg... #####Linting 0
***** Module test
12,0,error,import-error:Unable to import 'demonstrate'
-----
Your code has been rated at -2.50/10
complete
Unpacking archive... done

```

### 8.1.1.5. Viewing logs for Go

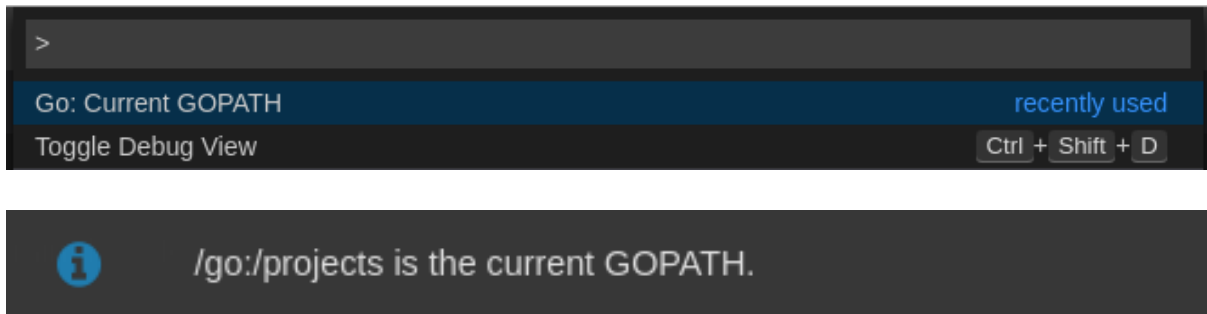
This section describes how to view logs for the Go language server.

#### 8.1.1.5.1. Finding the Go path

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

#### Procedure

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



### 8.1.1.5.2. 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.



```

Debug Console x
API server listening at: 127.0.0.1:22841
2019-06-18T18:51:06Z info layer=debugger launching process with args: [/projects/__debug_bin]
2019-06-18T18:51:07Z debug layer=rpc <- RPCServer.GetVersion(api.GetVersionIn{})
2019-06-18T18:51:07Z debug layer=rpc -> *api.GetVersionOut{"DelveVersion":{"Version: 1.2.0\nBuild: $Id: 068e2451004e95d0b042e5257e34f0f08ce01466 $"},"APIVersion":2} error: ""
2019-06-18T18:51:07Z debug layer=rpc (async 2) <-
RPCServer.Command(api.DebuggerCommand{"name":"continue","ReturnInfoLoadConfig":null})
2019-06-18T18:51:07Z debug layer=debugger continuing
2019-06-18T18:51:07Z debug layer=rpc (async 2) -> rpc2.CommandOut{"State":
{"Running":false,"Threads":null,"NextInProgress":false,"exited":true,"exitStatus":0,"When":""}} error: ""
2019-06-18T18:51:07Z debug layer=rpc (async 3) <-
RPCServer.Command(api.DebuggerCommand{"name":"halt","ReturnInfoLoadConfig":null})
2019-06-18T18:51:07Z debug layer=debugger halting
2019-06-18T18:51:07Z debug layer=rpc (async 3) -> null error: "Process 1219 has exited with status 0"
2019-06-18T18:51:07Z debug layer=rpc <- RPCServer.Detach(rpc2.DetachIn{"Kill":true})
2019-06-18T18:51:07Z debug layer=rpc -> *rpc2.DetachOut{} error: ""
Process exiting with code: 0

```

### 8.1.1.5.3. 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.

```

Output x
Starting linting the current package at /projects
Starting "go vet" under the folder /projects
Starting building the current package at /projects
Not able to determine import path of current package by using cwd: /projects and Go workspace:
/projects>Finished running tool: /go/bin/golint
/projects>Finished running tool: /usr/local/go/bin/go vet ./...
/projects>Finished running tool: /usr/local/go/bin/go build -i -o /tmp/vscode-go6JoFlE/go-code-check .

```

### 8.1.1.6. Viewing logs for the NodeDebug NodeDebug2 adapter



#### NOTE

No specific diagnostics exist other than the general ones.

### 8.1.1.7. Viewing logs for Typescript

#### 8.1.1.7.1. 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.

#### 8.1.1.7.2. Viewing the Typescript language server log

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

## Procedure

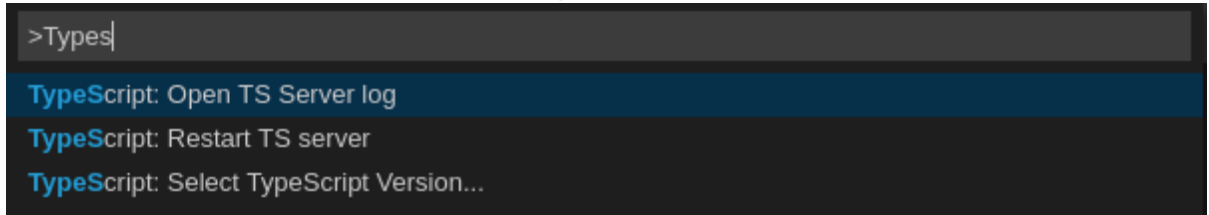
1. To get the path to the log file, see the Typescript **Output** console:



```

Info - 11:14:26 AM] Using tsserver from: /tmp/vscode-unpacked/che-incubator.typescript.latest.dvuuojoyht.che-typescript-language-1.35.1.vsix/extension/node_modules/typescript/Lib
Info - 11:14:26 AM] TSServer log file: /home/theia/.theia/logs/20190621T111312/host/vscode.typescript-language-features/tsserver-log-cdBAj1/tsserver.log
Info - 11:14:26 AM] Forking TSServer
Info - 11:14:26 AM] Started TSServer
  
```

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



```

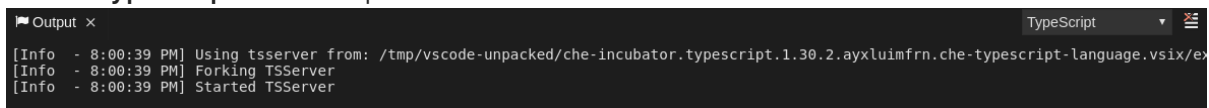
>TypeS|
TypeScript: Open TS Server log
TypeScript: Restart TS server
TypeScript: Select TypeScript Version...
  
```

### 8.1.1.7.3. 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.



```

[Info - 8:00:39 PM] Using tsserver from: /tmp/vscode-unpacked/che-incubator.typescript.1.30.2.ayxluimfrn.che-typescript-language.vsix/ex
[Info - 8:00:39 PM] Forking TSServer
[Info - 8:00:39 PM] Started TSServer
  
```

### 8.1.1.8. 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.

#### 8.1.1.8.1. 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 [Section 8.1, "Viewing OpenShift Dev Spaces workspaces logs"](#).

#### 8.1.1.8.2. 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.

### 8.1.1.8.3. 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**.

### 8.1.1.8.4. 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](#) .

### 8.1.1.9. Viewing logs for Intelephense

#### 8.1.1.9.1. 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**).

#### 8.1.1.9.2. 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.

### 8.1.1.10. 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.

### 8.1.1.11. Viewing logs for XML

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

#### 8.1.1.11.1. 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 is not, see the [Section 8.1, "Viewing OpenShift Dev Spaces workspaces logs"](#) chapter.

#### 8.1.1.11.2. 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

the editor.

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

#### 8.1.11.3. 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**.

#### 8.1.11.4. 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**.

#### 8.1.11.12. 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.

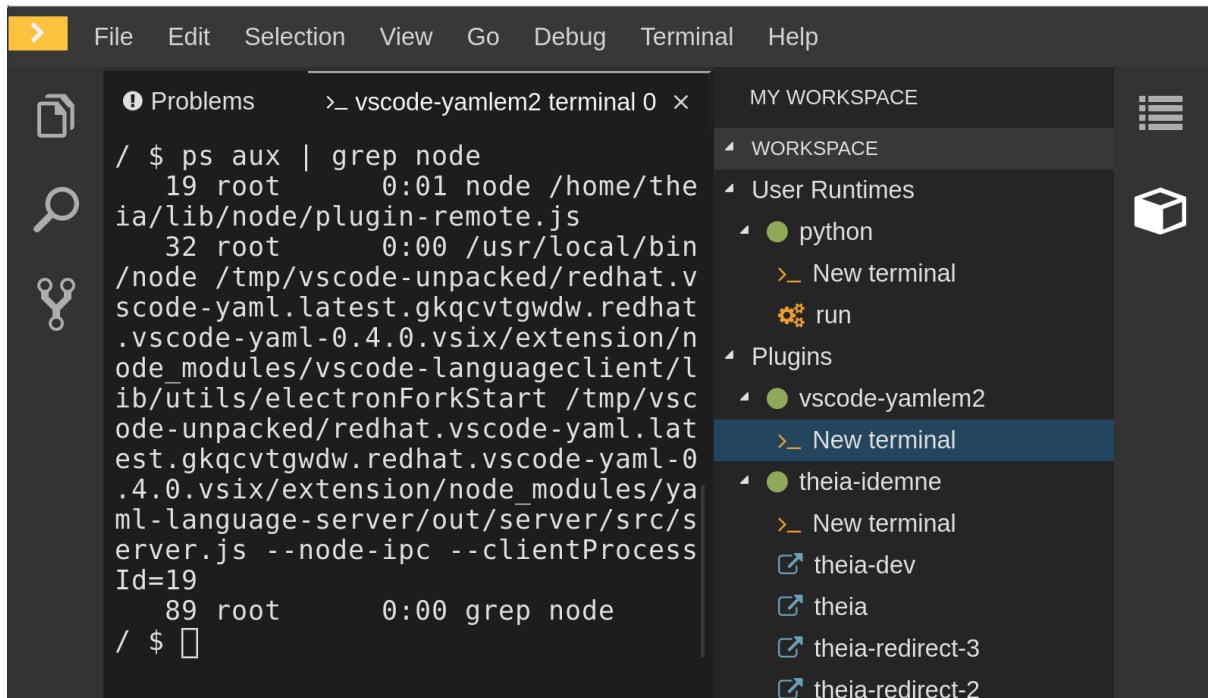
##### 8.1.11.12.1. Verifying the state of the YAML language server

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

##### Procedure

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

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



The `node **/server.js` running in the container indicates that the language server is running. If it is not running, the language server has not started inside the container. In this case, see [Section 8.1, “Viewing OpenShift Dev Spaces workspaces logs”](#).

#### 8.1.1.12.2. 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](#).

#### 8.1.1.12.3. 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`.

### 8.1.1.13. Viewing logs for .NET with OmniSharp-Theia plug-in

#### 8.1.1.13.1. OmniSharp-Theia plug-in

OpenShift Dev Spaces uses the OmniSharp-Theia plug-in as a remote plug-in. It is located at [github.com/redhat-developer/omnisharp-theia-plugin](https://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.

#### 8.1.1.13.2. 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 [Section 8.1, "Viewing OpenShift Dev Spaces workspaces logs"](#).

#### 8.1.1.13.3. 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.

#### 8.1.1.13.4. 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.

### 8.1.1.14. Viewing logs for .NET with NetcoredebugOutput plug-in

#### 8.1.1.14.1. 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.

### 8.1.1.14.2. Verifying the state of the NetcoredebugOutput plug-in

#### Procedure

1. Search for a `netcoredbg` debug configuration in the `launch.json` file.

Example 8.1. Sample debug configuration

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

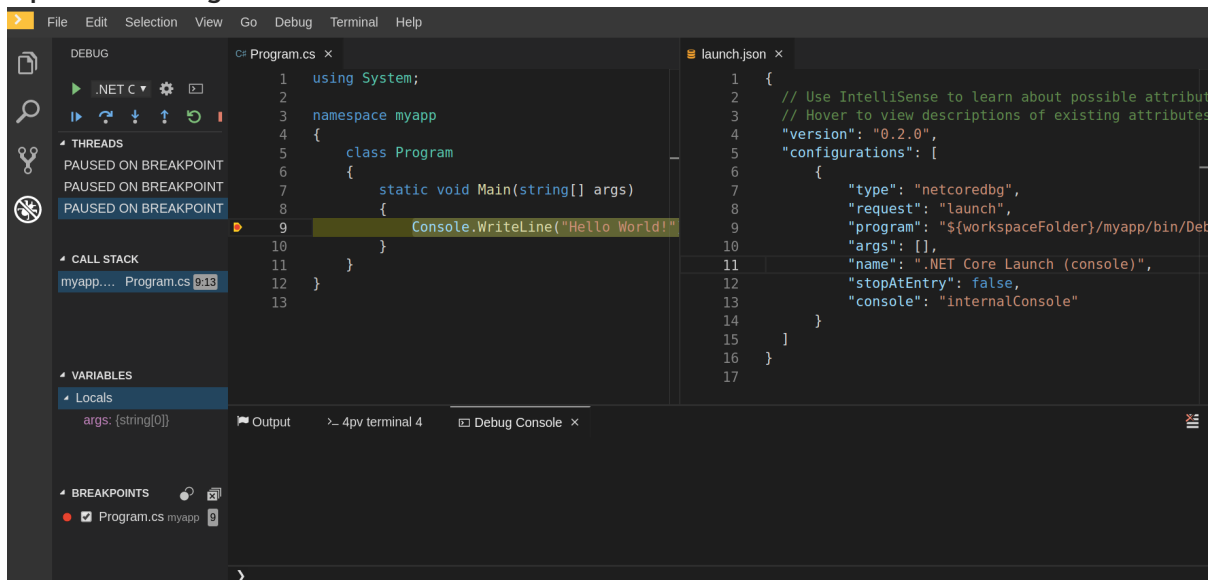
2. Test the autocompletion feature within the braces of the `configuration` section of the `launch.json` file. If you can find `netcoredbg`, the Che-Theia plug-in is correctly initialized. If not, see [Section 8.1, "Viewing OpenShift Dev Spaces workspaces logs"](#)

### 8.1.1.14.3. 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.



### 8.1.1.15. Viewing logs for Camel

#### 8.1.1.15.1. 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:

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

3. If you cannot find it, see [Section 8.1, “Viewing OpenShift Dev Spaces workspaces logs”](#)

### 8.1.1.15.2. 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`.

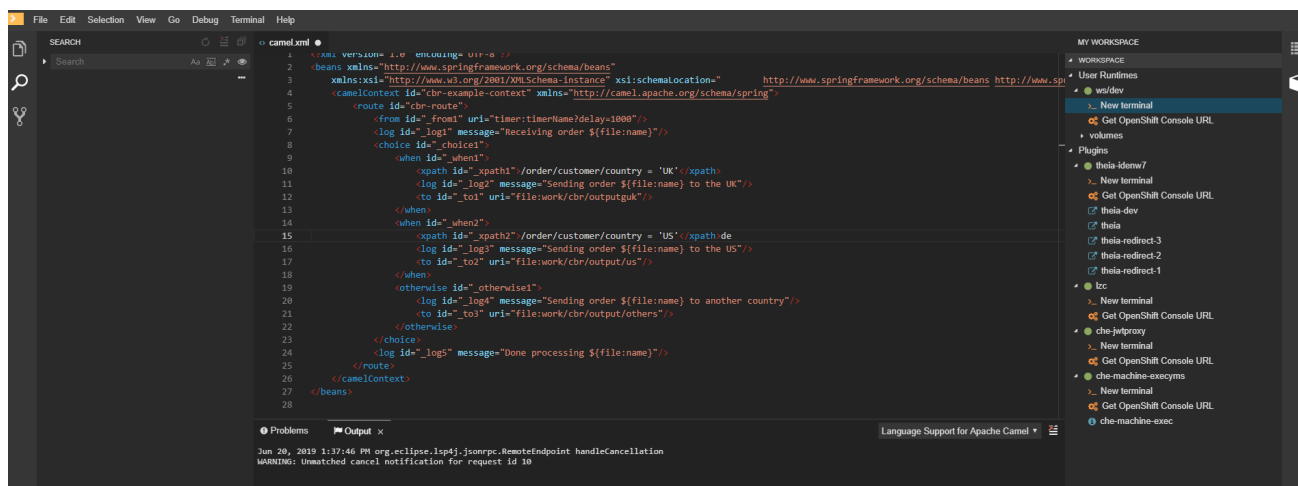
#### Procedure

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.



## 8.1.2. Viewing Che-Theia IDE logs

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

### 8.1.2.1. 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

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

## Procedure

1. Obtain the list of the available Pods:

```
$ oc get pods
```

### Example

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

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

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

### Example:

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

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

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

### Example:

```
$ oc logs --follow workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -container
theia-idexzb
>root INFO unzipping the plug-in 'task_plugin.theia' to directory: /tmp/theia-
unpacked/task_plugin.theia
root INFO unzipping the plug-in 'theia_yeoman_plugin.theia' to directory: /tmp/theia-
unpacked/theia_yeoman_plugin.theia
root WARN A handler with prefix term is already registered.
root INFO [nsfw-watcher: 75] Started watching: /home/theia/.theia
root WARN e.onStart is slow, took: 367.4600000013015 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [nsfw-watcher: 75] Started watching: /projects/.theia/tasks.json
root INFO [4f9590c5-e1c5-40d1-b9f8-ec31ec3bdac5] Sync of 9 plugins took:
62.260000000242493 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [hosted-plugin: 88] PLUGIN_HOST(88) starting instance
```

## 8.2. 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.

### 8.2.1. Restarting a OpenShift Dev Spaces workspace in Verbose mode after start failure

This section describes how to restart a OpenShift Dev Spaces 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

- A running instance of OpenShift Dev Spaces.
- 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 modelink.
3. Check the Logs tab to find a reason for the workspace failure.

### 8.2.2. Starting a OpenShift Dev Spaces workspace in Verbose mode

This section describes how to start the Red Hat OpenShift Dev Spaces workspace in Verbose mode.

#### Prerequisites

- A running instance of Red Hat OpenShift Dev Spaces.
- An existing workspace defined on this instance of OpenShift Dev Spaces.

#### 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 modeoption. 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.

## 8.3. 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.

### 8.3.1. 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\\_openshift\\_dev\\_spaces/3.0/html-single/administration\\_guide/index#caching-images-for-faster-workspace-start](https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#caching-images-for-faster-workspace-start).

#### 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.

#### Installing offline

*Role: Administrator*

Components of OpenShift Dev Spaces are OCI images. Set up Red Hat OpenShift Dev Spaces 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\\_openshift\\_dev\\_spaces/3.0/html-single/administration\\_guide/index#installing-devspaces-in-a-restricted-environment-on-openshift](https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#installing-devspaces-in-a-restricted-environment-on-openshift).

#### 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\\_openshift\\_dev\\_spaces/3.0/html-single/administration\\_guide/index#caching-images-for-faster-workspace-start](https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#caching-images-for-faster-workspace-start).

#### 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.

## 8.3.2. 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/plugin
  cpuLimit: 1360Mi 1
  cpuRequest: 100m 2
```

- 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"
    memoryLimit: "512Mi" 1
  extensions:
  - https://link.to/vsix
```

- 1 Specifies the memory limit for the plug-in.

In the devfile definition (`devfile.yaml`):

```
apiVersion: 1.0.0
```

**components:**

-  
type: chePlugin  
id: id/of/plug-in  
memoryLimit: 1048M **1**  
memoryRequest: 256M

- 1** Specifies the memory limit for this plug-in.

## 8.4. TROUBLESHOOTING NETWORK PROBLEMS

This section describes how to prevent or resolve issues related to network policies. OpenShift Dev Spaces 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 9. ADDING A VISUAL STUDIO CODE EXTENSION TO A WORKSPACE

Previously, with the devfiles v1 format, you used the devfile to specify IDE-specific plug-ins and Visual Studio Code extensions. Now, with devfiles v2, you use a specific meta-folder rather than the devfile to specify the plug-ins and extensions.

### 9.1. OPENSIFT DEV SPACES PLUG-IN REGISTRIES OVERVIEW

Every OpenShift Dev Spaces instance has a registry of default plug-ins and extensions. The Che-Theia IDE gets information about these plug-ins and extensions from the registry and installs them.

Check this OpenShift Dev Spaces [registry project](#) for an overview of the default plug-ins, extensions, and source code. An online instance that refreshes after every commit to the main branch, is located [here](#). You can use a different plug-in or extension registry for Che-Theia if you don't work in air-gapped environment: only the default registry is available there.

The plug-in and extension overview for Che-Code Visual Studio Code editor is located in the [OpenVSX instance](#). Air gap is not yet supported for this editor.

### 9.2. ADDING AN EXTENSION TO .VSCODE/EXTENSIONS.JSON

The easiest way to add a Visual Studio Code extension to a workspace is to add it to the `.vscode/extensions.json` file. The main advantage of this method is that it works with all supported OpenShift Dev Spaces IDEs.

If you use the Che-Theia IDE, the extension is installed and configured automatically. If you use a different supported IDE with the Che-Code Visual Studio Code fork, the IDE displays a pop-up with a recommendation to install the extension.

#### Prerequisites

1. You have the `.vscode/extensions.json` file in the root of the GitHub repository.

#### Procedure

1. Add the extension ID to the `.vscode/extensions.json` file. Use a `.` sign to separate the publisher and extension. The following example uses the IDs of Red Hat Visual Studio Code Java extension:

```
{
  "recommendations": [
    "redhat.java"
  ]
}
```



#### NOTE

If the specified set of extension IDs isn't available in the OpenShift Dev Spaces registry, the workspace starts without the extension.

### 9.3. ADDING PLUG-IN PARAMETERS TO .CHE/CHE-THEIA-PLUGINS.YAML

You can add extra parameters to a plug-in by modifying the `.che/che-theia-plugins.yaml` file. These modifications include:

- Defining the plug-ins for workspace installation.
- Changing the default memory limit.
- Overriding default preferences.

### 9.3.1. Defining the plug-ins for workspace installation

Define the plug-ins to be installed in the workspace.

#### Prerequisites

1. You have the `.che/che-theia-plugins.yaml` file in the root of the GitHub repository.

#### Procedure

1. Add the ID of the plug-in to the `.che/che-theia-plugins.yaml` file. Use the `/` sign to separate the publisher and plug-in name. The following example uses the IDs of Red Hat Visual Studio Code Java extension:

```
- id: redhat/java/latest
```

### 9.3.2. Changing the default memory limit

Override container settings such as the memory limit.

#### Prerequisites

1. You have the `.che/che-theia-plugins.yaml` file in the root of the GitHub repository.

#### Procedure

1. Add an **override** section to the `.che/che-theia-plugins.yaml` file under the `theid` of the plug-in.
2. Specify the memory limit for the extension. In the following example, OpenShift Dev Spaces automatically installs the Red Hat Visual Studio Code Java extension in the OpenShift Dev Spaces workspace and increases the memory of the workspace by two gigabytes:

```
- id: redhat/java/latest
  override:
    sidecar:
      memoryLimit: 2Gi
```

### 9.3.3. Overriding default preferences

Override the default preferences of the Visual Studio Code extension for the workspace.

#### Prerequisites



1. You have the `.che/che-theia-plugins.yaml` file in the root of the GitHub repository.

#### Procedure

1. Add an **override** section to the `.che/che-theia-plugins.yaml` file under the `theid` of the extension.
2. Specify the preferences in the **Preferences** section. In the following example, OpenShift Dev Spaces automatically installs Red Hat Visual Studio Code Java extension in the OpenShift Dev Spaces workspace and sets the `java.server.launchMode` preference to **LightWeight**:

```
- id: redhat/java/latest
  override:
    preferences:
      java.server.launchMode: LightWeight
```

#### NOTE

You can also define the preferences in the `.vscode/settings.json` file, either by changing the preferences in the UI of your IDE or by adding them to the `.vscode/settings.json` file:

```
{
  "my.preferences": "my-value"
}
```

## 9.4. DEFINING VISUAL STUDIO CODE EXTENSION ATTRIBUTES IN THE DEVFILE

If it's not possible to add extra files in the GitHub repository, you can define some of the plug-in or extension attributes by inlining them in the devfile. You can use this procedure with both `.vscode/extensions.json` and `.che/che-theia-plugins.yaml` file contents.

### 9.4.1. Inlining `.vscode/extensions.json` file

Use `.vscode/extensions.json` file contents to inline the extension attributes in the devfile.

#### Procedure

1. Add an **attributes** section to your `devfile.yaml` file.
2. Add `.vscode/extensions.json` to the `attributes` section. Add a `|` sign after the colon separator.
3. Paste the contents of the `.vscode/extensions.json` file after the `|` sign. The following example uses Red Hat Visual Studio Code Java extension attributes:

```
schemaVersion: 2.2.0
metadata:
  name: my-example
attributes:
  .vscode/extensions.json: |
```

```
{  
  "recommendations": [  
    "redhat.java"  
  ]  
}
```

### 9.4.2. Inlining `.che/che-theia-plugins.yaml` file

Use `.che/che-theia-plugins.yaml` file contents to inline the plug-in attributes in the devfile.

#### Procedure

1. Add an **attributes** section to your `devfile.yaml` file.
2. Add `.vscode/extensions.json` to the **attributes** section. Add a `|` sign after the colon separator.
3. Paste the content of the `.che/che-theia-plugins.yaml` file after the `|` sign. The following example uses Red Hat Visual Studio Code Java extension attributes:

```
schemaVersion: 2.2.0  
metadata:  
  name: my-example  
attributes:  
  .che/che-theia-plugins.yaml: |  
    - id: redhat/java/latest
```