Red Hat OpenShift Dev Spaces 3.13

User guide

Using Red Hat OpenShift Dev Spaces 3.13

Last Updated: 2024-04-25

Red Hat OpenShift Dev Spaces 3.13 User guide

Using Red Hat OpenShift Dev Spaces 3.13

Jana Vrbkova
jvrbkova@redhat.com

Legal Notice

Abstract

Information for users using Red Hat OpenShift Dev Spaces.

Table of Contents

CHAPTER 1. GETTING STARTED WITH DEV SPACES

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:

1.1. STARTING A WORKSPACE FROM A GIT REPOSITORY URL

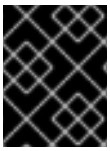
With OpenShift Dev Spaces, you can use a URL in your browser to start a new workspace that contains a clone of a Git repository. This way, you can clone a Git repository that is hosted on GitHub, GitLab, Bitbucket or Microsoft Azure DevOps server instances.

TIP

You can also use the **Git Repository 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.

IMPORTANT

- Starting a workspace using a URL from a Git service other than GitHub, GiLab, Bitbucket, or Microsoft Azure DevOps will fail.
- If you use an SSH URL to start a new workspace, you must propagate the SSH key. See Configuring DevWorkspaces to use SSH keys for Git operations for more information.
- If the SSH URL points to a private repository, you must apply an access token to be able to fetch the **devfile.yaml** content. You can do this either by accepting an SCM authentication page or following a Personal Access Token procedure.



IMPORTANT

Configure personal access token to access private repositories. See Section 6.1.2, “Using a Git-provider access token”.

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://*<openshift_dev_spaces_fqdn>***.
- 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 1.1.1, “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://<openshift_dev_spaces_fqdn>#<git_repository_url>
```

TIP

You can extend this URL with optional parameters:

```
https://<openshift_dev_spaces_fqdn>#<git_repository_url>?<optional_parameters> 1
```

1 See Section 1.1.1, “Optional parameters for the URLs for starting a new workspace” .

TIP

You can use Git+SSH URLs to start a new workspace. See Configuring DevWorkspaces to use SSH keys for Git operations

Example 1.1. A URL for starting a new workspace

- **https://*<openshift_dev_spaces_fqdn>*#https://github.com/che-samples/cpp-hello-world**
- **https://*<openshift_dev_spaces_fqdn>*#git@github.com:che-samples/cpp-hello-world.git**

Example 1.2. The URL syntax for starting a new workspace with a clone of a GitHub instance repository

- **https://*<openshift_dev_spaces_fqdn>*#https://*<github_host>*/*<user_or_org>*/*<repository>*** starts a new workspace with a clone of the default branch.
- **https://*<openshift_dev_spaces_fqdn>*#https://*<github_host>*/*<user_or_org>*/*<repository>*/tree/*<branch_name>*** starts a new workspace with a clone of the specified branch.

- **https://*<openshift_dev_spaces_fqdn>*#https://*<github_host>*/*<user_or_org>*/*<repository>*/pull/*<pull_request_id>*** starts a new workspace with a clone of the branch of the pull request.
- **https://*<openshift_dev_spaces_fqdn>*#git@*<github_host>*:*<user_or_org>*/*<repository>*.git** starts a new workspace from Git+SSH URL.

Example 1.3. The URL syntax for starting a new workspace with a clone of a GitLab instance repository

- **https://*<openshift_dev_spaces_fqdn>*#https://*<gitlab_host>*/*<user_or_org>*/*<repository>*** starts a new workspace with a clone of the default branch.
- **https://*<openshift_dev_spaces_fqdn>*#https://*<gitlab_host>*/*<user_or_org>*/*<repository>*/-/tree/*<branch_name>*** starts a new workspace with a clone of the specified branch.
- **https://*<openshift_dev_spaces_fqdn>*#git@*<gitlab_host>*:*<user_or_org>*/*<repository>*.git** starts a new workspace from Git+SSH URL.

Example 1.4. The URL syntax for starting a new workspace with a clone of a BitBucket Server repository

- **https://*<openshift_dev_spaces_fqdn>*#https://*<bb_host>*/scm/*<project-key>*/*<repository>*.git** starts a new workspace with a clone of the default branch.
- **https://*<openshift_dev_spaces_fqdn>*#https://*<bb_host>*/users/*<user_slug>*/repos/*<repository>*/** starts a new workspace with a clone of the default branch, if a repository was created under the user profile.
- **https://*<openshift_dev_spaces_fqdn>*#https://*<bb_host>*/users/*<user-slug>*/repos/*<repository>*/browse?at=refs%2Fheads%2F*<branch-name>*** starts a new workspace with a clone of the specified branch.
- **https://*<openshift_dev_spaces_fqdn>*#git@*<bb_host>*:*<user_slug>*/*<repository>*.git** starts a new workspace from Git+SSH URL.

Example 1.5. The URL syntax for starting a new workspace with a clone of a Microsoft Azure DevOps Git repository

- **https://*<openshift_dev_spaces_fqdn>*#https://*<organization>*@dev.azure.com/*<organization>*/*<project>*/_git/*<repository>*** starts a new workspace with a clone of the default branch.
- **https://*<openshift_dev_spaces_fqdn>*#https://*<organization>*@dev.azure.com/*<organization>*/*<project>*/_git/*<repository>*?version=GB*<branch>*** starts a new workspace with a clone of the specific branch.
- **https://*<openshift_dev_spaces_fqdn>*#git@ssh.dev.azure.com:v3/*<organization>*/*<project>*/*<repository>*** starts a new workspace from Git+SSH URL.

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

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://*<openshift_dev_spaces_fqdn>*/*<user_name>*/*<unique_url>*.

Additional resources

- Section 1.1.1, "Optional parameters for the URLs for starting a new workspace"
- Section 1.3, "Basic actions you can perform on a workspace"
- Section 6.1.2, "Using a Git-provider access token"
- Section 6.2.1, "Mounting Git configuration"
- Configuring DevWorkspaces to use SSH keys for Git operations

1.1.1. 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 1.1.1.1, "URL parameter concatenation"
- Section 1.1.1.2, "URL parameter for the IDE"
- Section 1.1.1.3, "URL parameter for the IDE image"
- Section 1.1.1.4, "URL parameter for starting duplicate workspaces"
- Section 1.1.1.5, "URL parameter for the devfile file name"
- Section 1.1.1.6, "URL parameter for the devfile file path"
- Section 1.1.1.7, "URL parameter for the workspace storage"
- Section 1.1.1.8, "URL parameter for additional remotes"
- Section 1.1.1.9, "URL parameter for a container image"

1.1.1.1. 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://*<openshift_dev_spaces_fqdn>*#*<git_repository_url>*?*<url_parameter_1>*&*<url_parameter_2>*&*<url_parameter_3>*

Example 1.6. 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://*<openshift_dev_spaces_fqdn>*#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://<openshift_dev_spaces_fqdn> 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.

1.1.1.2. URL parameter for the IDE

You can use the **che-editor=** URL parameter to specify a supported IDE when starting a workspace.

TIP

Use the **che-editor=** parameter when you cannot add or edit a **/.che/che-editor.yaml** file in the source-code Git repository to be cloned for workspaces.

NOTE

The **che-editor=** parameter overrides the **/.che/che-editor.yaml** file.

This parameter accepts two types of values:

- **che-editor=*<editor_key>***

  ```
  https://<openshift_dev_spaces_fqdn>#<git_repository_url>?che-editor=<editor_key>
  ```

  Table 1.1. The URL parameter *<editor_key>* values for supported IDEs

IDE	*<editor_key>* value	Note
Microsoft Visual Studio Code - Open Source	**che-incubator/che-code/latest**	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. Use the Dashboard to select this IDE.

- **che-editor=*<url_to_a_file>***

  ```
  https://<openshift_dev_spaces_fqdn>#<git_repository_url>?che-editor=<url_to_a_file>
  ```
  (1)

  (1) URL to a file with devfile content.

  TIP

  - The URL must point to the raw file content.
  - To use this parameter with a **che-editor.yaml** file, copy the file with another name or path, and remove the line with **inline** from the file.
  - The che-editors.yaml file features the devfiles of all supported IDEs.

1.1.1.3. URL parameter for the IDE image

You can use the **editor-image** parameter to set the custom IDE image for the workspace.

IMPORTANT

- If the Git repository contains **/.che/che-editor.yaml** file, the custom editor will be overridden with the new IDE image.
- If there is no **/.che/che-editor.yaml** file in the Git repository, the default editor will be overridden with the new IDE image.
- If you want to override the supported IDE and change the target editor image, you can use both parameters together: **che-editor** and **editor-image** URL parameters.

The URL parameter to override the IDE image is **editor-image=**:

```
https://<openshift_dev_spaces_fqdn>#<git_repository_url>?editor-image=<container_registry/image_name:image_tag>
```

Example:

https://*<openshift_dev_spaces_fqdn>*#https://github.com/eclipse-che/che-docs?editor-image=quay.io/che-incubator/che-code:next

or

https://*<openshift_dev_spaces_fqdn>*#https://github.com/eclipse-che/che-docs?che-editor=che-incubator/che-code/latest&editor-image=quay.io/che-incubator/che-code:next

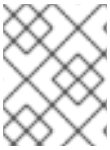
1.1.1.4. URL parameter for starting duplicate workspaces

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://<openshift_dev_spaces_fqdn>#<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.

1.1.1.5. 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://<openshift_dev_spaces_fqdn>#<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**.

1.1.1.6. 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://<openshift_dev_spaces_fqdn>#<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.

1.1.1.7. 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://*<openshift_dev_spaces_fqdn>*#*<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*

1.1.1.8. 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://*<openshift_dev_spaces_fqdn>*#*<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.

1.1.1.9. URL parameter for a container image

You can use the **image** parameter to use a custom reference to a container image in the following scenarios:

- The Git repository contains no devfile, and you want to start a new workspace with the custom image.

- The Git repository contains a devfile, and you want to override the first container image listed in the **components** section of the devfile.

The URL parameter for the path to the container image is **image=**:

```
https://<openshift_dev_spaces_fqdn>#<git_repository_url>?image=<container_image_url>
```

Example

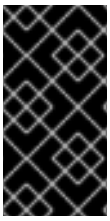
https://*<openshift_dev_spaces_fqdn>*#https://github.com/eclipse-che/che-docs?image=quay.io/devfile/universal-developer-image:ubi8-latest

1.2. STARTING A WORKSPACE FROM A RAW DEVFILE URL

With OpenShift Dev Spaces, you can open a **devfile** URL in your browser to start a new workspace.

TIP

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



IMPORTANT

To initiate a clone of the Git repository in the filesystem of a new workspace, the devfile must contain project info.

See https://devfile.io/docs/2.2.0/adding-projects.

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://*<openshift_dev_spaces_fqdn>***.

Procedure

To start a new workspace from a devfile URL:

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 from a **public** repository using the basic syntax:

   ```
   https://<openshift_dev_spaces_fqdn>#<devfile_url>
   ```

   You can pass your personal access token to the URL to access a devfile from **private** repositories:

   ```
   https://<openshift_dev_spaces_fqdn>#https://<token>@<host>/<path_to_devfile> (1)
   ```

   (1) Your personal access token that you generated on the Git provider's website.

This works for GitHub, GitLab, Bitbucket, Microsoft Azure, and other providers that support Personal Access Token.

IMPORTANT

Automated Git credential injection does not work in this case. To configure the Git credentials, use the configure personal access token guide.

TIP

You can extend this URL with optional parameters:

```
https://<openshift_dev_spaces_fqdn>#<devfile_url>?<optional_parameters>
```
(1)

(1) See Section 1.1.1, "Optional parameters for the URLs for starting a new workspace" .

Example 1.7. A URL for starting a new workspace from a public repository

https://*<openshift_dev_spaces_fqdn>*#https://raw.githubusercontent.com/che-samples/cpp-hello-world/main/devfile.yaml

Example 1.8. A URL for starting a new workspace from a private repository

https://*<openshift_dev_spaces_fqdn>*#https://*<token>*@raw.githubusercontent.com/che-samples/cpp-hello-world/main/devfile.yaml

Verification

After you enter the URL to start a new workspace in a browser tab, the workspace starting page appears. When the new workspace is ready, the workspace IDE loads in the browser tab.

The workspace has a unique URL:
https://*<openshift_dev_spaces_fqdn>*/*<user_name>*/*<unique_url>*.

Additional resources

- Section 1.1.1, "Optional parameters for the URLs for starting a new workspace"
- Section 1.3, "Basic actions you can perform on a workspace"
- Section 6.1.2, "Using a Git-provider access token"
- Section 6.2.1, "Mounting Git configuration"
- Configuring DevWorkspaces to use SSH keys for Git operations

1.3. BASIC ACTIONS YOU CAN PERFORM ON A WORKSPACE

You manage your workspaces and verify their current states in the **Workspaces** page (**https://*<openshift_dev_spaces_fqdn>*/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 1.2. Basic actions you can perform on a workspace

Action	GUI steps in the Workspaces page
Reopen a running workspace	Click **Open**.
Restart a running workspace	Go to ⋮ > **Restart Workspace**.
Stop a running workspace	Go to ⋮ > **Stop Workspace**.
Start a stopped workspace	Click **Open**.
Delete a workspace	Go to ⋮ > **Delete Workspace**.

1.4. 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, Bitbucket, or Microsoft Azure Repos 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 or configure SSH keys for Git operations .

Additional resources

- Administration Guide: Configuring OAuth for Git providers
- User Guide: Using a Git-provider access token
- Configuring DevWorkspaces to use SSH keys for Git operations

1.5. USING THE FUSE-OVERLAYFS STORAGE DRIVER FOR PODMAN AND BUILDAH

By default, newly created workspaces that do not specify a devfile will use the Universal Developer Image (UDI). The UDI contains common development tools and dependencies commonly used by developers.

Podman and Buildah are included in the UDI, allowing developers to build and push container images from their workspace.

By default, Podman and Buildah in the UDI are configured to use the **vfs** storage driver. For more efficient image management, use the fuse-overlayfs storage driver which supports copy-on-write in rootless environments.

You must meet the following requirements to fuse-overlayfs in a workspace:

1. For OpenShift versions older than 4.15, the administrator has enabled **/dev/fuse** access on the cluster by following https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.13/html-single/administration_guide/index#administration-guide:configuring-fuse.

2. The workspace has the necessary annotations for using the **/dev/fuse** device. See Section 1.5.1, "Accessing /dev/fuse".

3. The **storage.conf** file in the workspace container has been configured to use fuse-overlayfs. See Section 1.5.2, "Enabling fuse-overlayfs with a ConfigMap" .

Additional resources

- Universal Developer Image

1.5.1. Accessing /dev/fuse

You must have access to **/dev/fuse** to use fuse-overlayfs. This section describes how to make **/dev/fuse** accessible to workspace containers.

Prerequisites

- For OpenShift versions older than 4.15, the administrator has enabled access to **/dev/fuse** by following https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.13/html-single/administration_guide/index#administration-guide:configuring-fuse.

- Determine a workspace to use fuse-overlayfs with.

Procedure

1. Use the **pod-overrides** attribute to add the required annotations defined in https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.13/html-single/administration_guide/index#administration-guide:configuring-fuse to the workspace. The **pod-overrides** attribute allows merging certain fields in the workspace pod's **spec**.
For OpenShift versions older than 4.15:

```
$ oc patch devworkspace <DevWorkspace_name> \
  --patch '{"spec":{"template":{"attributes":{"pod-overrides":{"metadata":{"annotations":
{"io.kubernetes.cri-o.Devices":"/dev/fuse","io.openshift.podman-fuse":""}}}}}}}' \
  --type=merge
```

For OpenShift version 4.15 and later:

```
$ oc patch devworkspace <DevWorkspace_name> \
  --patch '{"spec":{"template":{"attributes":{"pod-overrides":{"metadata":{"annotations":
{"io.kubernetes.cri-o.Devices":"/dev/fuse"}}}}}}}' \
  --type=merge
```

Verification steps

1. Start the workspace and verify that **/dev/fuse** is available in the workspace container.

```
$ stat /dev/fuse
```

After completing this procedure, follow the steps in Section 1.5.2, "Enabling fuse-overlayfs with a ConfigMap" to use fuse-overlayfs for Podman.

1.5.2. Enabling fuse-overlayfs with a ConfigMap

You can define the storage driver for Podman and Buildah in the **~/.config/containers/storage.conf** file. Here are the default contents of the **/home/user/.config/containers/storage.conf** file in the UDI container:

storage.conf

```
[storage]
driver = "vfs"
```

To use fuse-overlayfs, **storage.conf** can be set to the following:

storage.conf

```
[storage]
driver = "overlay"

[storage.options.overlay]
mount_program="/usr/bin/fuse-overlayfs" 1
```

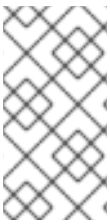
1 The absolute path to the **fuse-overlayfs** binary. The **/usr/bin/fuse-overlayfs** path is the default for the UDI.

You can do this manually after starting a workspace. Another option is to build a new image based on the UDI with changes to **storage.conf** and use the new image for workspaces.

Otherwise, you can update the **/home/user/.config/containers/storage.conf** for all workspaces in your project by creating a ConfigMap that mounts the updated file. See Section 6.2, "Mounting ConfigMaps".

Prerequisites

- For OpenShift versions older than 4.15, the administrator has enabled access to **/dev/fuse** by following https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.13/html-single/administration_guide/index#administration-guide:configuring-fuse.
- A workspace with the required annotations are set by following Section 1.5.1, "Accessing /dev/fuse"



NOTE

Since ConfigMaps mounted by following this guide mounts the ConfigMap's data to all workspaces, following this procedure will set the storage driver to fuse-overlayfs for all workspaces. Ensure that your workspaces contain the required annotations to use fuse-overlayfs by following Section 1.5.1, "Accessing /dev/fuse".

Procedure

1. Apply a ConfigMap that mounts a **/home/user/.config/containers/storage.conf** file in your project.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: fuse-overlay
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
  annotations:
    controller.devfile.io/mount-as: file
    controller.devfile.io/mount-path: /home/user/.config/containers
data:
  storage.conf: |
    [storage]
    driver = "overlay"

    [storage.options.overlay]
    mount_program="/usr/bin/fuse-overlayfs"
```

Verification steps

1. Start the workspace containing the required annotations and verify that the storage driver is **overlay**.

```
$ podman info | grep overlay
```

Example output:

```
graphDriverName: overlay
  overlay.mount_program:
    Executable: /usr/bin/fuse-overlayfs
    Package: fuse-overlayfs-1.12-1.module+el8.9.0+20326+387084d0.x86_64
      fuse-overlayfs: version 1.12
  Backing Filesystem: overlayfs
```

CHAPTER 2. USING DEV SPACES IN TEAM WORKFLOW

Learn about the benefits of using OpenShift Dev Spaces in your organization in the following articles:

- Section 2.1, “Badge for first-time contributors”
- Section 2.2, “Reviewing pull and merge requests”

2.1. BADGE FOR FIRST-TIME CONTRIBUTORS

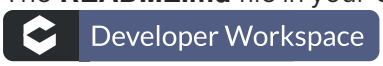
To enable a first-time contributor to start a workspace with a project, add a badge with a link to your OpenShift Dev Spaces instance.

Figure 2.1. Factory badge

Procedure

1. Substitute your OpenShift Dev Spaces URL (**https://*<openshift_dev_spaces_fqdn>***) 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://<openshift_dev_spaces_fqdn>/#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.

2.2. 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.

2.3. TRY IN WEB IDE GITHUB ACTION

The Try in Web IDE GitHub action can be added to a GitHub repository workflow to help reviewers quickly test pull requests on Eclipse Che hosted by Red Hat. The action achieves this by listening to pull request events and providing a factory URL by creating a comment, a status check, or both. This factory URL creates a new workspace from the pull request branch on Eclipse Che hosted by Red Hat.

NOTE

The Che documentation repository (https://github.com/eclipse/che-docs) is a real-life example where the Try in Web IDE GitHub action helps reviewers quickly test pull requests. Experience the workflow by navigating to a recent pull request and opening a factory URL.

Figure 2.2. Pull request comment created by the Try in Web IDE GitHub action. Clicking the badge opens a new workspace for reviewers to test the pull request.

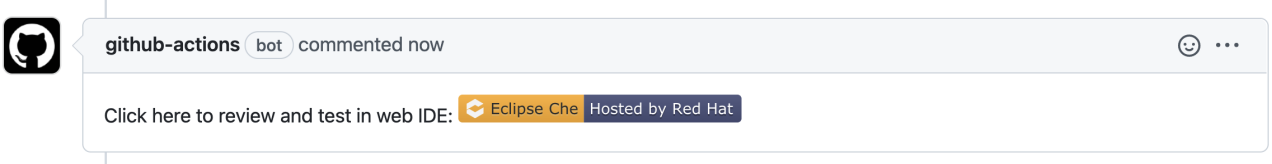
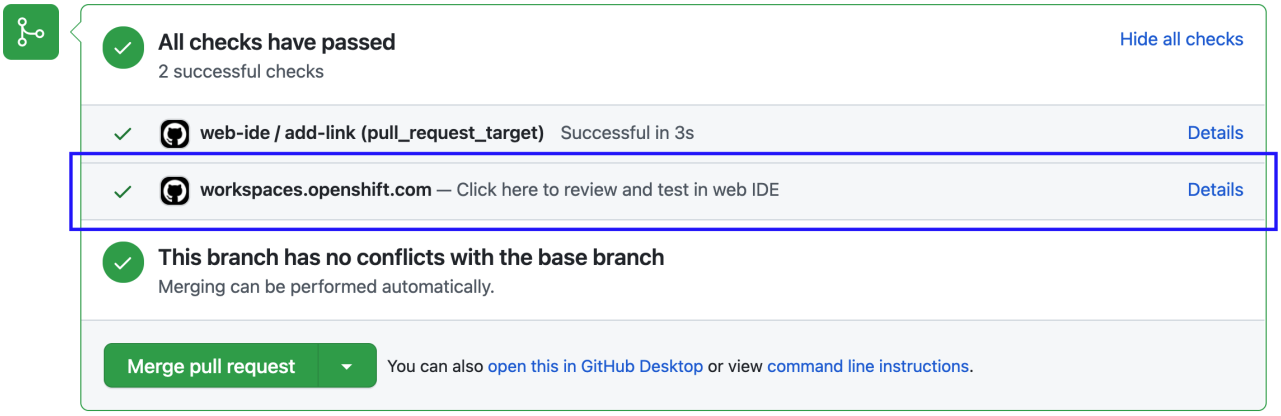


Figure 2.3. Pull request status check created by the Try in Web IDE GitHub action. Clicking the "Details" link opens a new workspace for reviewers to test the pull request.



2.3.1. Adding the action to a GitHub repository workflow

This section describes how to integrate the Try in Web IDE GitHub action to a GitHub repository workflow.

Prerequisites

- A GitHub repository
- A devfile in the root of the GitHub repository.

Procedure

1. In the GitHub repository, create a **.github/workflows** directory if it does not exist already.
2. Create an **example.yml** file in the **.github/workflows** directory with the following content:

Example 2.1. example.yml

```
name: Try in Web IDE example

on:
  pull_request_target:
    types: [opened]

jobs:
  add-link:
    runs-on: ubuntu-20.04
    steps:
      - name: Web IDE Pull Request Check
        id: try-in-web-ide
        uses: redhat-actions/try-in-web-ide@v1
        with:
          # GitHub action inputs

          # required
          github_token: ${{ secrets.GITHUB_TOKEN }}

          # optional - defaults to true
          add_comment: true
```

```
        # optional - defaults to true
        add_status: true
```

This code snippet creates a workflow named **Try in Web IDE example**, with a job that runs the **v1** version of the **redhat-actions/try-in-web-ide** community action. The workflow is triggered on the **pull_request_target** event, on the **opened** activity type.

3. Optionally configure the activity types from the **on.pull_request_target.types** field to customize when workflow trigger. Activity types such as **reopened** and **synchronize** can be useful.

Example 2.2. Triggering the workflow on both opened and synchronize activity types

```
on:
  pull_request_target:
    types: [opened, synchronize]
```

4. Optionally configure the **add_comment** and **add_status** GitHub action inputs within **example.yml**. These inputs are sent to the Try in Web IDE GitHub action to customize whether comments and status checks are to be made.

2.3.2. Providing a devfile

Providing a devfile in the root directory of the repository is recommended to define the development environment of the workspace created by the factory URL. In this way, the workspace contains everything users need to review pull requests, such as plugins, development commands, and other environment setup.

The Che documentation repository devfile is an example of a well-defined and effective devfile.

CHAPTER 3. CUSTOMIZING WORKSPACE COMPONENTS

To customize workspace components:

- Choose a Git repository for your workspace .
- Use a devfile.
- Configure an IDE.
- Add OpenShift Dev Spaces specific attributes in addition to the generic devfile specification.

CHAPTER 4. INTRODUCTION TO DEVFILE IN DEV SPACES

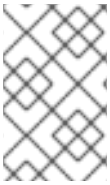
Devfiles are **yaml** text files used for development environment customization. Use them to configure a devfile to suit your specific needs and share the customized devfile across multiple workspaces to ensure identical user experience and build, run, and deploy behaviours across your team.

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. IDES IN WORKSPACES

5.1. SUPPORTED IDES

The default IDE in a new workspace is Microsoft Visual Studio Code - Open Source. Alternatively, you can choose another supported IDE:

Table 5.1. Supported IDEs

IDE	id	Note
Microsoft Visual Studio Code - Open Source	**che-incubator/che-code/latest**	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. Use the Dashboard to select this IDE.

5.2. REPOSITORY-LEVEL IDE CONFIGURATION IN OPENSHIFT DEV SPACES

You can store IDE configuration files directly in the remote Git repository that contains your project source code. This way, one common IDE configuration is applied to all new workspaces that feature a clone of that repository. Such IDE configuration files might include the following:

- The **/.che/che-editor.yaml** file that stores a definition of the chosen IDE.
- IDE-specific configuration files that one would typically store locally for a desktop IDE. For example, the **/.vscode/extensions.json** file.

5.3. MICROSOFT VISUAL STUDIO CODE - OPEN SOURCE

The OpenShift Dev Spaces build of Microsoft Visual Studio Code - Open Source is the default IDE of a new workspace.

You can automate installation of Microsoft Visual Studio Code extensions from the Open VSX registry at workspace startup. See *Automating installation of Microsoft Visual Studio Code extensions at workspace startup*.

TIP

- Use Tasks to find and run the commands specified in **devfile.yaml**.
- Use **Dev Spaces** commands by clicking **Dev Spaces** in the Status Bar or finding them through the Command Palette:
  - **Dev Spaces: Open Dashboard**
  - **Dev Spaces: Open OpenShift Console**
  - **Dev Spaces: Stop Workspace**
  - **Dev Spaces: Restart Workspace**
  - **Dev Spaces: Restart Workspace from Local Devfile**
  - **Dev Spaces: Open Documentation**

TIP

Configure IDE preferences on a per-workspace basis by invoking the Command Palette and selecting **Preferences: Open Workspace Settings**.

NOTE

You might see your organization's branding in this IDE if your organization customized it through a branded build.

5.3.1. Automating installation of Microsoft Visual Studio Code extensions at workspace startup

To have the Microsoft Visual Studio Code - Open Source IDE automatically install chosen extensions, you can add an **extensions.json** file to the remote Git repository that contains your project source code and that will be cloned into workspaces.

Prerequisites

- The public OpenVSX registry at open-vsx.org is selected and accessible over the internet. See Selecting an Open VSX registry instance .

  TIP

  To install recommended extensions in a restricted environment, consider the following options instead:

  - https://access.redhat.com/documentation/en-us/red_hat_openshift_dev_spaces/3.13/html-single/administration_guide/index#administration-guide:configuring-the-open-vsx-registry-url to point to your OpenVSX registry.
  - Section 5.4, "Defining a common IDE" .
  - Installing extensions from VSX files .

Procedure

1. Get the publisher and extension names of each chosen extension:
   a. Find the extension on the Open VSX registry website and copy the URL of the extension's listing page.
   b. Extract the *<publisher>* and *<extension>* names from the copied URL:

```
https://www.open-vsx.org/extension/<publisher>/<extension>
```

2. Create a **.vscode/extensions.json** file in the remote Git repository.
3. Add the *<publisher>* and *<extension>* names to the **extensions.json** file as follows:

```
{
  "recommendations": [
    "<publisher_A>.<extension_B>",
    "<publisher_C>.<extension_D>",
    "<publisher_E>.<extension_F>"
  ]
}
```

Verification

1. Start a new workspace by using the URL of the remote Git repository that contains the created **extensions.json** file.
2. In the IDE of the workspace, press **Ctrl**+**Shift**+**X** or go to **Extensions** to find each of the extensions listed in the file.
3. The extension has the label **This extension is enabled globally**

Additional resources

- Open VSX registry - Extensions for Microsoft Visual Studio Code compatible editors
- Microsoft Visual Studio Code - Workspace recommended extensions

5.4. DEFINING A COMMON IDE

While the Section 1.1.1.2, "URL parameter for the IDE" enables you to start a workspace with your personal choice of the supported IDE, you might find it more convenient to define the same IDE for all workspaces for the same source code Git repository. To do so, use the **che-editor.yaml** file. This file supports even a detailed IDE configuration.

TIP

If you intend to start most or all of your organization's workspaces with the same IDE other than Microsoft Visual Studio Code - Open Source, an alternative is for the administrator of your organization's OpenShift Dev Spaces instance to specify another supported IDE as the default IDE at the OpenShift Dev Spaces instance level. This can be done with **.spec.devEnvironments.defaultEditor** in the **CheCluster** Custom Resource.

5.4.1. Setting up che-editor.yaml

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

Procedure

- In the remote Git repository of your project source code, create a **/.che/che-editor.yaml** file with lines that specify the relevant parameter.

Verification

1. Start a new workspace with a clone of the Git repository .
2. Verify that the specified IDE loads in the browser tab of the started workspace.

5.4.2. 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 from the table of supported IDEs:

Table 5.2. Supported IDEs

IDE	id	Note
Microsoft Visual Studio Code - Open Source	**che-incubator/che-code/latest**	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. Use the Dashboard to select this 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 remoteche-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
- Git configuration
- 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, Bitbucket, or Microsoft Azure Repos 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 use a personal access token. You can configure personal access tokens on the **User Preferences** page of your OpenShift Dev Spaces dashboard: **https://*<openshift_dev_spaces_fqdn>*/dashboard/#/user-preferences?tab=personal-access-tokens**, or apply it manually as a Kubernetes Secret in the namespace.

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 workspaces with clones of private Git repositories that are hosted on GitHub, GitLab, Bitbucket Server, or Microsoft Azure Repos.

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

Prerequisites

- You have logged in to the cluster.

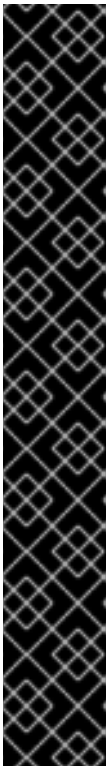
  TIP

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

  $ oc login https://*<openshift_dev_spaces_fqdn>* --username=*<my_user>*

Procedure

1. Generate your access token on your Git provider's website.



IMPORTANT

Personal access tokens are sensitive information and should be kept confidential. Treat them like passwords. If you are having trouble with authentication, ensure you are using the correct token and have the appropriate permissions for cloning repositories:

1. Open a terminal locally on your computer
2. Use the **git** command to clone the repository using your personal access token. The format of the **git** command vary based on the Git Provider. As an example, GitHub personal access token verification can be done using the following command:

```
git clone https://<PAT>@github.com/username/repo.git
```

Replace **<PAT>** with your personal access token, and **username/repo** with the appropriate repository path. If the token is valid and has the necessary permissions, the cloning process should be successful. Otherwise, this is an indicator of incorrect personal access token, insufficient permissions, or other issues.

IMPORTANT

For GitHub Enterprise Cloud, verify that the token is authorized for use within the organization.

2. Go to **https://*<openshift_dev_spaces_fqdn>*/api/user/id** in the web browser to get your OpenShift Dev Spaces user ID.
3. 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-organization: <git_provider_organization> (4)
stringData:
  token: <Content_of_access_token>
type: Opaque
```

(1) Your OpenShift Dev Spaces user ID.

(2) The Git provider name: **github** or **gitlab** or **bitbucket-server** or **azure-devops**.

(3) The Git provider URL.

4 This line is only applicable to **azure-devops**: your Git provider user organization.

4. Visit **https://*<openshift_dev_spaces_fqdn>*/api/kubernetes/namespace** to get your OpenShift Dev Spaces user namespace as **name**.

5. 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>*

6. 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.

Additional resources

- Deploying Che with support for Git repositories with self-signed certificates
- Authorizing a personal access token for use with SAML single sign-on

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
```

2. 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 **/etc/config/*<ConfigMap_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 6.2. 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.

6.2.1. Mounting Git configuration

NOTE

The **user.name** and **user.email** fields will be set automatically to the **gitconfig** content from a git provider, connected to OpenShift Dev Spaces by a Git-provider access token or a token generated via OAuth, if username and email are set on the provider's user profile page.

Follow the instructions below to mount a Git config file in a workspace.

Prerequisites

- You have logged in to the cluster.

Procedure

1. Prepare a new OpenShift ConfigMap.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: workspace-userdata-gitconfig-configmap
  namespace: <user_namespace> 1
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /etc/
data:
  gitconfig: <gitconfig content> 2
```

1. A user namespace. Visit **https://*<openshift_dev_spaces_fqdn>*/api/kubernetes/namespace** to get your OpenShift Dev Spaces user namespace as **name**.
2. The content of your gitconfig file content.

2. Apply the ConfigMap.

```
$ oc apply -f - <<EOF
<ConfigMap_prepared_in_step_1>
EOF
```

Verification

1. Start a new workspace by using the URL of a remote Git repository that the Git provider hosts.

2. Once the workspace is started, open a new terminal in the **tools** container and run **git config --get-regexp user.***. Your Git user name and email should appear in the output.

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>
```

3. Optional: When using JBoss 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.

4. 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.
6. 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

1. Apply the Secret for the TLS certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /public-certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  nexus.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:
  NPM_CONFIG_REGISTRY: >-
    https://<npm_artifact_repository_route>/repository/npm-all/
```

6.3.3.1. Disabling self-signed certificate validation

Run the command below to disable SSL/TLS, bypassing the validation of your self-signed certificates. Note that this is a potential security risk. For a better solution, configure a self-signed certificate you trust with **NODE_EXTRA_CA_CERTS**.

Procedure

- Run the following command in the terminal:

```
npm config set strict-ssl false
```

6.3.3.2. Configuring NODE_EXTRA_CA_CERTS to use a certificate

Use the command below to set NODE_EXTRA_CA_CERTS to point to where you have your SSL/TLS certificate.

Procedure

- Run the following command in the terminal:

```
`export NODE_EXTRA_CA_CERTS=/public-certs/nexus.cer` (1)
`npm install`
```

(1) **/public-certs/nexus.cer** is the path to self-signed SSL/TLS certificate of Nexus artifactory.

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)
    controller.devfile.io/read-only: 'true'
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi (2)
  storageClassName: <storage_class_name> (3)
  volumeMode: Filesystem
```

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

(2) Example size value of the requested storage.

(3) The name of the StorageClass required by the claim. Remove this line if you want to use a default StorageClass.

default StorageClass.

CHAPTER 8. INTEGRATING WITH OPENSHIFT

8.1. MANAGING WORKSPACES WITH OPENSHIFT 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://*<openshift_dev_spaces_fqdn>*/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

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

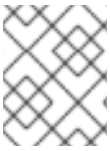
  ```
  $ oc get devworkspaces
  ```

  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.

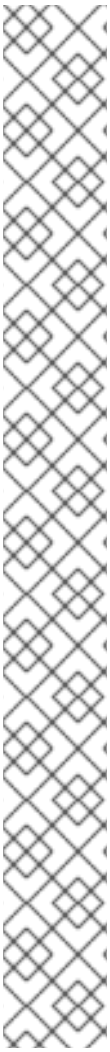


NOTE

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.

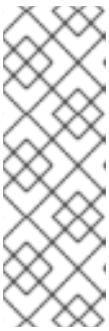
  TIP

  You can visit **https://*<openshift_dev_spaces_fqdn>*/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.



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/3.13/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://<openshift_dev_spaces_fqdn>/plugin-registry/v3/plugins/che-incubator/che-code/latest/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
```

2. 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://*<openshift_dev_spaces_fqdn>*/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 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://*<openshift_dev_spaces_fqdn>*/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 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://*<openshift_dev_spaces_fqdn>*/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:

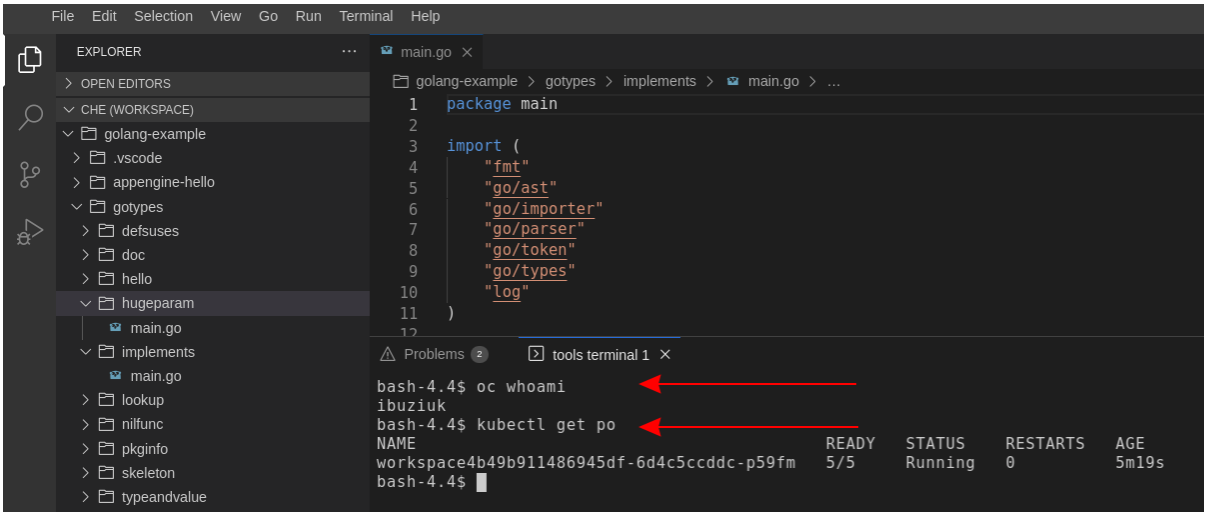
```
$ oc delete devworkspace <workspace_name> -n <user_namespace>
```

8.2. AUTOMATIC 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.



WARNING

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

8.3. NAVIGATING 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:

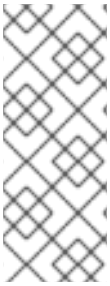
- Create and deploy applications on the OpenShift Container Platform by importing existing codebases, images, and Dockerfiles.
- Visually interact with applications, components, and services associated with them within a project and monitor their deployment and build status.
- Group components within 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.

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

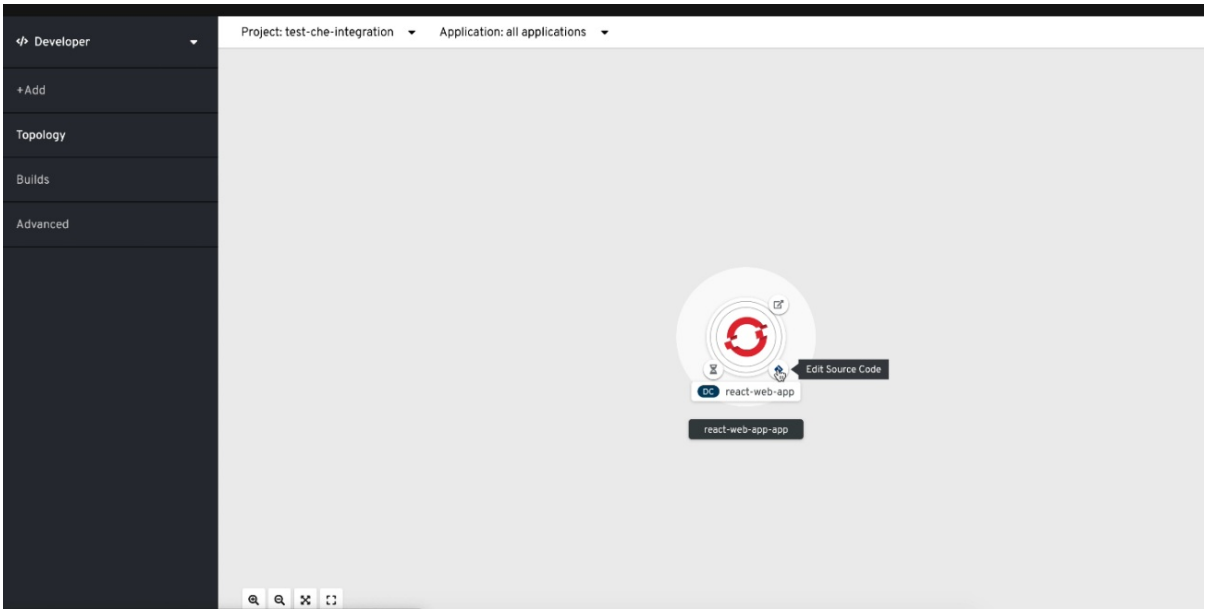
This section describes how to start editing the source code of applications running on OpenShift using OpenShift Dev Spaces.

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**.



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.

8.3.3. Accessing OpenShift Dev Spaces from Red Hat Applications menu

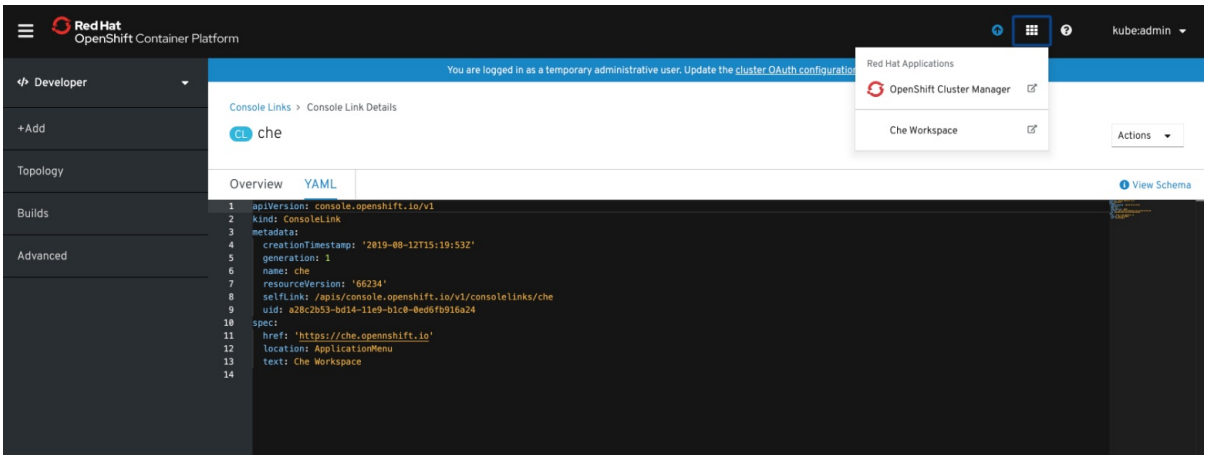
This section describes how to access OpenShift Dev Spaces workspaces from the **Red Hat Applications** menu on the OpenShift Container Platform.

Prerequisites

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

Procedure

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.

8.4. NAVIGATING OPENSHIFT WEB CONSOLE FROM DEV SPACES

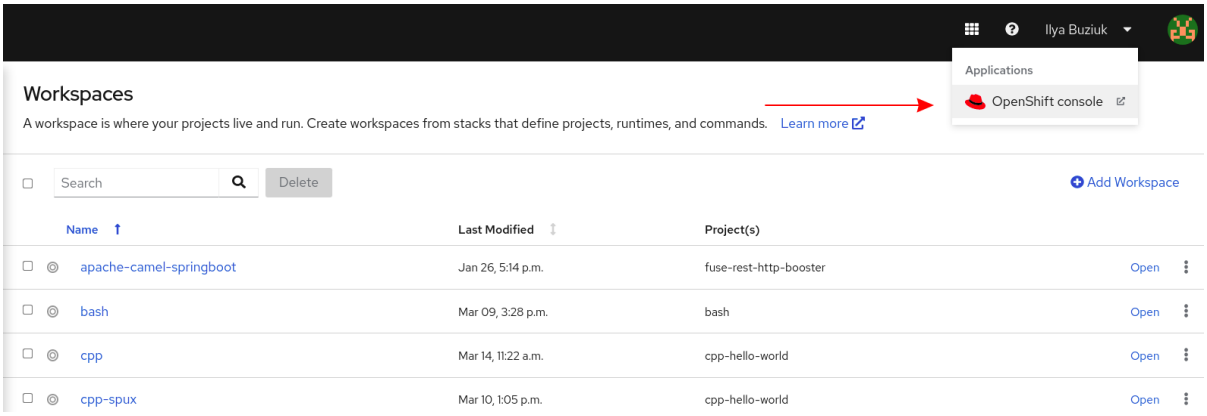
This section describes how to access OpenShift web console from OpenShift Dev Spaces.

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 slow workspaces”
- Section 9.3, “Troubleshooting network problems”
- Section 9.4, “Troubleshooting webview loading error”

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 Microsoft Visual Studio Code - Open Source 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 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.

9.2.1. 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/3.13/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/3.13/html-single/administration_guide/index#administration-guide:installing-che-in-a-restricted-environment.

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**), Microsoft Visual Code - Open Source has three optional routes.

By reducing the number of endpoints and checking endpoints of all plugins, workspace start can be faster.

9.2.2. 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:

```
components:
  - name: tools
    container:
      image: quay.io/devfile/universal-developer-image:ubi8-latest
      cpuLimit: 4000m (1)
      cpuRequest: 1000m (2)
```

1. Specifies the CPU limit
2. Specifies the CPU request

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 container can increase performance. Ensure that memory settings in the devfile definition **devfile.yaml** file are correct.

```
components:
  - name: tools
    container:
      image: quay.io/devfile/universal-developer-image:ubi8-latest
      memoryLimit: 6G (1)
      memoryRequest: 512Mi (2)
```

1. Specifies the memory limit
2. Specifies the memory request

9.3. 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.

9.4. TROUBLESHOOTING WEBVIEW LOADING ERROR

If you use Microsoft Visual Studio Code - Open Source in a private browsing window, you might encounter the following error message: **Error loading webview: Error: Could not register service workers**.

This is a known issue affecting following browsers:

- Google Chrome in Incognito mode

- Mozilla Firefox in Private Browsing mode

Table 9.1. Dealing with the webview error in a private browsing window

Browser	Workarounds
Google Chrome	Go to Settings → Privacy and security → Cookies and other site data → Allow all cookies.
Mozilla Firefox	Webviews are not supported in Private Browsing mode. See this reported bug for details.