



Red Hat OpenShift Dev Spaces 3.4

User guide

Using Red Hat OpenShift Dev Spaces 3.4

Red Hat OpenShift Dev Spaces 3.4 User guide

Using Red Hat OpenShift Dev Spaces 3.4

Robert Kratky
rkratky@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Jana Vrbkova
jvrbkova@redhat.com

Max Leonov
mleonov@redhat.com

Legal Notice

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Information for users using Red Hat OpenShift Dev Spaces.

Table of Contents

CHAPTER 1. ADOPTING DEV SPACES	4
1.1. DEVELOPER WORKSPACES	4
1.1.1. Microsoft Visual Studio Code - Open Source	4
1.2. STACK SAMPLES	5
1.3. BADGE FOR FIRST-TIME CONTRIBUTORS	7
1.4. REVIEWING PULL AND MERGE REQUESTS	7
CHAPTER 2. USER ONBOARDING	9
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	13
2.2.4. URL parameter for the devfile file name	13
2.2.5. URL parameter for the devfile file path	13
2.2.6. URL parameter for the workspace storage	14
2.2.7. URL parameter for additional remotes	14
2.3. BASIC ACTIONS YOU CAN PERFORM ON A WORKSPACE	15
2.4. AUTHENTICATING TO A GIT SERVER FROM A WORKSPACE	15
CHAPTER 3. CUSTOMIZING WORKSPACE COMPONENTS	16
CHAPTER 4. INTRODUCTION TO DEVFILE IN DEV SPACES	17
CHAPTER 5. SELECTING A WORKSPACE IDE	18
5.1. SELECTING AN IN-BROWSER IDE FOR ONE NEW WORKSPACE	18
5.2. SELECTING AN IN-BROWSER IDE FOR ALL WORKSPACES THAT CLONE THE SAME GIT REPOSITORY	19
5.2.1. Setting up che-editor.yaml	19
5.2.2. Parameters for che-editor.yaml	19
CHAPTER 6. USING CREDENTIALS AND CONFIGURATIONS IN WORKSPACES	21
6.1. MOUNTING SECRETS	21
6.1.1. Creating image pull Secrets	23
6.1.1.1. Creating an image pull Secret with oc	23
6.1.1.2. Creating an image pull Secret from a .dockercfg file	23
6.1.1.3. Creating an image pull Secret from a config.json file	24
6.1.2. Using a Git-provider access token	25
6.2. MOUNTING CONFIGMAPS	27
6.3. ENABLING ARTIFACT REPOSITORIES IN A RESTRICTED ENVIRONMENT	28
6.3.1. Maven	29
6.3.2. Gradle	31
6.3.3. npm	33
6.3.4. Python	34
6.3.5. Go	35
6.3.6. NuGet	36
CHAPTER 7. REQUESTING PERSISTENT STORAGE FOR WORKSPACES	38
7.1. REQUESTING PERSISTENT STORAGE IN A DEVFILE	38
7.2. REQUESTING PERSISTENT STORAGE IN A PVC	39
CHAPTER 8. INTEGRATING WITH OPENSIFT	41
8.1. MANAGING WORKSPACES WITH OPENSIFT APIS	41

8.1.1. Listing all workspaces	41
8.1.2. Creating workspaces	42
8.1.3. Stopping workspaces	45
8.1.4. Starting stopped workspaces	46
8.1.5. Removing workspaces	46
8.2. AUTOMATIC OPENSIFT TOKEN INJECTION	47
8.3. NAVIGATING DEV SPACES FROM OPENSIFT DEVELOPER PERSPECTIVE	48
8.3.1. OpenShift Developer Perspective integration with OpenShift Dev Spaces	49
8.3.2. Editing the code of applications running in OpenShift Container Platform using OpenShift Dev Spaces	49
8.3.3. Accessing OpenShift Dev Spaces from Red Hat Applications menu	50
8.4. NAVIGATING OPENSIFT WEB CONSOLE FROM DEV SPACES	50
CHAPTER 9. TROUBLESHOOTING DEV SPACES	52
9.1. VIEWING DEV SPACES WORKSPACES LOGS	52
9.1.1. Workspace logs in CLI	52
9.1.2. Workspace logs in OpenShift console	53
9.1.3. Language servers and debug adapters logs in the editor	53
9.2. TROUBLESHOOTING WORKSPACE START FAILURES	53
9.2.1. Restarting a OpenShift Dev Spaces workspace in Verbose mode after start failure	53
9.2.2. Starting a OpenShift Dev Spaces workspace in Verbose mode	54
9.3. TROUBLESHOOTING SLOW WORKSPACES	54
9.3.1. Improving workspace start time	54
9.3.2. Improving workspace runtime performance	55
9.4. TROUBLESHOOTING NETWORK PROBLEMS	57

CHAPTER 1. ADOPTING DEV SPACES

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

- [Section 1.1, “Developer workspaces”](#)
- [Section 1.3, “Badge for first-time contributors”](#)
- [Section 1.4, “Reviewing pull and merge requests”](#)
- [Section 1.2, “Stack samples”](#)

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.

1.1.1. Microsoft Visual Studio Code - Open Source

Microsoft Visual Studio Code - Open Source is the default browser-based IDE.

OpenShift Dev Spaces adds these features:

Open VSX registry

The IDE uses the [Open VSX registry](#) to list and download extensions. The OpenShift Dev Spaces administrator can [configure the Open VSX registry URL](#).

Recommended extensions

The IDE installs automatically the [recommended extensions](#).

OpenShift Dev Spaces adds these extensions:

Commands

Translates Devfile commands to Microsoft Visual Studio Code - Open Source tasks.

Procedure

- To see the drop-down list of available tasks, type: **F1 Tasks: Run Task Enter che**.

Activity tracker

Tracks events provided by the Microsoft Visual Studio Code - Open Source to determine and stop inactive workspaces. This extension does not save, collect, or store data.

API

Provides helpers to interact with Dev Workspace and OpenShift Dev Spaces.

GitHub authentication

Provides support for authenticating to GitHub. It registers the **github** Authentication Provider that can be leveraged by other extensions. This also provides the GitHub authentication used by Settings Sync.

Port

Detects opening ports and provides redirect URI. When a process starts listening to a port, OpenShift Dev Spaces displays a notification with a link to open the resulting resource.

Procedure

- To display the endpoint list, type: **F1 Explorer: Focus on endpoints View Enter**.

Remote

Provides commands for the remote authority.

Resource monitor

Monitors resources such as CPU and RAM.

Telemetry

Detects and sends the following events to a backend telemetry plugin listening on **http://localhost:\${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}**:

WORKSPACE_OPENED

Sent when the telemetry extension activates

EDITOR_USED

Sent on the **vscode.workspace.onDidChangeTextDocument** event

Terminal

Opens a terminal to a Dev Workspace container.

1.2. STACK SAMPLES

To demonstrate the capabilities of Red Hat OpenShift Dev Spaces as remote development environment, Red Hat OpenShift Dev Spaces contains stack samples using various programming languages. Each sample includes a devfile and you can use them as a reference to bootstrap a new project. You can customize the samples if you are a OpenShift Dev Spaces administrator.

Table 1.1. Supported languages

Language	Builders, runtimes, and databases	Maturity
Apache Camel K	<ul style="list-style-type: none"> • Red Hat Fuse 	GA

Language	Builders, runtimes, and databases	Maturity
Java	<ul style="list-style-type: none"> ● OpenJDK 11 ● Maven 3.6 ● Gradle 6.1 ● Quarkus Tools ● Lombok 1.18 ● JBoss EAP 7.4 ● JBoss EAP XP 3.0 	GA
Node.js	<ul style="list-style-type: none"> ● Node.js 16 ● NPM 8 ● Express ● MongoDB 3.6 	GA
Python	<ul style="list-style-type: none"> ● Python 3.8 ● Pip 22.2 	GA
C/C++	<ul style="list-style-type: none"> ● GCC ● cmake ● make 	Technology preview
C#	<ul style="list-style-type: none"> ● Dotnet 3.1 on AMD64 and Intel 64 (x86_64) ● Dotnet 6.0 on AMD64 and Intel 64 (x86_64), and IBM Z (s390x) 	Technology preview
Go	<ul style="list-style-type: none"> ● Golang 	Technology preview
PHP	<ul style="list-style-type: none"> ● CakePHP ● Composer 	Technology preview

1.3. BADGE FOR FIRST-TIME CONTRIBUTORS

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 ("[">https://devspaces-*<openshift_deployment_name>*.*<domain_name>*](https://devspaces-<i><openshift_deployment_name></i>.<i><domain_name></i;)") and repository URL (*<your_repository_url>*), and add the link to your repository in the project **README.md** file.

```
[[Contribute](https://www.eclipse.org/che/contribute.svg)]
(/https://<your\_repository\_url>">https://devspaces-<openshift\_deployment\_name>.<domain\_name>"/#https://<your\_repository\_url>)
```

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.

1.4. REVIEWING PULL AND MERGE REQUESTS

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 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.

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 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>*”**.
- Optional: You have [authentication to the Git server](#) configured.
- 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 Universal Developer Image and with Microsoft Visual Studio Code - Open Source as the workspace IDE.

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"](#)
- [Section 2.2.6, "URL parameter for the workspace storage"](#)
- [Section 2.2.7, "URL parameter for additional remotes"](#)

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-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;/#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-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;" 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 in-browser IDE, which is Microsoft Visual Studio Code - Open Source.

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

```
"https://devspaces-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;/#<git_repository_url>?che-editor=<editor_key>"
```



NOTE

The workspace IDE might be already set for a remote Git repository in the **che-editor.yaml** file of the repository.

Table 2.1. The URL parameter <editor_key> values for supported IDEs

IDE	<editor_key> value	Note
Microsoft Visual Studio Code - Open Source	che-incubator/che-code/insiders	This is the default IDE that loads in a new workspace when the URL parameter or che-editor.yaml is not used.
JetBrains IntelliJ IDEA Community Edition	che-incubator/che-idea/latest	Technology Preview .

IDE	<code><editor_key></code> value	Note
Eclipse Theia	<code>eclipse/che-theia/latest</code>	Deprecated and will be removed in a future release.

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 might 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-&lt;openshift_deployment_name&gt;;&lt;domain_name&gt;;"#<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 might 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-&lt;openshift_deployment_name&gt;;&lt;domain_name&gt;;"#<git_repository_url>?
df=<filename>.yaml ❶
```

❶ **`<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 might 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-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;/#<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.

2.2.6. URL parameter for the workspace storage

If the URL for starting a new workspace does not contain a URL parameter specifying the storage type, the new workspace is created in ephemeral or persistent storage, whichever is defined as the default storage type in the **CheCluster** Custom Resource.

The URL parameter for specifying a storage type for a workspace is **storageType=<storage_type>**:

```
"https://devspaces-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;/#<git_repository_url>?
storageType=<storage_type> 1
```

1 Possible **<storage_type>** values:

- **ephemeral**
- **per-user** (persistent)
- **per-workspace** (persistent)

TIP

With the **ephemeral** or **per-workspace** storage type, you can run multiple workspaces concurrently, which is not possible with the default **per-user** storage type.

Additional resources

- [Chapter 7, Requesting persistent storage for workspaces](#)

2.2.7. URL parameter for additional remotes

When you visit a URL for starting a new workspace, OpenShift Dev Spaces configures the **origin** remote to be the Git repository that you specified with **#** after the FQDN URL of your organization's OpenShift Dev Spaces instance.

The URL parameter for cloning and configuring additional remotes for the workspace is **remotes=**:

```
"https://devspaces-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;/#<git_repository_url>?
remotes={{<name_1>,<url_1>},{<name_2>,<url_2>},{<name_3>,<url_3>},...}
```



IMPORTANT

- If you do not enter the name **origin** for any of the additional remotes, the remote from `<git_repository_url>` will be cloned and named **origin** by default, and its expected branch will be checked out automatically.
- If you enter the name **origin** for one of the additional remotes, its default branch will be checked out automatically, but the remote from `<git_repository_url>` will NOT be cloned for the workspace.

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") 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
<i>Reopen a running workspace</i>	Click Open .
<i>Restart a running workspace</i>	Go to ⋮ > Restart Workspace .
<i>Stop a running workspace</i>	Go to ⋮ > Stop Workspace .
<i>Start a stopped workspace</i>	Click Open .
<i>Delete a workspace</i>	Go to ⋮ > Delete Workspace .

2.4. AUTHENTICATING 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.

User authentication to a Git server from a workspace is configured by the administrator or, in some cases, by the individual user:

- Your administrator sets up an [OAuth application on GitHub, GitLab, or Bitbucket](#) for your organization's Red Hat OpenShift Dev Spaces instance.
- As a workaround, some users create and apply their own Kubernetes Secrets for their personal [Git-provider access tokens](#).

Additional resources

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

CHAPTER 3. CUSTOMIZING WORKSPACE COMPONENTS

To customize workspace components:

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

CHAPTER 4. INTRODUCTION TO DEVFILE IN DEV SPACES

[Devfiles](#) are **yaml** text files used for development environment customization. Use them to configure a devfile to suit your specific needs and share the customized devfile across multiple workspaces to ensure identical user experience and build, run, and deploy behaviours across your team.

Devfile and Universal Developer Image

You do not need a devfile to start a workspace. If you do not include a devfile in your project repository, Red Hat OpenShift Dev Spaces automatically loads a default devfile with a Universal Developer Image (UDI).

OpenShift Dev Spaces devfile registry

[OpenShift Dev Spaces devfile registry](#) contains ready-to-use devfiles for different languages and technologies.



NOTE

Devfiles included in the registry are specific to Red Hat OpenShift Dev Spaces and should be treated as samples rather than templates. They might require updates to work with other versions of the components featured in the samples.

Additional resources

- [What is a devfile](#)
- [Benefits of devfile](#)
- [Devfile customization overview](#)

CHAPTER 5. SELECTING A WORKSPACE IDE

The default in-browser IDE in a new workspace is Microsoft Visual Studio Code - Open Source.



NOTE

Because the OpenShift Dev Spaces build of Microsoft Visual Studio Code - Open Source supports custom branding, your organization might be using a branded build.

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 5.1, “Selecting an in-browser IDE for one new workspace”](#).
- 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 5.2, “Selecting an in-browser IDE for all workspaces that clone the same Git repository”](#).

Table 5.1. Supported in-browser IDEs

IDE	id	Note
Microsoft Visual Studio Code - Open Source	che-incubator/che-code/insiders	This is the default IDE that loads in a new workspace when the URL parameter or che-editor.yaml is not used.
JetBrains IntelliJ IDEA Community Edition	che-incubator/che-idea/latest	Technology Preview .
Eclipse Theia	eclipse/che-theia/latest	Deprecated and will be removed in a future release.

5.1. SELECTING AN IN-BROWSER IDE FOR ONE NEW WORKSPACE

You can select your preferred in-browser IDE when using a URL for starting a new workspace. This way, each developer using OpenShift Dev Spaces can start a workspace with a clone of the same project repository and the personal choice of the in-browser IDE.

Procedure

1. Include the [Section 2.2.2, “URL parameter for the workspace IDE”](#) in the [URL for starting a new workspace](#).
2. Visit the URL in the browser.

Verification

- Verify that the selected in-browser IDE loads in the browser tab of the started workspace.

5.2. SELECTING AN IN-BROWSER IDE FOR ALL WORKSPACES THAT CLONE THE SAME GIT REPOSITORY

5.2.1. Setting up che-editor.yaml

To define the same in-browser IDE for all workspaces that will clone the same remote Git repository of your project, you can use the **che-editor.yaml** file.

This way, you can set a common default editor for your team and provide new contributors with the most suitable editor for your project. You can also use the **che-editor.yaml** file when you need to set a different IDE default for a particular project repository rather than the default IDE of your organization's OpenShift Dev Spaces instance.

Procedure

- In the remote Git repository of your project, create a `/.che/che-editor.yaml` file with lines that specify the relevant parameter, as described in the next section.

Verification

1. Visit the [URL for starting a new workspace](#) .
2. Verify that the selected in-browser IDE loads in the browser tab of the started workspace.

5.2.2. Parameters for che-editor.yaml

The simplest way to select an IDE in the **che-editor.yaml** is to specify the **id** of an IDE that is available in the table of supported in-browser IDEs in [Chapter 5, Selecting a workspace IDE](#):

Example 5.1. id selects an IDE from the plugin registry

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

As alternatives to providing the **id** parameter, the **che-editor.yaml** file supports a **reference** to the URL of another **che-editor.yaml** file or an **inline** definition for an IDE outside of a plugin registry:

Example 5.2. reference points to a remote che-editor.yaml file

```
reference: https://<hostname_and_path_to_a_remote_file>/che-editor.yaml
```

Example 5.3. inline specifies a complete definition for a customized IDE without a plugin registry

```
inline:
  schemaVersion: 2.1.0
  metadata:
    name: JetBrains IntelliJ IDEA Community IDE
  components:
    - name: intellij
      container:
```

```

image: 'quay.io/che-incubator/che-idea:next'
volumeMounts:
  - name: projector-user
    path: /home/projector-user
mountSources: true
memoryLimit: 2048M
memoryRequest: 32Mi
cpuLimit: 1500m
cpuRequest: 100m
endpoints:
  - name: intellij
    attributes:
      type: main
      cookiesAuthEnabled: true
      urlRewriteSupported: true
      discoverable: false
      path: /?backgroundColor=434343&wss
      targetPort: 8887
      exposure: public
      secure: false
      protocol: https
    attributes: {}
  - name: projector-user
    volume: {}

```

For more complex scenarios, the **che-editor.yaml** file supports the **registryUrl** and **override** parameters:

Example 5.4. **registryUrl** points to a custom plugin registry rather than to the default OpenShift Dev Spaces plugin registry

```

id: <editor_id> 1
registryUrl: <url_of_custom_plugin_registry>

```

1 The **id** of the IDE in the custom plugin registry.

Example 5.5. **override** of the default value of one or more defined properties of the IDE

```

... 1
override:
  containers:
    - name: che-idea
      memoryLimit: 1280Mi
      cpuLimit: 1510m
      cpuRequest: 102m
  ...

```

1 **id**, **registryUrl**, or **reference**.

CHAPTER 6. 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 **Dev Workspace** containers in the OpenShift cluster of your organization's OpenShift Dev Spaces instance:

- Mount your credentials and sensitive configurations as Kubernetes [Secrets](#).
- Mount your non-sensitive configurations as Kubernetes [ConfigMaps](#).

If you need to allow the **Dev Workspace** Pods in the cluster to access container registries that require authentication, create an [image pull Secret](#) for the **Dev Workspace** 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 [user-specific settings.xml](#) file
- SSH key pairs
- [Git-provider access tokens](#)
- AWS authorization tokens
- Configuration files
- Persistent storage

Additional resources

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

6.1. 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 **Dev Workspace** 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](#).

- In your user project, you created a new Secret or determined an existing Secret to mount to all **Dev Workspace** containers.

Procedure

1. Add the labels, which are required for mounting the Secret, to the Secret.

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

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

Table 6.1. Optional annotations

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

Example 6.1. 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 **Dev Workspace** 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
```

6.1.1. Creating image pull Secrets

To allow the **Dev Workspace** 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.

6.1.1.1. 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
```

6.1.1.2. 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
```

6.1.1.3. 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
```

6.1.2. Using a Git-provider access token

OAuth for GitHub, GitLab, or Bitbucket needs to be [configured by the administrator](#) of your organization's OpenShift Dev Spaces instance. If your administrator could not configure it for OpenShift Dev Spaces users, the workaround is for you to apply your personal access token as a Kubernetes Secret.

Mounting your access token as a Secret enables the OpenShift Dev Spaces Server to access the remote repository that is cloned during workspace creation, including access to the repository's `/.che` and `/.vscode` folders.

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

After applying the Secret, you can create new workspaces from a private GitHub, GitLab, or Bitbucket-server repository.

You can create and apply multiple access-token Secrets per a Git provider. You must apply each of those Secrets in your user project.

Prerequisites

- You have cluster administrator permissions for the cluster on which your organization's OpenShift Dev Spaces instance is running.
- You have logged in to the cluster.

TIP

On OpenShift, you can use the `oc` command-line tool to log in to the cluster:

```
$ oc login
"https://devspaces-<openshift_deployment_name>.<domain_name>" --
username=<my_user>
```

Procedure

1. Generate your access token on your Git provider's website.
2. Encode your access token to Base64.

TIP

If you have the `base64` command-line tools installed in the operating system, you can use the command line:

```
$ echo -n '<your_access_token_string>' | base64
```

3. Visit


```
"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/user
```

in the web browser and copy the **id** value from the response. This is your OpenShift Dev Spaces user ID.

4. Get your Git provider user ID by following the Git provider's API documentation:

- GitHub: [Get a user](#). See the **id** value in the response.
- GitLab: [List users: For normal users](#), use the **username** filter: `/users?username=:username`. See the **id** value in the response.
- Bitbucket Server: [Get users](#). See the **id** value in the response.

5. Prepare a new OpenShift Secret.

```
kind: Secret
apiVersion: v1
metadata:
  name: personal-access-token-<your_choice_of_name_for_this_token>
  labels:
    app.kubernetes.io/component: scm-personal-access-token
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/che-userid: <devspaces_user_id> 1
    che.eclipse.org/scm-personal-access-token-name: <git_provider_name> 2
    che.eclipse.org/scm-url: <git_provider_endpoint> 3
    che.eclipse.org/scm-userid: '<git_provider_user_id>' 4
    che.eclipse.org/scm-username: <git_provider_username>
data:
  token: <Base64_encoded_access_token>
type: Opaque
```

- 1** Your OpenShift Dev Spaces user ID.
- 2** The Git provider name: **github** or **gitlab** or **bitbucket-server**.
- 3** The Git provider URL.
- 4** Your Git provider user ID.

6. Visit

"https://devspaces-*<openshift_deployment_name>*; *<domain_name>*"/api/kubernetes/namespace to get your OpenShift Dev Spaces user namespace as **name**.

7. Switch to your OpenShift Dev Spaces user namespace in the cluster.

TIP

On OpenShift:

- The **oc** command-line tool can return the namespace you are currently on in the cluster, which you can use to check your current namespace:

```
$ oc project
```

- You can switch to your OpenShift Dev Spaces user namespace on a command line if needed:

```
$ oc project <your_user_namespace>
```

8. Apply the Secret.

TIP

On OpenShift, you can use the **oc** command-line tool:

```
$ oc apply -f - <<EOF
<Secret_prepared_in_step_5>
EOF
```

Verification

1. [Start a new workspace by using the URL of a remote Git repository](#) that the Git provider hosts.
2. Make some changes and push to the remote Git repository from the workspace.

6.2. 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 **Dev Workspace** 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](#).
- In your user project, you created a new ConfigMap or determined an existing ConfigMap to mount to all **Dev Workspace** containers.

Procedure

1. Add the labels, which are required for mounting the ConfigMap, to the ConfigMap.

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

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

Table 6.2. Optional annotations

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

Example 6.2. 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 **Dev Workspace** containers.

6.3. 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](#)

6.3.1. Maven

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

Prerequisites

- You are not running any Maven workspace.
- You know your user namespace, which is **<username>-devspaces** where **<username>** is your OpenShift Dev Spaces username.

Procedure

1. In the **<username>-devspaces** namespace, 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. In the **<username>-devspaces** namespace, 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>

```

- Optional: When using EAP-based devfiles, apply a second **settings.xml** ConfigMap in the **<username>-devspaces** namespace, and with the same content, a different name, and the **/home/jboss/.m2** mount path.
- In the **<username>-devspaces** namespace, apply the ConfigMap for the TrustStore initialization script:

Java 8

```

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-java8-truststore.sh: |
  #!/usr/bin/env bash

  keytool -importcert -noprompt -file /home/user/certs/tls.cer -trustcacerts -keystore
  ~/.java/current/jre/lib/security/cacerts -storepass changeit

```

Java 11

```

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-java11-truststore.sh: |
    #!/usr/bin/env bash

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

```

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

6.3.2. Gradle

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.

- Run `~/init-truststore.sh`.

6.3.3. npm

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

- 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:
```

```
NODE_EXTRA_CA_CERTS: /home/user/certs/tls.cer
NPM_CONFIG_REGISTRY: >-
  https://<npm_artifact_repository_route>/repository/npm-all/
```

6.3.4. Python

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

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:
  PIP_INDEX_URL: >-
    https://<python_artifact_repository_route>/repository/pypi-all/
  PIP_CERT: /home/user/certs/tls.cer

```

6.3.5. Go

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

```

6.3.6. NuGet

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

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 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>

```

CHAPTER 7. 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 **Dev Workspace** 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** might fail.

Additional resources

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

7.1. 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 7.1. 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
```

7.2. REQUESTING PERSISTENT STORAGE IN A PVC

You can 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 **Dev Workspace** 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 **Dev Workspace** 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 7.1. Optional annotations

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

Example 7.2. 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 8. INTEGRATING WITH OPENSIFT

- [Section 8.2, “Automatic OpenShift token injection”](#)
- [Section 8.3, “Navigating Dev Spaces from OpenShift Developer Perspective”](#)
- [Section 8.4, “Navigating OpenShift web console from Dev Spaces”](#)

8.1. MANAGING WORKSPACES WITH OPENSIFT APIS

On your organization’s OpenShift cluster, OpenShift Dev Spaces workspaces are represented as **DevWorkspace** custom resources of the same name. As a result, if there is a workspace named **my-workspace** in the OpenShift Dev Spaces dashboard, there is a corresponding **DevWorkspace** custom resource named **my-workspace** in the user’s project on the cluster.

Because each **DevWorkspace** custom resource on the cluster represents a OpenShift Dev Spaces workspace, you can manage OpenShift Dev Spaces workspaces by using OpenShift APIs with clients such as the command-line **oc**.

Each **DevWorkspace** custom resource contains details derived from the devfile of the Git repository cloned for the workspace. For example, a devfile might provide devfile commands and workspace container configurations.

8.1.1. Listing all workspaces

As a user, you can list your workspaces by using the command line.

Prerequisites

- An active **oc** session with permissions to **get** the **DevWorkspace** resources in your project on the cluster. See [Getting started with the CLI](#).
- You know the relevant OpenShift Dev Spaces user namespace on the cluster.

TIP

You can visit

"https://devspaces-*<openshift_deployment_name>*;.<i;domain_name>"/api/kuber netes/namespace to get your OpenShift Dev Spaces user namespace as **name**.

- You are in the OpenShift Dev Spaces user namespace on the cluster.

TIP

On OpenShift, you can use the command-line **oc** tool to [display your current namespace or switch to a namespace](#).

Procedure

- To list your workspaces, enter the following on a command line:

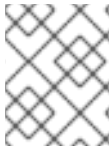
```
$ oc get devworkspaces
```

Example 8.1. Output

NAMESPACE	NAME	DEVWORKSPACE ID	PHASE	INFO
user1-dev	spring-petclinic	workspace6d99e9ffb9784491	Running	https://url-to-workspace.com
user1-dev	golang-example	workspacedf64e4a492cd4701	Stopped	Stopped
user1-dev	python-hello-world	workspace69c26884bbc141f2	Failed	Container tooling has state CrashLoopBackOff

TIP

You can view **PHASE** changes live by adding the **--watch** flag to this command.



NOTE

Users with administrative permissions on the cluster can list all workspaces from all OpenShift Dev Spaces users by including the **--all-namespaces** flag.

8.1.2. Creating workspaces

If your use case does not permit use of the OpenShift Dev Spaces dashboard, you can create workspaces with OpenShift APIs by applying custom resources to the cluster.



NOTE

Creating workspaces through the OpenShift Dev Spaces dashboard provides better user experience and configuration benefits compared to using the command line:

- As a user, you are automatically logged in to the cluster.
- OpenShift clients work automatically.
- OpenShift Dev Spaces and its components automatically convert the target Git repository's devfile into the **DevWorkspace** and **DevWorkspaceTemplate** custom resources on the cluster.
- Access to the workspace is secured by default with the **routingClass: che** in the **DevWorkspace** of the workspace.
- Recognition of the **DevWorkspaceOperatorConfig** configuration is managed by OpenShift Dev Spaces.
- Recognition of configurations in **spec.devEnvironments** specified in the **CheCluster** custom resource including:
 - Persistent storage strategy is specified with **devEnvironments.storage**.
 - Default IDE is specified with **devEnvironments.defaultEditor**.
 - Default plugins are specified with **devEnvironments.defaultPlugins**.
 - Container build configuration is specified with **devEnvironments.containerBuildConfiguration**.

Prerequisites

- An active **oc** session with permissions to create **DevWorkspace** resources in your project on the cluster. See [Getting started with the CLI](#).
- You know the relevant OpenShift Dev Spaces user namespace on the cluster.

TIP

You can visit

**"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kuber
netes/namespace** to get your OpenShift Dev Spaces user namespace as **name**.

- You are in the OpenShift Dev Spaces user namespace on the cluster.

TIP

On OpenShift, you can use the command-line **oc** tool to [display your current namespace or switch to a namespace](#).



NOTE

OpenShift Dev Spaces administrators who intend to create workspaces for other users must create the **DevWorkspace** custom resource in a user namespace that is provisioned by OpenShift Dev Spaces or by the administrator. See https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.4/html-single/administration_guide/index#administration-guide:configuring-namespace-provisioning.

Procedure

1. To prepare the **DevWorkspace** custom resource, copy the contents of the target Git repository's devfile.

Example 8.2. Copied devfile contents with `schemaVersion: 2.2.0`

```
components:
- name: tooling-container
  container:
    image: quay.io/devfile/universal-developer-image:ubi8-latest
```

TIP

For more details, see the [devfile v2 documentation](#).

2. Create a **DevWorkspace** custom resource, pasting the devfile contents from the previous step under the **spec.template** field.

Example 8.3. A **DevWorkspace** custom resource

```
kind: DevWorkspace
apiVersion: workspace.devfile.io/v1alpha2
```

```

metadata:
  name: my-devworkspace 1
  namespace: user1-dev 2
spec:
  routingClass: che
  started: true 3
  contributions: 4
    - name: ide
      uri:
        "https://devspaces-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;"/plugin-
        registry/v3/plugins/che-incubator/che-code/insiders/devfile.yaml
  template:
    projects: 5
      - name: my-project-name
        git:
          remotes:
            origin: https://github.com/eclipse-che/che-docs
    components: 6
      - name: tooling-container
        container:
          image: quay.io/devfile/universal-developer-image:ubi8-latest

```

- 1** Name of the **DevWorkspace** custom resource. This will be the name of the new workspace.
- 2** User namespace, which is the target project for the new workspace.
- 3** Determines whether the workspace must be started when the **DevWorkspace** custom resource is created.
- 4** URL reference to the [Microsoft Visual Studio Code - Open Source](#) IDE devfile from the plugin registry.
- 5** Details about the Git repository to clone into the workspace when it starts.
- 6** List of components such as workspace containers and volume components.

3. Apply the **DevWorkspace** custom resource to the cluster.

Verification

1. Verify that the workspace is starting by checking the **PHASE** status of the **DevWorkspace**.

```
$ oc get devworkspaces -n <user_project> --watch
```

Example 8.4. Output

```

NAMESPACE   NAME                DEVWORKSPACE ID           PHASE   INFO
user1-dev   my-devworkspace    workspacedf64e4a492cd4701 Starting Waiting
for workspace deployment

```


- When the workspace has successfully started, its **PHASE** status changes to **Running** in the output of the **oc get devworkspaces** command.

Example 8.5. Output

NAMESPACE	NAME	DEVWORKSPACE ID	PHASE	INFO
user1-dev	my-devworkspace	workspacedf64e4a492cd4701	Running	https://url-to-workspace.com

You can then open the workspace by using one of these options:

- Visit the URL provided in the **INFO** section of the output of the **oc get devworkspaces** command.
- Open the workspace from the OpenShift Dev Spaces dashboard.

8.1.3. Stopping workspaces

You can stop a workspace by setting the **spec.started** field in the **Devworkspace** custom resource to **false**.

Prerequisites

- An active **oc** session on the cluster. See [Getting started with the CLI](#).
- You know the workspace name.

TIP

You can find the relevant workspace name in the output of **\$ oc get devworkspaces**.

- You know the relevant OpenShift Dev Spaces user namespace on the cluster.

TIP

You can visit

"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kuber netes/namespace to get your OpenShift Dev Spaces user namespace as **name**.

- You are in the OpenShift Dev Spaces user namespace on the cluster.

TIP

On OpenShift, you can use the command-line **oc** tool to [display your current namespace or switch to a namespace](#).

Procedure

- Run the following command to stop a workspace:

```
$ oc patch devworkspace <workspace_name> \
-p '{"spec":{"started":false}}'
```

```
--type=merge -n <user_namespace> && \  
oc wait --for=jsonpath='{.status.phase}'=Stopped \  
dw/<workspace_name> -n <user_namespace>
```

8.1.4. Starting stopped workspaces

You can start a stopped workspace by setting the **spec.started** field in the **Devworkspace** custom resource to **true**.

Prerequisites

- An active **oc** session on the cluster. See [Getting started with the CLI](#).
- You know the workspace name.

TIP

You can find the relevant workspace name in the output of **\$ oc get devworkspaces**.

- You know the relevant OpenShift Dev Spaces user namespace on the cluster.

TIP

You can visit

"https://devspaces-*<openshift_deployment_name>*;*<domain_name>*"/api/kubernetes/namespaces to get your OpenShift Dev Spaces user namespace as **name**.

- You are in the OpenShift Dev Spaces user namespace on the cluster.

TIP

On OpenShift, you can use the command-line **oc** tool to [display your current namespace or switch to a namespace](#).

Procedure

- Run the following command to start a stopped workspace:

```
$ oc patch devworkspace <workspace_name> \  
-p '{"spec":{"started":true}}' \  
--type=merge -n <user_namespace> && \  
oc wait --for=jsonpath='{.status.phase}'=Running \  
dw/<workspace_name> -n <user_namespace>
```

8.1.5. Removing workspaces

You can remove a workspace by simply deleting the **DevWorkspace** custom resource.

**WARNING**

Deleting the **DevWorkspace** custom resource will also delete other workspace resources if they were created by OpenShift Dev Spaces: for example, the referenced **DevWorkspaceTemplate** and per-workspace **PersistentVolumeClaims**.

TIP

Remove workspaces by using the OpenShift Dev Spaces dashboard whenever possible.

Prerequisites

- An active **oc** session on the cluster. See [Getting started with the CLI](#).
- You know the workspace name.

TIP

You can find the relevant workspace name in the output of **\$ oc get devworkspaces**.

- You know the relevant OpenShift Dev Spaces user namespace on the cluster.

TIP

You can visit

"https://devspaces-*<openshift_deployment_name>*;*<domain_name>*"/api/kubernetes/*namespace* to get your OpenShift Dev Spaces user namespace as **name**.

- You are in the OpenShift Dev Spaces user namespace on the cluster.

TIP

On OpenShift, you can use the command-line **oc** tool to [display your current namespace or switch to a namespace](#).

Procedure

- Run the following command to remove a workspace:

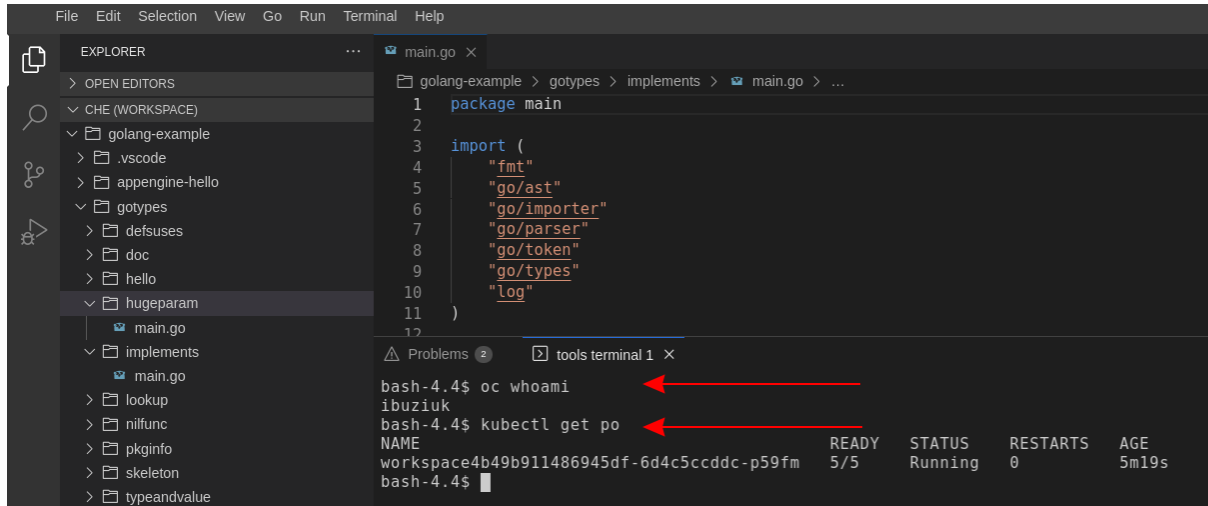
```
$ oc delete devworkspace <workspace_name> -n <user_namespace>
```

8.2. AUTOMATIC OPENSIFT 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
  Problems 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.

8.3. NAVIGATING DEV SPACES FROM OPENSIFT 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.

8.3.1. 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.

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

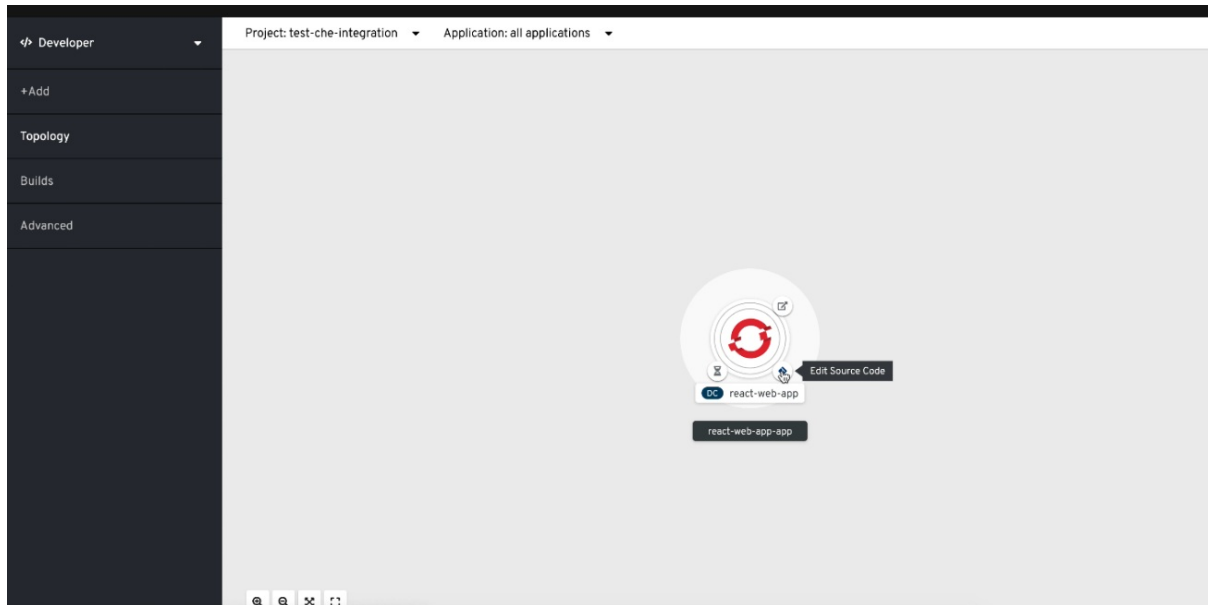
This section describes how to start editing the source code of applications running on OpenShift using OpenShift Dev Spaces.

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**.



- 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.

8.3.3. Accessing OpenShift Dev Spaces from Red Hat Applications menu

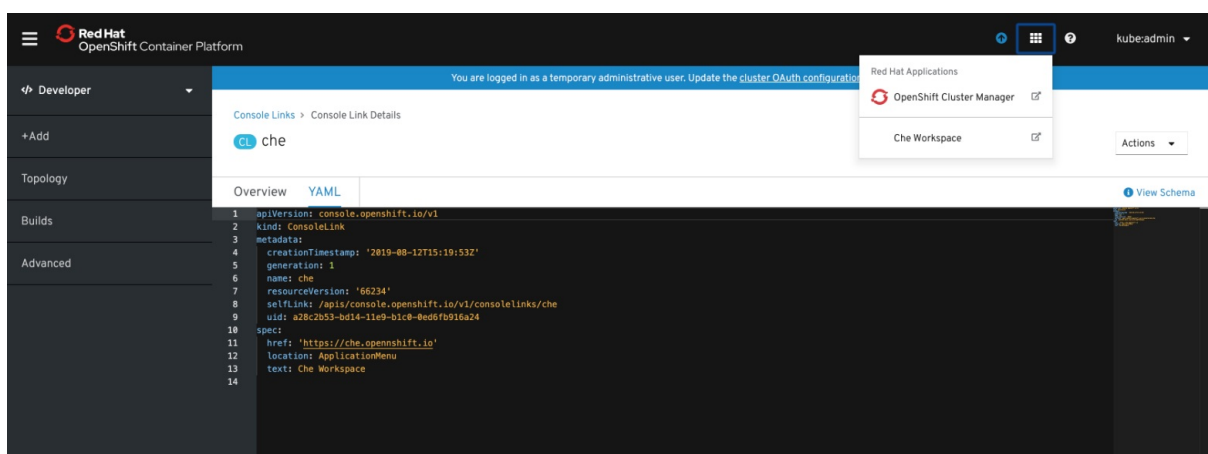
This section describes how to access OpenShift Dev Spaces workspaces from the **Red Hat Applications** menu on the OpenShift Container Platform.

Prerequisites

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

Procedure

- 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.



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

8.4. NAVIGATING OPENSHIFT WEB CONSOLE FROM DEV SPACES

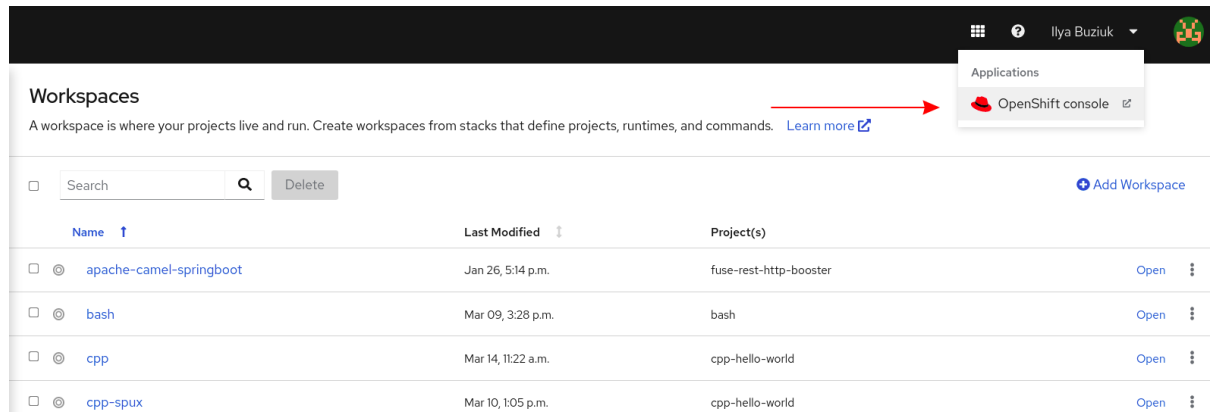
This section describes how to access OpenShift web console from OpenShift Dev Spaces.

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 9. TROUBLESHOOTING DEV SPACES

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

Additional resources

- [Section 9.1, “Viewing Dev Spaces workspaces logs”](#)
- [Section 9.2, “Troubleshooting workspace start failures”](#)
- [Section 9.3, “Troubleshooting slow workspaces”](#)
- [Section 9.4, “Troubleshooting network problems”](#)

9.1. VIEWING DEV SPACES WORKSPACES LOGS

You can view OpenShift Dev Spaces logs to better understand and debug background processes should a problem occur.

An IDE extension misbehaves or needs debugging

The logs list the plugins that have been loaded by the editor.

The container runs out of memory

The logs contain an **OOMKilled** error message. Processes running in the container attempted to request more memory than is configured to be available to the container.

A process runs out of memory

The logs contain an error message such as **OutOfMemoryException**. A process inside the container ran out of memory without the container noticing.

Additional resources

- [Section 9.1.1, “Workspace logs in CLI”](#)
- [Section 9.1.2, “Workspace logs in OpenShift console”](#)
- [Section 9.1.3, “Language servers and debug adapters logs in the editor”](#)

9.1.1. Workspace logs in CLI

You can use the OpenShift CLI to observe the OpenShift Dev Spaces workspace logs.

Prerequisites

- The OpenShift Dev Spaces workspace `<workspace_name>` is running.
- Your OpenShift CLI session has access to the OpenShift project `<namespace_name>` containing this workspace.

Procedure

- Get the logs from the pod running the `<workspace_name>` workspace in the `<namespace_name>` project:


```
$ oc logs --follow --namespace='<workspace_namespace>' \
--selector='controller.devfile.io/devworkspace_name=<workspace_name>'
```

9.1.2. Workspace logs in OpenShift console

You can use the OpenShift console to observe the OpenShift Dev Spaces workspace logs.

Procedure

1. In the OpenShift Dev Spaces dashboard, go to **Workspaces**.
2. Click on a workspace name to display the workspace overview page. This page displays the OpenShift project name *<project_name>*.
3. Click on the upper right **Applications** menu, and click the OpenShift console link.
4. Run the next steps in the OpenShift console, in the **Administrator** perspective.
5. Click **Workloads > Pods** to see a list of all the active workspaces.
6. In the **Project** drop-down menu, select the *<project_name>* project to narrow the search.
7. Click on the name of the running pod that runs the workspace. The **Details** tab contains the list of all containers with additional information.
8. Go to the **Logs** tab.

9.1.3. Language servers and debug adapters logs in the editor

In the Visual Studio Code editor running in your workspace, you can configure the installed language server and debug adapter extensions to view their logs.

Procedure

1. Configure the extension: click **File > Preferences > Settings**, expand the **Extensions** section, search for your extension, and set the **trace.server** or similar configuration to **verbose**, if such configuration exists. Refer to the extension documentation for further configuration.
2. View your language server logs by clicking **View → Output**, and selecting your language server in the drop-down list for the Output view.

Additional resources

- [Open VSX registry](#)

9.2. TROUBLESHOOTING WORKSPACE START FAILURES

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.

9.2.1. 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 mode** link.
3. Check the **Logs** tab to find a reason for the workspace failure.

9.2.2. 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 mode** option. Alternatively, this option is also available in the workspace details, under the **Actions** drop-down menu.
3. Check the **Logs** tab to find a reason for the workspace failure.

9.3. 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.

9.3.1. 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/3.4/html-single/administration_guide/index#administration-guide: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 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/3.4/html-single/administration_guide/index#administration-guide:installing-che-in-a-restricted-environment.

Optimizing workspace plugins

Role: User

When selecting various plugins, each plugin can bring its own sidecar container, which is an OCI image. OpenShift pulls the images of these sidecar containers.

Reduce the number of plugins, or disable them to see if start time is faster. See also https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.4/html-single/administration_guide/index#administration-guide: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 plugin has three optional routes.

By reducing the number of endpoints and checking endpoints of all plugins, 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.

9.3.2. Improving workspace runtime performance

Providing enough CPU resources

Plugins consume CPU resources. For example, when a plugin provides IntelliSense features, adding more CPU resources can improve performance.

Ensure the CPU settings in the devfile definition, **devfile.yaml**, are correct:

```
apiVersion: 1.0.0

components:
-
  type: chePlugin
  id: __<plugin_id>__
  cpuLimit: 1360Mi 1
  cpuRequest: 100m 2
```

- 1** Specifies the CPU limit for the plugin.
- 2** Specifies the CPU request for the plugin.

Providing enough memory

Plug-ins consume CPU and memory resources. For example, when a plugin provides IntelliSense features, collecting data can consume all the memory allocated to the container.

Providing more memory to the plugin can increase performance. Ensure about the correctness of memory settings:

- in the plugin 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 plugin.

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

```
apiVersion: 1.0.0

components:
-
  type: chePlugin
  id: __<plugin_id>__
  memoryLimit: 1048M 1
  memoryRequest: 256M
```

- 1** Specifies the memory limit for this plugin.

9.4. 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.

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.