Red Hat OpenShift Dev Spaces 3.0

Administration guide

Administering Red Hat OpenShift Dev Spaces 3.0

Last Updated: 2022-09-28

Red Hat OpenShift Dev Spaces 3.0 Administration guide

Administering Red Hat OpenShift Dev Spaces 3.0

Robert Kratky
rkratky@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Jana Vrbkova
jvrbkova@redhat.com

Max Leonov
mleonov@redhat.com

Legal Notice

Abstract

Information for administrators operating Red Hat OpenShift Dev Spaces.

Table of Contents

CHAPTER 1. PREPARING THE INSTALLATION

To prepare a OpenShift Dev Spaces installation, learn about OpenShift Dev Spaces ecosystem and deployment constraints:

- Section 1.1, “Supported platforms”
- Section 1.2, “OpenShift Dev Spaces architecture”
- Section 1.3, “Calculating OpenShift Dev Spaces resource requirements”
- Section 3.1, “Understanding the **CheCluster** Custom Resource”

1.1. SUPPORTED PLATFORMS

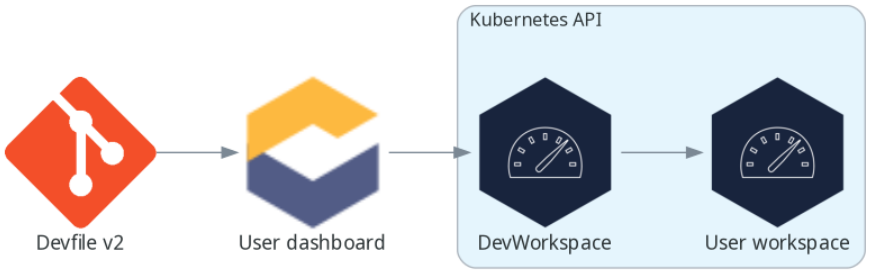
OpenShift Dev Spaces 3.0 is available on listed platforms with the listed supported installation methods:

Table 1.1. Supported deployment environments for OpenShift Dev Spaces 3.0

Platform	Architectures	Deployment method
OpenShift Container Platform 4.10	• AMD64 and Intel 64 (**x86_64**) • IBM Power (**ppc64le**) • IBM Z (**s390x**)	• OpenShift web console • **dsc** management tool
OpenShift Dedicated 4.10	• AMD64 and Intel 64 (**x86_64**)	• OpenShift web console
Red Hat OpenShift Service on AWS (ROSA) 4.10	• AMD64 and Intel 64 (**x86_64**)	• OpenShift web console

1.2. OPENSHIFT DEV SPACES ARCHITECTURE

Figure 1.1. High-level OpenShift Dev Spaces architecture with the DevWorkspace operator



OpenShift Dev Spaces runs on three groups of components:

OpenShift Dev Spaces server components

Manage User project and workspaces. The main component is the User dashboard, from which users control their workspaces.

DevWorkspace operator

Creates and controls the necessary OpenShift objects to run User workspaces. Including **Pods**, **Services**, and **PeristentVolumes**.

User workspaces

Container-based development environments, the IDE included.

The role of these OpenShift features is central:

DevWorkspace Custom Resources

Valid OpenShift objects representing the User workspaces and manipulated by OpenShift Dev Spaces. It is the communication channel for the three groups of components.

OpenShift role-based access control (RBAC)

Controls access to all resources.

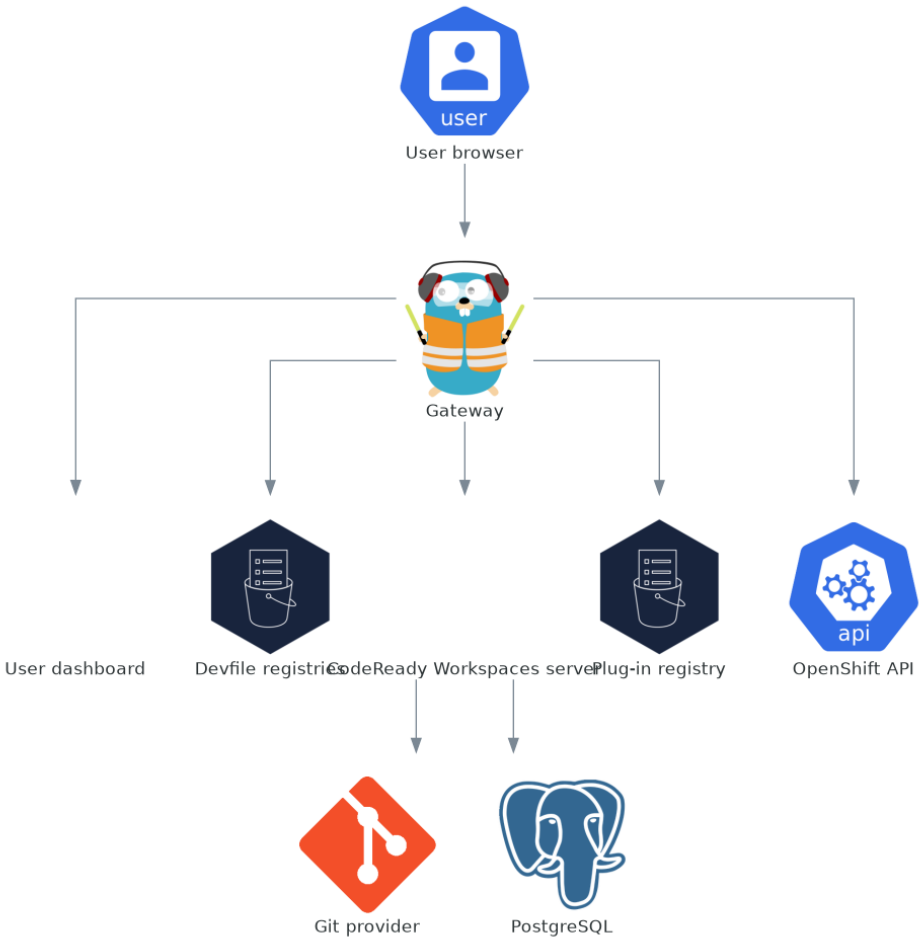
Additional resources

- Section 1.2.1, “OpenShift Dev Spaces server components”
- Section 1.2.1.2, “DevWorkspace operator”
- Section 1.2.2, “User workspaces”
- DevWorkspace Operator repository
- Kubernetes documentation - Custom Resources

1.2.1. OpenShift Dev Spaces server components

The OpenShift Dev Spaces server components ensure multi-tenancy and workspaces management.

Figure 1.2. OpenShift Dev Spaces server components interacting with the DevWorkspace operator



Additional resources

- Section 1.2.1.1, “OpenShift Dev Spaces operator”
- Section 1.2.1.2, “DevWorkspace operator”
- Section 1.2.1.3, “Gateway”
- Section 1.2.1.4, “User dashboard”
- Section 1.2.1.5, “Devfile registries”

- Section 1.2.1.6, “OpenShift Dev Spaces server”
- Section 1.2.1.7, “PostgreSQL”
- Section 1.2.1.8, “Plug-in registry”

1.2.1.1. OpenShift Dev Spaces operator

The OpenShift Dev Spaces operator ensure full lifecycle management of the OpenShift Dev Spaces server components. It introduces:

CheCluster custom resource definition (CRD)

Defines the **CheCluster** OpenShift object.

OpenShift Dev Spaces controller

Creates and controls the necessary OpenShift objects to run a OpenShift Dev Spaces instance, such as pods, services, and persistent volumes.

CheCluster custom resource (CR)

On a cluster with the OpenShift Dev Spaces operator, it is possible to create a **CheCluster** custom resource (CR). The OpenShift Dev Spaces operator ensures the full lifecycle management of the OpenShift Dev Spaces server components on this OpenShift Dev Spaces instance:

- Section 1.2.1.2, “DevWorkspace operator”
- Section 1.2.1.3, “Gateway”
- Section 1.2.1.4, “User dashboard”
- Section 1.2.1.5, “Devfile registries”
- Section 1.2.1.6, “OpenShift Dev Spaces server”
- Section 1.2.1.7, “PostgreSQL”
- Section 1.2.1.8, “Plug-in registry”

Additional resources

- Section 3.1, “Understanding the **CheCluster** Custom Resource”
- Chapter 2, *Installing OpenShift Dev Spaces*

1.2.1.2. DevWorkspace operator

The DevWorkspace operator extends OpenShift to provide DevWorkspace support. It introduces:

DevWorkspace custom resource definition

Defines the DevWorkspace OpenShift object from the Devfile v2 specification.

DevWorkspace controller

Creates and controls the necessary OpenShift objects to run a DevWorkspace, such as pods, services, and persistent volumes.

DevWorkspace custom resource

On a cluster with the DevWorkspace operator, it is possible to create DevWorkspace custom resources (CR). A DevWorkspace CR is a OpenShift representation of a Devfile. It defines a User workspaces in a OpenShift cluster.

Additional resources

- Devfile API repository

1.2.1.3. Gateway

The OpenShift Dev Spaces gateway has following roles:

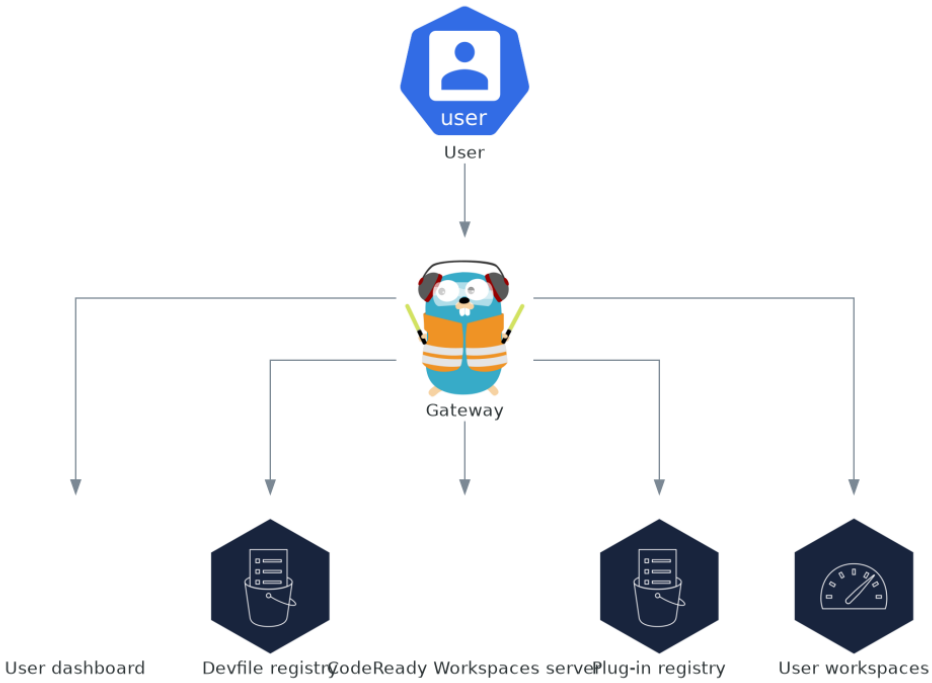
- Routing requests. It uses Traefik.
- Authenticating users with OpenID Connect (OIDC). It uses OpenShift OAuth2 proxy.
- Applying OpenShift Role based access control (RBAC) policies to control access to any OpenShift Dev Spaces resource. It uses `kube-rbac-proxy`.

The OpenShift Dev Spaces operator manages it as the **che-gateway** Deployment.

It controls access to:

- Section 1.2.1.4, “User dashboard”
- Section 1.2.1.5, “Devfile registries”
- Section 1.2.1.6, “OpenShift Dev Spaces server”
- Section 1.2.1.8, “Plug-in registry”
- Section 1.2.2, “User workspaces”

Figure 1.3. OpenShift Dev Spaces gateway interactions with other components



Additional resources

- Section 3.10, “Managing identities and authorizations”

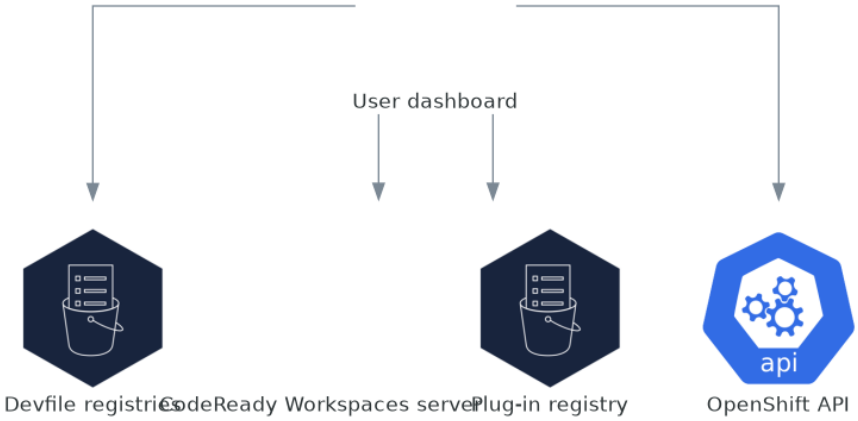
1.2.1.4. User dashboard

The user dashboard is the landing page of Red Hat OpenShift Dev Spaces. OpenShift Dev Spaces users browse the user dashboard to access and manage their workspaces. It is a React application. The OpenShift Dev Spaces deployment starts it in the **devspaces-dashboard** Deployment.

It need access to:

- Section 1.2.1.5, “Devfile registries”
- Section 1.2.1.6, “OpenShift Dev Spaces server”
- Section 1.2.1.8, “Plug-in registry”
- OpenShift API

Figure 1.4. User dashboard interactions with other components



When the user requests the user dashboard to start a workspace, the user dashboard executes this sequence of actions:

1. Collects the devfile from the Section 1.2.1.5, "Devfile registries", when the user is creating a workspace from a code sample.
2. Sends the repository URL to Section 1.2.1.6, "OpenShift Dev Spaces server" and expects a devfile in return, when the user is creating a workspace from a remote devfile.
3. Reads the devfile describing the workspace.
4. Collects the additional metadata from the Section 1.2.1.8, "Plug-in registry".
5. Converts the information into a DevWorkspace Custom Resource.
6. Creates the DevWorkspace Custom Resource in the user project using the OpenShift API.
7. Watches the DevWorkspace Custom Resource status.
8. Redirects the user to the running workspace IDE.

1.2.1.5. Devfile registries

Additional resources

The OpenShift Dev Spaces devfile registries are services providing a list of sample devfiles to create ready-to-use workspaces. The Section 1.2.1.4, "User dashboard" displays the samples list on the

Dashboard → Create Workspace page. Each sample includes a Devfile v2. The OpenShift Dev Spaces deployment starts one devfile registry instance in the **devfile-registry** deployment.

Figure 1.5. Devfile registries interactions with other components

Additional resources

- Devfile v2 documentation
- devfile registry latest community version online instance
- OpenShift Dev Spaces devfile registry repository

1.2.1.6. OpenShift Dev Spaces server

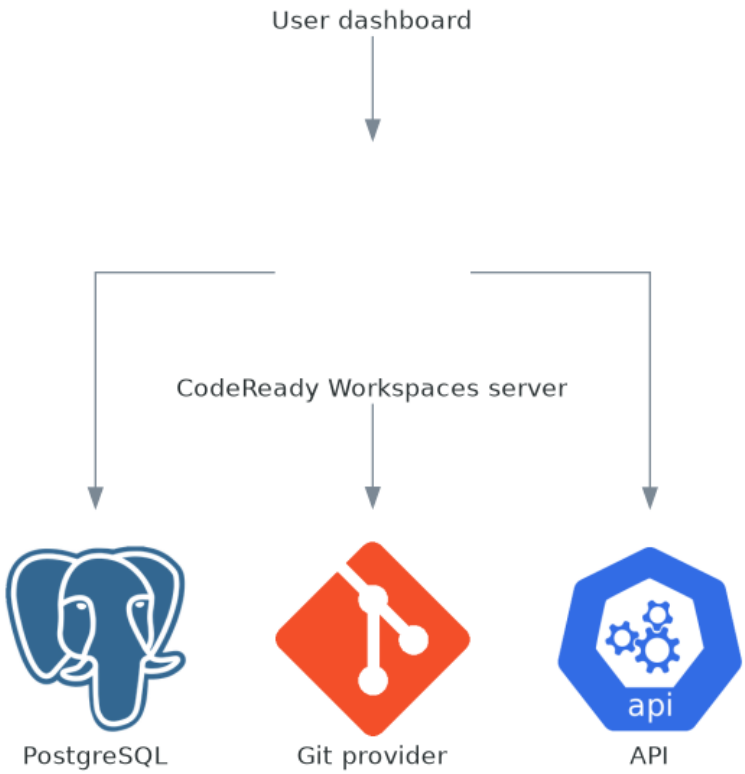
The OpenShift Dev Spaces server main functions are:

- Creating user namespaces.
- Provisioning user namespaces with required secrets and config maps.
- Integrating with Git services providers, to fetch and validate devfiles and authentication.

The OpenShift Dev Spaces server is a Java web service exposing an HTTP REST API and needs access to:

- Section 1.2.1.7, “PostgreSQL”
- Git service providers
- OpenShift API

Figure 1.6. OpenShift Dev Spaces server interactions with other components



Additional resources

- Section 3.3.2, "Advanced configuration options for the OpenShift Dev Spaces server component"

1.2.1.7. PostgreSQL

OpenShift Dev Spaces server uses the PostgreSQL database to persist user configurations such as workspaces metadata.

The OpenShift Dev Spaces deployment starts a dedicated PostgreSQL instance in the **postgres** Deployment. You can use an external database instead.

Figure 1.7. PostgreSQL interactions with other components

Additional resources

- **quay.io/eclipse/che—centos—postgresql-96-centos7** container image
- **quay.io/eclipse/che—centos—postgresql-13-centos7** container image

1.2.1.8. Plug-in registry

Each OpenShift Dev Spaces workspace starts with a specific editor and set of associated extensions. The OpenShift Dev Spaces plug-in registry provides the list of available editors and editor extensions. A Devfile v2 describes each editor or extension.

The Section 1.2.1.4, "User dashboard" is reading the content of the registry.

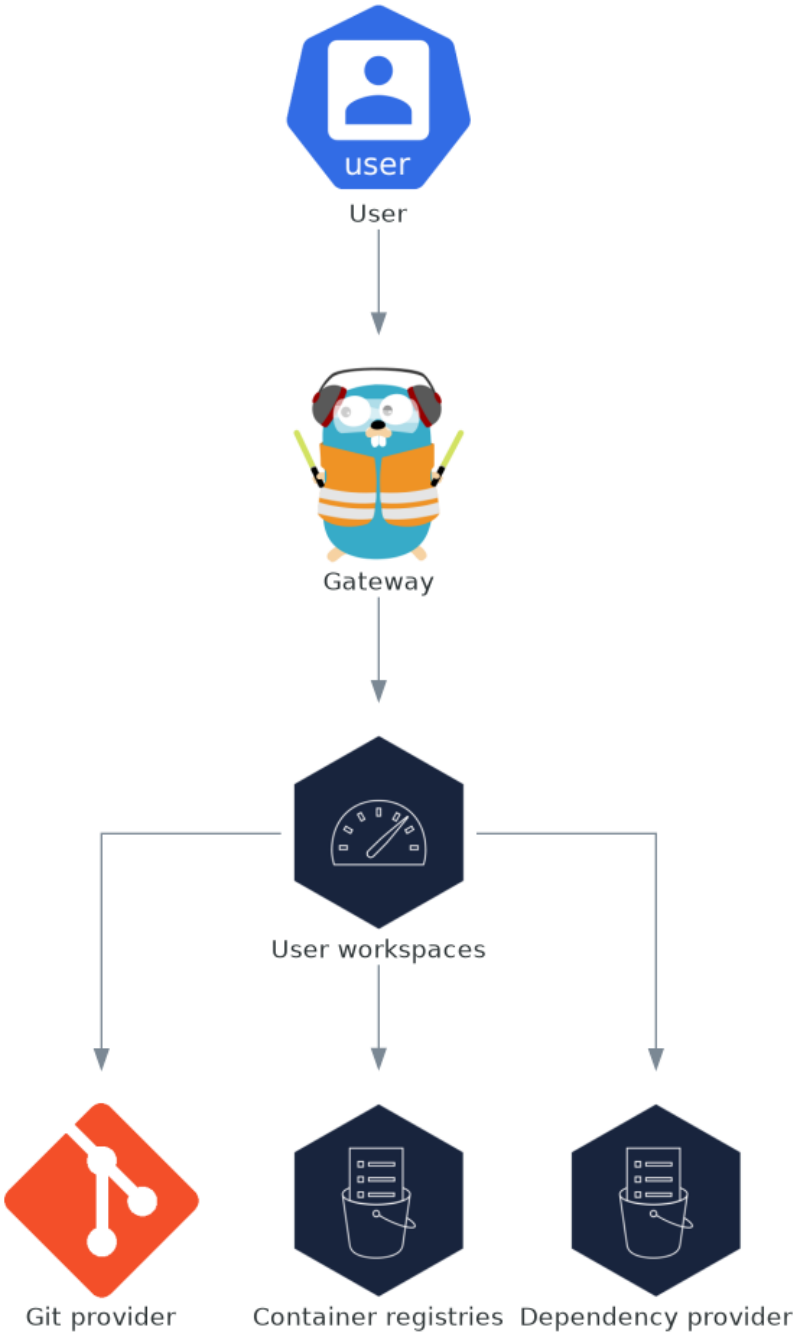
Figure 1.8. Plug-in registries interactions with other components

Additional resources

- Editors definitions in the OpenShift Dev Spaces plug-in registry repository
- Plug-ins definitions in the OpenShift Dev Spaces plug-in registry repository
- Plug-in registry latest community version online instance

1.2.2. User workspaces

Figure 1.9. User workspaces interactions with other components



User workspaces are web IDEs running in containers.

A User workspace is a web application. It consists of microservices running in containers providing all the services of a modern IDE running in your browser:

- Editor
- Language auto-completion
- Language server
- Debugging tools
- Plug-ins
- Application runtimes

A workspace is one OpenShift Deployment containing the workspace containers and enabled plug-ins, plus related OpenShift components:

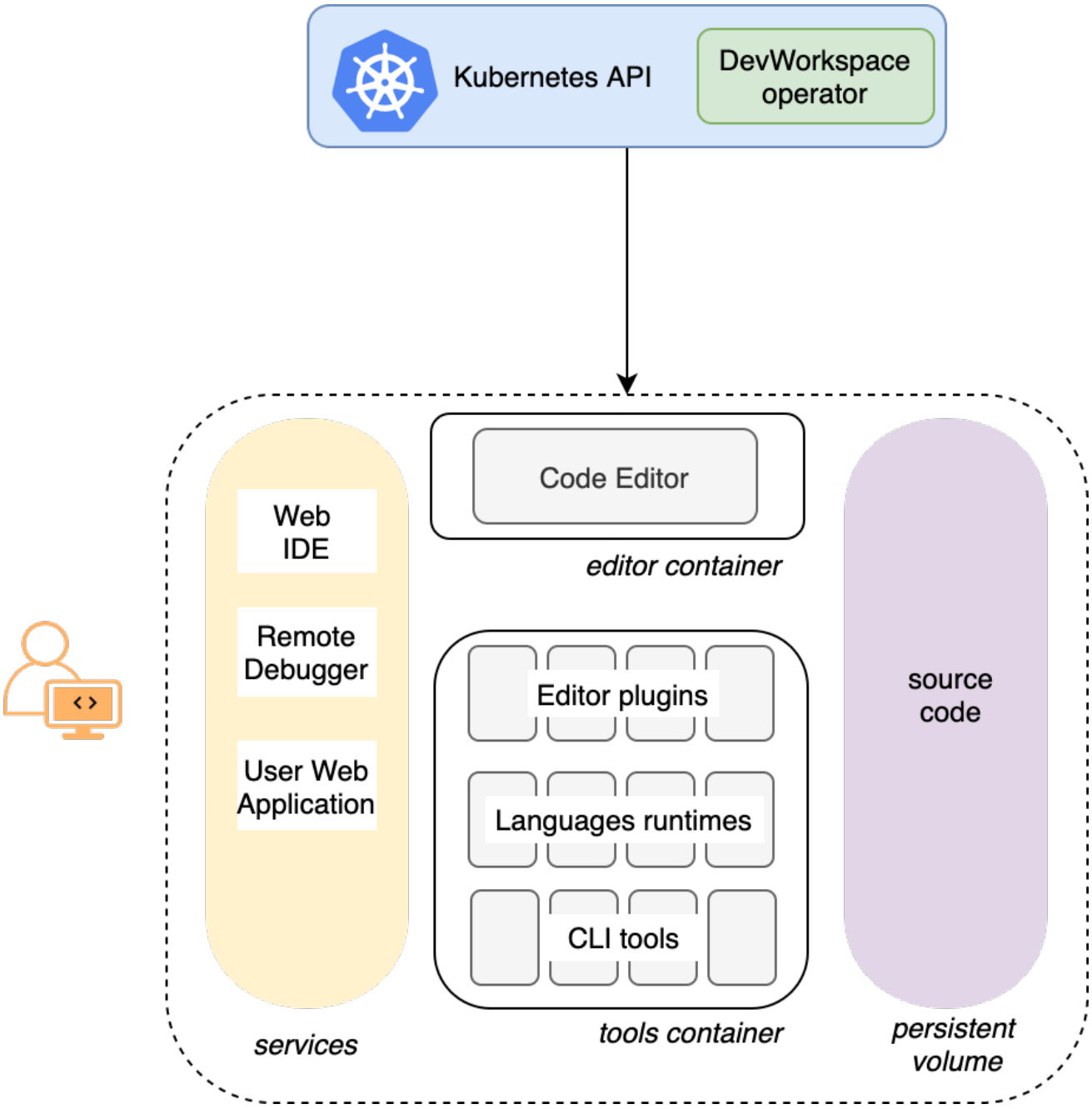
- Containers
- ConfigMaps
- Services
- Endpoints
- Ingresses or Routes
- Secrets
- Persistent Volumes (PVs)

A OpenShift Dev Spaces workspace contains the source code of the projects, persisted in a OpenShift Persistent Volume (PV). Microservices have read-write access to this shared directory.

Use the devfile v2 format to specify the tools and runtime applications of a OpenShift Dev Spaces workspace.

The following diagram shows one running OpenShift Dev Spaces workspace and its components.

Figure 1.10. OpenShift Dev Spaces workspace components



In the diagram, there is one running workspaces.

1.3. CALCULATING OPENSHIFT DEV SPACES RESOURCE REQUIREMENTS

The OpenShift Dev Spaces Operator, DevWorkspace Controller, and user workspaces consist of a set of pods. The pods contribute to the resource consumption in terms of CPU and RAM limits and requests. Learn how to calculate resources, such as memory and CPU, required to run Red Hat OpenShift Dev Spaces.

1.3.1. OpenShift Dev Spaces Operator requirements

The OpenShift Dev Spaces Operator manages six operands running in six distinct pods. The following table presents the default resource requirements of each of these operands.

Table 1.2. OpenShift Dev Spaces operator operands

Pod	Container names	Default memory limit	Default memory request
OpenShift Dev Spaces Server	OpenShift Dev Spaces	1 Gi	512 MiB
OpenShift Dev Spaces Gateway	**gateway**, **configbump**, **oauth-proxy**, **kube-rbac-proxy**	4 Gi, 256Mi, 512Mi, 512Mi	128 Mi, 64Mi, 64Mi, 64Mi
OpenShift Dev Spaces Dashboard	**OpenShift Dev Spaces-dashboard**	256 Mi	32 Mi
PostgreSQL	**postgres**	1 Gi	512 Mi
Devfile registry	**che-devfile-registry**	256 Mi	32 Mi
Plug-in registry	**che-plugin-registry**	256 Mi	32 Mi

The OpenShift Dev Spaces Operator, which powers all the operands, consists of a single container with the **64Mi** memory request and **256Mi** limit. These default values are sufficient when the OpenShift Dev Spaces Operator manages a relatively big amount of OpenShift Dev Spaces workspaces. For even larger deployments, consider increasing the defaults.

Additional resources

- Section 1.2, “OpenShift Dev Spaces architecture” .

1.3.2. DevWorkspace Operator requirements

The DevWorkspace Operator consists of 3 pods. The following table presents the default resource requirements of each of these pods.

Table 1.3. DevWorkspace Operator Pods

Pod	Container name	Default memory limit	Default memory request
DevWorkspace Controller Manager	**DevWorkspace-controller**, **kube-rbac-proxy**	1 Gi	100 Mi
DevWorkspace Operator Catalog	**registry-server**	N/A	50 Mi
DevWorkspace Webhook Server	**webhook-server}**, kube-rbac-proxy	300 Mi	20 Mi

These default values are sufficient when the DevWorkspace Controller manages a relatively big amount of OpenShift Dev Spaces workspaces. For larger deployments, consider increasing the defaults.

Additional resources

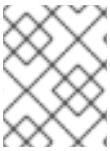
- Section 1.2, “OpenShift Dev Spaces architecture” .

1.3.3. Workspaces requirements

This section describes how to calculate the resources required for a workspace. That is the sum of the resources required for each container of the workspace.

Procedure

1. Identify the workspace components explicitly specified in the **components** section of the devfile.
2. Identify the implicit workspace components.



NOTE

OpenShift Dev Spaces implicitly loads the default **theia-ide**, **che-machine-exec**, **che-gateway** containers.

1. Calculate the requirements for each component.

Additional resources

- Section 1.2, “OpenShift Dev Spaces architecture” .

1.3.4. A workspace example

This section describes a OpenShift Dev Spaces workspace example.

The following devfile defines the OpenShift Dev Spaces workspace:

```
apiVersion: 1.0.0
metadata:
  generateName: nodejs-configmap-
projects:
  - name: nodejs-configmap
    source:
      location: "https://github.com/crw-samples/nodejs-configmap.git"
      branch: 12.x
      type: git
components:
  - id: vscode/typescript-language-features/latest
    type: chePlugin
  - mountSources: true
    type: kubernetes
    entrypoints:
      - command:
          - sleep
        args:
          - infinity
```

```
    reference: 'https://raw.githubusercontent.com/crw-samples/nodejs-mongodb-
sample/master/kubernetes-manifests/guestbook-app.deployment.yaml'
    alias: guestbook-frontend
```

This table provides the memory requirements for each workspace component:

Table 1.4. Total workspace memory requirement and limit

Pod	Container name	Default memory limit	Default memory request
Workspace	**theia-ide**	512 Mi	64 Mi
Workspace	**machine-exec**	128 Mi	32 Mi
Workspace	**tools**	4 Gi	64 Mi
Workspace	**che-gateway**	256 Mi	64 Mi
	Total	**4.9 Gi**	**224 Mi**

Additional resources

- Section 1.2, “OpenShift Dev Spaces architecture”
- Section 3.1, “Understanding the **CheCluster** Custom Resource”
- OpenShift Dev Spaces plug-ins registry repository
- Kubernetes resource management for pods and containers

CHAPTER 2. INSTALLING OPENSHIFT DEV SPACES

This section contains instructions to install Red Hat OpenShift Dev Spaces.

You can deploy only one instance of OpenShift Dev Spaces per cluster.

- Section 2.3, “Installing OpenShift Dev Spaces on OpenShift using the web console”
- Section 2.2, “Installing OpenShift Dev Spaces on OpenShift using the **dsc** management tool”
- Section 2.4, “Installing OpenShift Dev Spaces in a restricted environment on OpenShift”

2.1. INSTALL THE DSC MANAGEMENT TOOL

You can install **dsc**, the Red Hat OpenShift Dev Spaces command-line management tool, on Microsoft Windows, Apple MacOS, and Linux. With **dsc**, you can perform operations the OpenShift Dev Spaces server such as starting, stopping, updating, and deleting the server.

Procedure

1. Navigate to https://developers.redhat.com/products/openshift-dev-spaces/download, and download the OpenShift Dev Spaces CLI management tool archive for version 3.0.
2. Extract the archive to a folder, such as **$HOME/dsc**.
3. Run the **dsc** executable from the extracted folder. For example:

```
$ $HOME/dsc/bin/dsc
```

4. Optionally, to enable running **dsc** without the full path specification, add the extracted **bin** folder to your **$PATH**. For example:

```
PATH=$PATH:$HOME/dsc/bin
```

Verification step

- Display the current version of the tool.

```
$ dsc version
```

Additional resources

- "`dsc` reference documentation"

2.2. INSTALLING OPENSHIFT DEV SPACES ON OPENSHIFT USING THE DSC MANAGEMENT TOOL

You can install OpenShift Dev Spaces on OpenShift.

Prerequisites

- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.
- **dsc**. See: Section 2.1, "Install the dsc management tool" .

Procedure

- Create the OpenShift Dev Spaces instance:

```
$ dsc server:deploy --platform openshift
```

Verification steps

1. Verify the OpenShift Dev Spaces instance status:

```
$ dsc server:status
```

2. Navigate to the OpenShift Dev Spaces cluster instance:

```
$ dsc dashboard:open
```

2.3. INSTALLING OPENSHIFT DEV SPACES ON OPENSHIFT USING THE WEB CONSOLE

This section describes how to install OpenShift Dev Spaces using the OpenShift web console. Consider Section 2.2, "Installing OpenShift Dev Spaces on OpenShift using the **dsc** management tool" instead.

Prerequisites

- An OpenShift web console session by a cluster administrator. See Accessing the web console.

Procedure

1. Install the Red Hat OpenShift Dev Spaces Operator. See Installing from OperatorHub using the web console.
2. Create a OpenShift Dev Spaces instance from the Red Hat OpenShift Dev Spaces Operator. See Creating applications from installed Operators .

Verification

1. To verify that the OpenShift Dev Spaces instance has installed correctly, navigate to the **Dev Spaces Cluster** tab of the **Operator details** page. The **Red Hat OpenShift Dev Spaces instance Specification** page displays the list of Red Hat OpenShift Dev Spaces instances and their status.
2. Click **devspaces CheCluster** and navigate to the **Details** tab.
3. See the content of the following fields:
   - The **Message** field contains error messages. The expected content is **None**.

- The **Red Hat OpenShift Dev Spaces URL**field contains the URL of the Red Hat OpenShift Dev Spaces instance. The URL appears when the deployment finishes successfully.

4. Navigate to the **Resources** tab. View the list of resources assigned to the OpenShift Dev Spaces deployment and their status.

2.4. INSTALLING OPENSHIFT DEV SPACES IN A RESTRICTED ENVIRONMENT ON OPENSHIFT

On an OpenShift cluster operating in a restricted network, public resources are not available.

However, deploying OpenShift Dev Spaces and running workspaces requires the following public resources:

- Operator catalog
- Container images
- Sample projects

To make these resources available, you can replace them with their copy in a registry accessible by the OpenShift cluster.

Prerequisites

- The OpenShift cluster has at least 64 GB of disk space.
- The OpenShift cluster is ready to operate on a restricted network, and the OpenShift control plane has access to the public internet. See About disconnected installation mirroring and Using Operator Lifecycle Manager on restricted networks.
- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.
- An active **oc registry** session to the **registry.redhat.io** Red Hat Ecosystem Catalog. See: Red Hat Container Registry authentication.
- **opm**. See Installing the **opm** CLI.
- **jq**. See Downloading **jq**.
- **podman**. See Installing Podman.
- An active **skopeo** session with administrative access to the *<my_registry>* registry. See Installing Skopeo, Authenticating to a registry , and Mirroring images for a disconnected installation .
- **dsc** for OpenShift Dev Spaces version 3.0. See Section 2.1, “Install the dsc management tool” .

Procedure

1. Download and execute the mirroring script to install a custom Operator catalog and mirror the related images: prepare-restricted-environment.sh .

```
$ bash prepare-restricted-environment.sh \
  --ocp_ver "4.10" \
  --devworkspace_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
```

```
    --devworkspace_operator_version "v0.15.2" \
    --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
    --prod_operator_bundle_name "devspacesoperator" \
    --prod_operator_package_name "devspaces-operator" \
    --prod_operator_version "v3.0.1" \
    --my_registry "<my_registry>"
```

2. Install OpenShift Dev Spaces with the configuration set in the **che-operator-cr-patch.yaml** during the previous step:

```
$ dsc server:deploy --platform=openshift \
  --che-operator-cr-patch-yaml=che-operator-cr-patch.yaml
```

Additional resources

- Red Hat-provided Operator catalogs
- Managing custom catalogs

CHAPTER 3. CONFIGURING OPENSHIFT DEV SPACES

This section describes configuration methods and options for Red Hat OpenShift Dev Spaces.

3.1. UNDERSTANDING THE CHECLUSTER CUSTOM RESOURCE

A default deployment of OpenShift Dev Spaces consists of a **CheCluster** Custom Resource parameterized by the Red Hat OpenShift Dev Spaces Operator.

The **CheCluster** Custom Resource is a Kubernetes object. You can configure it by editing the **CheCluster** Custom Resource YAML file. This file contains sections to configure each component: **auth**, **database**, **server**, **storage**.

The Red Hat OpenShift Dev Spaces Operator translates the **CheCluster** Custom Resource into a config map usable by each component of the OpenShift Dev Spaces installation.

The OpenShift platform applies the configuration to each component, and creates the necessary Pods. When OpenShift detects changes in the configuration of a component, it restarts the Pods accordingly.

Example 3.1. Configuring the main properties of the OpenShift Dev Spaces server component

1. Apply the **CheCluster** Custom Resource YAML file with suitable modifications in the **server** component section.
2. The Operator generates the **che ConfigMap**.
3. OpenShift detects changes in the **ConfigMap** and triggers a restart of the OpenShift Dev Spaces Pod.

Additional resources

- Understanding Operators
- "Understanding Custom Resources"

3.1.1. Using dsc to configure the CheCluster Custom Resource during installation

To deploy OpenShift Dev Spaces with a suitable configuration, edit the **CheCluster** Custom Resource YAML file during the installation of OpenShift Dev Spaces. Otherwise, the OpenShift Dev Spaces deployment uses the default configuration parameterized by the Operator.

Prerequisites

- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.
- **dsc**. See: Section 2.1, “Install the dsc management tool” .

Procedure

- Create a **che-operator-cr-patch.yaml** YAML file that contains the subset of the **CheCluster** Custom Resource to configure:

```
spec:
  <component>:
    <property-to-configure>: <value>
```

- Deploy OpenShift Dev Spaces and apply the changes described in **che-operator-cr-patch.yaml** file:

```
$ dsc server:deploy \
--che-operator-cr-patch-yaml=che-operator-cr-patch.yaml \
--platform <chosen-platform>
```

Verification

1. Verify the value of the configured property:

```
$ oc get configmap che -o jsonpath='{.data.<configured-property>}' \
-n openshift-devspaces
```

Additional resources

- Section 3.1.3, "**CheCluster** Custom Resource fields reference".
- Section 3.3.2, "Advanced configuration options for the OpenShift Dev Spaces server component".

3.1.2. Using the CLI to configure the CheCluster Custom Resource

To configure a running instance of OpenShift Dev Spaces, edit the **CheCluster** Custom Resource YAML file.

Prerequisites

- An instance of OpenShift Dev Spaces on OpenShift.
- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.

Procedure

1. Edit the CheCluster Custom Resource on the cluster:

```
$ oc edit checluster/devspaces -n openshift-devspaces
```

2. Save and close the file to apply the changes.

Verification

1. Verify the value of the configured property:

```
$ oc get configmap che -o jsonpath='{.data.<configured-property>}' \
-n openshift-devspaces
```

Additional resources

Additional resources

- Section 3.1.3, "**CheCluster** Custom Resource fields reference".
- Section 3.3.2, "Advanced configuration options for the OpenShift Dev Spaces server component".

3.1.3. CheCluster Custom Resource fields reference

This section describes all fields available to customize the **CheCluster** Custom Resource.

- Example 3.2, "A minimal **CheCluster** Custom Resource example."
- Table 3.1, "**CheCluster** Custom Resource **server** settings, related to the OpenShift Dev Spaces server component."
- Table 3.2, "**CheCluster** Custom Resource **database** configuration settings related to the database used by OpenShift Dev Spaces."
- Table 3.3, "Custom Resource **auth** configuration settings related to authentication used by OpenShift Dev Spaces."
- Table 3.4, "**CheCluster** Custom Resource **storage** configuration settings related to persistent storage used by OpenShift Dev Spaces."
- Table 3.5, "**CheCluster** Custom Resource **k8s** configuration settings specific to OpenShift Dev Spaces installations on OpenShift."
- Table 3.6, "**CheCluster** Custom Resource **metrics** settings, related to the OpenShift Dev Spaces metrics collection used by OpenShift Dev Spaces."
- Table 3.7, "**CheCluster** Custom Resource **status** defines the observed state of OpenShift Dev Spaces installation"

Example 3.2. A minimal CheCluster Custom Resource example.

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: devspaces
spec:
  auth:
    externalIdentityProvider: false
  database:
    externalDb: false
  server:
    selfSignedCert: false
    gitSelfSignedCert: false
    tlsSupport: true
  storage:
    pvcStrategy: 'common'
    pvcClaimSize: '1Gi'
```

Table 3.1. CheCluster Custom Resource server settings, related to the OpenShift Dev Spaces server component.

Property	Description
airGapContainerRegistryHostname	Optional host name, or URL, to an alternate container registry to pull images from. This value overrides the container registry host name defined in all the default container images involved in a Che deployment. This is particularly useful to install Che in a restricted environment.
airGapContainerRegistryOrganization	Optional repository name of an alternate container registry to pull images from. This value overrides the container registry organization defined in all the default container images involved in a Che deployment. This is particularly useful to install OpenShift Dev Spaces in a restricted environment.
allowUserDefinedWorkspaceNamespaces	Deprecated. The value of this flag is ignored. Defines that a user is allowed to specify a Kubernetes namespace, or an OpenShift project, which differs from the default. It's NOT RECOMMENDED to set to **true** without OpenShift OAuth configured. The OpenShift infrastructure also uses this property.
cheClusterRoles	A comma-separated list of ClusterRoles that will be assigned to Che ServiceAccount. Each role must have **app.kubernetes.io/part-of=che.eclipse.org** label. Be aware that the Che Operator has to already have all permissions in these ClusterRoles to grant them.
cheDebug	Enables the debug mode for Che server. Defaults to **false**.
cheFlavor	Deprecated. The value of this flag is ignored. Specifies a variation of the installation. The options are **che** for upstream Che installations or **devspaces** for Red Hat OpenShift Dev Spaces (formerly Red Hat CodeReady Workspaces) installation
cheHost	Public host name of the installed Che server. When value is omitted, the value it will be automatically set by the Operator. See the **cheHostTLSSecret** field.
cheHostTLSSecret	Name of a secret containing certificates to secure ingress or route for the custom host name of the installed Che server. The secret must have **app.kubernetes.io/part-of=che.eclipse.org** label. See the **cheHost** field.
cheImage	Overrides the container image used in Che deployment. This does NOT include the container image tag. Omit it or leave it empty to use the default container image provided by the Operator.
cheImagePullPolicy	Overrides the image pull policy used in Che deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
cheImageTag	Overrides the tag of the container image used in Che deployment. Omit it or leave it empty to use the default image tag provided by the Operator.
cheLogLevel	Log level for the Che server: **INFO** or **DEBUG**. Defaults to **INFO**.

Property	Description
cheServerIngress	The Che server ingress custom settings.
cheServerRoute	The Che server route custom settings.
cheWorkspaceClusterRole	Custom cluster role bound to the user for the Che workspaces. The role must have **app.kubernetes.io/part-of=che.eclipse.org** label. The default roles are used when omitted or left blank.
customCheProperties	Map of additional environment variables that will be applied in the generated **che** ConfigMap to be used by the Che server, in addition to the values already generated from other fields of the **CheCluster** custom resource (CR). When **customCheProperties** contains a property that would be normally generated in **che** ConfigMap from other CR fields, the value defined in the **customCheProperties** is used instead.
dashboardCpuLimit	Overrides the CPU limit used in the dashboard deployment. In cores. (500m = .5 cores). Default to 500m.
dashboardCpuRequest	Overrides the CPU request used in the dashboard deployment. In cores. (500m = .5 cores). Default to 100m.
dashboardImage	Overrides the container image used in the dashboard deployment. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator.
dashboardImagePullPolicy	Overrides the image pull policy used in the dashboard deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
dashboardIngress	Deprecated. The value of this flag is ignored. Dashboard ingress custom settings.
dashboardMemoryLimit	Overrides the memory limit used in the dashboard deployment. Defaults to 256Mi.
dashboardMemoryRequest	Overrides the memory request used in the dashboard deployment. Defaults to 16Mi.
dashboardRoute	Deprecated. The value of this flag is ignored. Dashboard route custom settings.
devfileRegistryCpuLimit	Overrides the CPU limit used in the devfile registry deployment. In cores. (500m = .5 cores). Default to 500m.
devfileRegistryCpuRequest	Overrides the CPU request used in the devfile registry deployment. In cores. (500m = .5 cores). Default to 100m.

Property	Description
devfileRegistryImage	Overrides the container image used in the devfile registry deployment. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator.
devfileRegistryIngress	Deprecated. The value of this flag is ignored. The devfile registry ingress custom settings.
devfileRegistryMemoryLimit	Overrides the memory limit used in the devfile registry deployment. Defaults to 256Mi.
devfileRegistryMemoryRequest	Overrides the memory request used in the devfile registry deployment. Defaults to 16Mi.
devfileRegistryPullPolicy	Overrides the image pull policy used in the devfile registry deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
devfileRegistryRoute	Deprecated. The value of this flag is ignored. The devfile registry route custom settings.
devfileRegistryUrl	Deprecated in favor of **externalDevfileRegistries** fields.
disableInternalClusterSVCNames	Deprecated. The value of this flag is ignored. Disable internal cluster SVC names usage to communicate between components to speed up the traffic and avoid proxy issues.
externalDevfileRegistries	External devfile registries, that serves sample, ready-to-use devfiles. Configure this in addition to a dedicated devfile registry (when **externalDevfileRegistry** is **false**) or instead of it (when **externalDevfileRegistry** is **true**)
externalDevfileRegistry	Instructs the Operator on whether to deploy a dedicated devfile registry server. By default, a dedicated devfile registry server is started. When **externalDevfileRegistry** is **true**, no such dedicated server will be started by the Operator and configure at least one devfile registry with **externalDevfileRegistries** field.
externalPluginRegistry	Instructs the Operator on whether to deploy a dedicated plugin registry server. By default, a dedicated plugin registry server is started. When **externalPluginRegistry** is **true**, no such dedicated server will be started by the Operator and you will have to manually set the **pluginRegistryUrl** field.
gitSelfSignedCert	When enabled, the certificate from **che-git-self-signed-cert** ConfigMap will be propagated to the Che components and provide particular configuration for Git. Note, the **che-git-self-signed-cert** ConfigMap must have **app.kubernetes.io/part-of=che.eclipse.org** label.

Property	Description
nonProxyHosts	List of hosts that will be reached directly, bypassing the proxy. Specify wild card domain use the following form **.<DOMAIN>** and | as delimiter, for example: **localhost|.my.host.com|123.42.12.32** Only use when configuring a proxy is required. Operator respects OpenShift cluster wide proxy configuration and no additional configuration is required, but defining **nonProxyHosts** in a custom resource leads to merging non proxy hosts lists from the cluster proxy configuration and ones defined in the custom resources. See the doc https://docs.openshift.com/container-platform/4.4/networking/enable-cluster-wide-proxy.html. See also the **proxyURL** fields.
pluginRegistryCpuLimit	Overrides the CPU limit used in the plugin registry deployment. In cores. (500m = .5 cores). Default to 500m.
pluginRegistryCpuRequest	Overrides the CPU request used in the plugin registry deployment. In cores. (500m = .5 cores). Default to 100m.
pluginRegistryImage	Overrides the container image used in the plugin registry deployment. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator.
pluginRegistryIngress	Deprecated. The value of this flag is ignored. Plugin registry ingress custom settings.
pluginRegistryMemoryLimit	Overrides the memory limit used in the plugin registry deployment. Defaults to 256Mi.
pluginRegistryMemoryRequest	Overrides the memory request used in the plugin registry deployment. Defaults to 16Mi.
pluginRegistryPullPolicy	Overrides the image pull policy used in the plugin registry deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
pluginRegistryRoute	Deprecated. The value of this flag is ignored. Plugin registry route custom settings.
pluginRegistryUrl	Public URL of the plugin registry that serves sample ready-to-use devfiles. Set this ONLY when a use of an external devfile registry is needed. See the **externalPluginRegistry** field. By default, this will be automatically calculated by the Operator.
proxyPassword	Password of the proxy server. Only use when proxy configuration is required. See the **proxyURL**, **proxyUser** and **proxySecret** fields.
proxyPort	Port of the proxy server. Only use when configuring a proxy is required. See also the **proxyURL** and **nonProxyHosts** fields.

Property	Description
proxySecret	The secret that contains **user** and **password** for a proxy server. When the secret is defined, the **proxyUser** and **proxyPassword** are ignored. The secret must have **app.kubernetes.io/part-of=che.eclipse.org** label.
proxyURL	URL (protocol+host name) of the proxy server. This drives the appropriate changes in the **JAVA_OPTS** and **https(s)_proxy** variables in the Che server and workspaces containers. Only use when configuring a proxy is required. Operator respects OpenShift cluster wide proxy configuration and no additional configuration is required, but defining **proxyUrl** in a custom resource leads to overrides the cluster proxy configuration with fields **proxyUrl**, **proxyPort**, **proxyUser** and **proxyPassword** from the custom resource. See the doc https://docs.openshift.com/container-platform/4.4/networking/enable-cluster-wide-proxy.html. See also the **proxyPort** and **nonProxyHosts** fields.
proxyUser	User name of the proxy server. Only use when configuring a proxy is required. See also the **proxyURL**, **proxyPassword** and **proxySecret** fields.
selfSignedCert	Deprecated. The value of this flag is ignored. The Che Operator will automatically detect whether the router certificate is self-signed and propagate it to other components, such as the Che server.
serverCpuLimit	Overrides the CPU limit used in the Che server deployment In cores. (500m = .5 cores). Default to 1.
serverCpuRequest	Overrides the CPU request used in the Che server deployment In cores. (500m = .5 cores). Default to 100m.
serverExposureStrategy	Deprecated. The value of this flag is ignored. Sets the server and workspaces exposure type. Possible values are **multi-host**, **single-host**, **default-host**. Defaults to **multi-host**, which creates a separate ingress, or OpenShift routes, for every required endpoint. **single-host** makes Che exposed on a single host name with workspaces exposed on subpaths. Read the docs to learn about the limitations of this approach. Also consult the **singleHostExposureType** property to further configure how the Operator and the Che server make that happen on Kubernetes. **default-host** exposes the Che server on the host of the cluster. Read the docs to learn about the limitations of this approach.
serverMemoryLimit	Overrides the memory limit used in the Che server deployment. Defaults to 1Gi.
serverMemoryRequest	Overrides the memory request used in the Che server deployment. Defaults to 512Mi.

Property	Description
serverTrustStoreConfigMapName	Name of the ConfigMap with public certificates to add to Java trust store of the Che server. This is often required when adding the OpenShift OAuth provider, which has HTTPS endpoint signed with self-signed cert. The Che server must be aware of its CA cert to be able to request it. This is disabled by default. The Config Map must have **app.kubernetes.io/part-of=che.eclipse.org** label.
singleHostGatewayConfigMapLabels	The labels that need to be present in the ConfigMaps representing the gateway configuration.
singleHostGatewayConfigSidecarImage	The image used for the gateway sidecar that provides configuration to the gateway. Omit it or leave it empty to use the default container image provided by the Operator.
singleHostGatewayImage	The image used for the gateway in the single host mode. Omit it or leave it empty to use the default container image provided by the Operator.
tlsSupport	Deprecated. Instructs the Operator to deploy Che in TLS mode. This is enabled by default. Disabling TLS sometimes cause malfunction of some Che components.
useInternalClusterSVCNames	Deprecated in favor of **disableInternalClusterSVCNames**.
workspaceNamespaceDefault	Defines Kubernetes default namespace in which user's workspaces are created for a case when a user does not override it. It's possible to use **<username>**, **<userid>** and **<workspaceid>** placeholders, such as che-workspace-<username>. In that case, a new namespace will be created for each user or workspace.
workspacePodNodeSelector	The node selector that limits the nodes that can run the workspace pods.
workspacePodTolerations	The pod tolerations put on the workspace pods to limit where the workspace pods can run.
workspacesDefaultPlugins	Default plug-ins applied to Devworkspaces.

Table 3.2. CheCluster Custom Resource database configuration settings related to the database used by OpenShift Dev Spaces.

Property	Description
chePostgresContainerResources	PostgreSQL container custom settings
chePostgresDb	PostgreSQL database name that the Che server uses to connect to the DB. Defaults to **dbche**.

Property	Description
chePostgresHostName	PostgreSQL Database host name that the Che server uses to connect to. Defaults is **postgres**. Override this value ONLY when using an external database. See field **externalDb**. In the default case it will be automatically set by the Operator.
chePostgresPassword	PostgreSQL password that the Che server uses to connect to the DB. When omitted or left blank, it will be set to an automatically generated value.
chePostgresPort	PostgreSQL Database port that the Che server uses to connect to. Defaults to 5432. Override this value ONLY when using an external database. See field **externalDb**. In the default case it will be automatically set by the Operator.
chePostgresSecret	The secret that contains PostgreSQL`user` and **password** that the Che server uses to connect to the DB. When the secret is defined, the **chePostgresUser** and **chePostgresPassword** are ignored. When the value is omitted or left blank, the one of following scenarios applies: 1. **chePostgresUser** and **chePostgresPassword** are defined, then they will be used to connect to the DB. 2. **chePostgresUser** or **chePostgresPassword** are not defined, then a new secret with the name **postgres-credentials** will be created with default value of**pgche** for **user** and with an auto-generated value for **password**. The secret must have **app.kubernetes.io/part-of=che.eclipse.org** label.
chePostgresUser	PostgreSQL user that the Che server uses to connect to the DB. Defaults to **pgche**.
externalDb	Instructs the Operator on whether to deploy a dedicated database. By default, a dedicated PostgreSQL database is deployed as part of the Che installation. When **externalDb** is **true**, no dedicated database will be deployed by the Operator and you will need to provide connection details to the external DB you are about to use. See also all the fields starting with: **chePostgres**.
postgresImage	Overrides the container image used in the PostgreSQL database deployment. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator.
postgresImagePullPolicy	Overrides the image pull policy used in the PostgreSQL database deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
postgresVersion	Indicates a PostgreSQL version image to use. Allowed values are: **9.6** and **13.3**. Migrate your PostgreSQL database to switch from one version to another.
pvcClaimSize	Size of the persistent volume claim for database. Defaults to **1Gi**. To update pvc storageclass that provisions it must support resize when OpenShift Dev Spaces has been already deployed.

Table 3.3. Custom Resource auth configuration settings related to authentication used by OpenShift Dev Spaces.

Property	Description
debug	Deprecated. The value of this flag is ignored. Debug internal identity provider.
externalIdentityProvider	Deprecated. The value of this flag is ignored. Instructs the Operator on whether or not to deploy a dedicated Identity Provider (Keycloak or RH SSO instance). Instructs the Operator on whether to deploy a dedicated Identity Provider (Keycloak or RH-SSO instance). By default, a dedicated Identity Provider server is deployed as part of the Che installation. When **externalIdentityProvider** is **true**, no dedicated identity provider will be deployed by the Operator and you will need to provide details about the external identity provider you are about to use. See also all the other fields starting with: **identityProvider**.
gatewayAuthenticationSidecarImage	Gateway sidecar responsible for authentication when NativeUserMode is enabled. See oauth2-proxy or openshift/oauth-proxy.
gatewayAuthorizationSidecarImage	Gateway sidecar responsible for authorization when NativeUserMode is enabled. See kube-rbac-proxy or openshift/kube-rbac-proxy
gatewayHeaderRewriteSidecarImage	Deprecated. The value of this flag is ignored. Sidecar functionality is now implemented in Traefik plugin.
identityProviderAdminUserName	Deprecated. The value of this flag is ignored. Overrides the name of the Identity Provider administrator user. Defaults to **admin**.
identityProviderClientId	Deprecated. The value of this flag is ignored. Name of a Identity provider, Keycloak or RH-SSO, **client-id** that is used for Che. Override this when an external Identity Provider is in use. See the **externalIdentityProvider** field. When omitted or left blank, it is set to the value of the **flavour** field suffixed with **-public**.
identityProviderContainerResources	Deprecated. The value of this flag is ignored. Identity provider container custom settings.
identityProviderImage	Deprecated. The value of this flag is ignored. Overrides the container image used in the Identity Provider, Keycloak or RH-SSO, deployment. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator.
identityProviderImagePullPolicy	Deprecated. The value of this flag is ignored. Overrides the image pull policy used in the Identity Provider, Keycloak or RH-SSO, deployment. Default value is **Always** for **nightly**, **next** or **latest** images, and **IfNotPresent** in other cases.
identityProviderIngress	Deprecated. The value of this flag is ignored. Ingress custom settings.

Property	Description
identityProviderPassword	Deprecated. The value of this flag is ignored. Overrides the password of Keycloak administrator user. Override this when an external Identity Provider is in use. See the **externalIdentityProvider** field. When omitted or left blank, it is set to an auto-generated password.
identityProviderPostgresPassword	Deprecated. The value of this flag is ignored. Password for a Identity Provider, Keycloak or RH-SSO, to connect to the database. Override this when an external Identity Provider is in use. See the **externalIdentityProvider** field. When omitted or left blank, it is set to an auto-generated password.
identityProviderPostgresSecret	Deprecated. The value of this flag is ignored. The secret that contains **password** for the Identity Provider, Keycloak or RH-SSO, to connect to the database. When the secret is defined, the **identityProviderPostgresPassword** is ignored. When the value is omitted or left blank, the one of following scenarios applies: 1. **identityProviderPostgresPassword** is defined, then it will be used to connect to the database. 2. **identityProviderPostgresPassword** is not defined, then a new secret with the name **che-identity-postgres-secret** will be created with an auto-generated value for **password**. The secret must have **app.kubernetes.io/part-of=che.eclipse.org** label.
identityProviderRealm	Deprecated. The value of this flag is ignored. Name of a Identity provider, Keycloak or RH-SSO, realm that is used for Che. Override this when an external Identity Provider is in use. See the **externalIdentityProvider** field. When omitted or left blank, it is set to the value of the **flavour** field.
identityProviderRoute	Deprecated. The value of this flag is ignored. Route custom settings.
identityProviderSecret	Deprecated. The value of this flag is ignored. The secret that contains **user** and **password** for Identity Provider. When the secret is defined, the **identityProviderAdminUserName** and **identityProviderPassword** are ignored. When the value is omitted or left blank, the one of following scenarios applies: 1. **identityProviderAdminUserName** and **identityProviderPassword** are defined, then they will be used. 2. **identityProviderAdminUserName** or **identityProviderPassword** are not defined, then a new secret with the name **che-identity-secret** will be created with default value **admin** for **user** and with an auto-generated value for **password**. The secret must have**app.kubernetes.io/part-of=che.eclipse.org** label.
identityProviderURL	Public URL of the Identity Provider server (Keycloak / RH-SSO server). Set this ONLY when a use of an external Identity Provider is needed. See the **externalIdentityProvider** field. By default, this will be automatically calculated and set by the Operator.

Property	Description
initialOpenShiftOAuthUser	Deprecated. The value of this flag is ignored. For operating with the OpenShift OAuth authentication, create a new user account since the kubeadmin can not be used. If the value is true, then a new OpenShift OAuth user will be created for the HTPasswd identity provider. If the value is false and the user has already been created, then it will be removed. If value is an empty, then do nothing. The user's credentials are stored in the **openshift-oauth-user-credentials** secret in 'openshift-config' namespace by Operator. Note that this solution is Openshift 4 platform-specific.
nativeUserMode	Deprecated. The value of this flag is ignored. Enables native user mode. Currently works only on OpenShift and DevWorkspace engine. Native User mode uses OpenShift OAuth directly as identity provider, without Keycloak.
oAuthClientName	Name of the OpenShift **OAuthClient** resource used to setup identity federation on the OpenShift side. Auto-generated when left blank. See also the **OpenShiftoAuth** field.
oAuthSecret	Name of the secret set in the OpenShift **OAuthClient** resource used to setup identity federation on the OpenShift side. Auto-generated when left blank. See also the **OAuthClientName** field.
openShiftoAuth	Deprecated. The value of this flag is ignored. Enables the integration of the identity provider (Keycloak / RHSSO) with OpenShift OAuth. Empty value on OpenShift by default. This will allow users to directly login with their OpenShift user through the OpenShift login, and have their workspaces created under personal OpenShift namespaces. WARNING: the **kubeadmin** user is NOT supported, and logging through it will NOT allow accessing the Che Dashboard.
updateAdminPassword	Deprecated. The value of this flag is ignored. Forces the default **admin** Che user to update password on first login. Defaults to **false**.

Table 3.4. CheCluster Custom Resource storage configuration settings related to persistent storage used by OpenShift Dev Spaces.

Property	Description
postgresPVCStorageClassName	Storage class for the Persistent Volume Claim dedicated to the PostgreSQL database. When omitted or left blank, a default storage class is used.
preCreateSubPaths	Instructs the Che server to start a special Pod to pre-create a sub-path in the Persistent Volumes. Defaults to **false**, however it will need to enable it according to the configuration of your Kubernetes cluster.
pvcClaimSize	Size of the persistent volume claim for workspaces. Defaults to **10Gi**.

Property	Description
pvcJobsImage	Overrides the container image used to create sub-paths in the Persistent Volumes. This includes the image tag. Omit it or leave it empty to use the default container image provided by the Operator. See also the **preCreateSubPaths** field.
pvcStrategy	Persistent volume claim strategy for the Che server. This Can be:`common` (all workspaces PVCs in one volume), **per-workspace** (one PVC per workspace for all declared volumes) and **unique** (one PVC per declared volume). Defaults to **common**.
workspacePVCStorageClassName	Storage class for the Persistent Volume Claims dedicated to the Che workspaces. When omitted or left blank, a default storage class is used.

Table 3.5. CheCluster Custom Resource k8s configuration settings specific to OpenShift Dev Spaces installations on OpenShift.

Property	Description
ingressClass	Ingress class that will define the which controller will manage ingresses. Defaults to **nginx**. NB: This drives the **kubernetes.io/ingress.class** annotation on Che-related ingresses.
ingressDomain	Global ingress domain for a Kubernetes cluster. This MUST be explicitly specified: there are no defaults.
ingressStrategy	Deprecated. The value of this flag is ignored. Strategy for ingress creation. Options are: **multi-host** (host is explicitly provided in ingress), **single-host** (host is provided, path-based rules) and **default-host** (no host is provided, path-based rules). Defaults to **multi-host** Deprecated in favor of **serverExposureStrategy** in the **server** section, which defines this regardless of the cluster type. When both are defined, the **serverExposureStrategy** option takes precedence.
securityContextFsGroup	The FSGroup in which the Che Pod and workspace Pods containers runs in. Default value is **1724**.
securityContextRunAsUser	ID of the user the Che Pod and workspace Pods containers run as. Default value is **1724**.
singleHostExposureType	Deprecated. The value of this flag is ignored. When the serverExposureStrategy is set to **single-host**, the way the server, registries and workspaces are exposed is further configured by this property. The possible values are **native**, which means that the server and workspaces are exposed using ingresses on K8s or **gateway** where the server and workspaces are exposed using a custom gateway based on Traefik. All the endpoints whether backed by the ingress or gateway **route** always point to the subpaths on the same domain. Defaults to **native**.

Property	Description
tlsSecretName	Name of a secret that will be used to setup ingress TLS termination when TLS is enabled. When the field is empty string, the default cluster certificate will be used. See also the **tlsSupport** field.

Table 3.6. CheCluster Custom Resource metrics settings, related to the OpenShift Dev Spaces metrics collection used by OpenShift Dev Spaces.

Property	Description
enable	Enables **metrics** the Che server endpoint. Default to **true**.

Table 3.7. CheCluster Custom Resource status defines the observed state of OpenShift Dev Spaces installation

Property	Description
cheClusterRunning	Status of a Che installation. Can be **Available**, **Unavailable**, or **Available, Rolling Update in Progress**.
cheURL	Public URL to the Che server.
cheVersion	Current installed Che version.
dbProvisioned	Indicates that a PostgreSQL instance has been correctly provisioned or not.
devfileRegistryURL	Public URL to the devfile registry.
devworkspaceStatus	The status of the Devworkspace subsystem
gitHubOAuthProvisioned	Indicates whether an Identity Provider instance, Keycloak or RH-SSO, has been configured to integrate with the GitHub OAuth.
helpLink	A URL that points to some URL where to find help related to the current Operator status.
keycloakProvisioned	Indicates whether an Identity Provider instance, Keycloak or RH-SSO, has been provisioned with realm, client and user.
keycloakURL	Public URL to the Identity Provider server, Keycloak or RH-SSO,.
message	A human readable message indicating details about why the Pod is in this condition.
openShiftOAuthUserCredentialsSecret	OpenShift OAuth secret in **openshift-config** namespace that contains user credentials for HTPasswd identity provider.

Property	Description
openShiftoAuthProvisioned	Indicates whether an Identity Provider instance, Keycloak or RH-SSO, has been configured to integrate with the OpenShift OAuth.
pluginRegistryURL	Public URL to the plugin registry.
reason	A brief CamelCase message indicating details about why the Pod is in this state.

3.2. CONFIGURING USER PROJECT PROVISIONING

For each user, OpenShift Dev Spaces isolates workspaces in a project. OpenShift Dev Spaces identifies the user project by the presence of labels and annotations. When starting a workspace, if the required project doesn't exist, OpenShift Dev Spaces creates the project using a template name.

You can modify OpenShift Dev Spaces behavior by:

- Section 3.2.1, "Configuring a user project name for automatic provisioning"
- Section 3.2.2, "Provisioning projects in advance"

3.2.1. Configuring a user project name for automatic provisioning

You can configure the project name template that OpenShift Dev Spaces uses to create the required project when starting a workspace.

A valid project name template follows these conventions:

- The **<username>** or **<userid>** placeholder is mandatory.
- Usernames and IDs cannot contain invalid characters. If the formatting of a username or ID is incompatible with the naming conventions for OpenShift objects, OpenShift Dev Spaces changes the username or ID to a valid name by replacing incompatible characters with the **-** symbol.
- OpenShift Dev Spaces evaluates the **<userid>** placeholder into a 14 character long string, and adds a random six character long suffix to prevent IDs from colliding. The result is stored in the user preferences for reuse.
- Kubernetes limits the length of a project name to 63 characters.
- OpenShift limits the length further to 49 characters.

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    workspaceNamespaceDefault: <workspace_namespace_template_>
```

Example 3.3. User workspaces project name template examples

User workspaces project name template	Resulting project example
<username>-devspaces (default)	user1-devspaces
<userid>-namespace	**cge1egvsb2nhba-namespace-ul1411**
<userid>-aka-<username>-namespace	**cgezegvsb2nhba-aka-user1-namespace-6m2w2b**

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.2.2. Provisioning projects in advance

You can provision workspaces projects in advance, rather than relying on automatic provisioning. Repeat the procedure for each user.

Procedure

- Create the *<project_name>* project for *<username>* user with the following labels and annotations:

```
kind: Namespace
apiVersion: v1
metadata:
  name: <project_name> 1
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-namespace
  annotations:
    che.eclipse.org/username: <username>
```

1 Use a project name of your choosing.

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.3. CONFIGURING SERVER COMPONENTS

3.3.1. Mounting a Secret or a ConfigMap as a file or an environment variable into a OpenShift Dev Spaces container

Secrets are OpenShift objects that store sensitive data such as:

- usernames
- passwords
- authentication tokens

in an encrypted form.

Users can mount a OpenShift Secret that contains sensitive data or a ConfigMap that contains configuration in a OpenShift Dev Spaces managed containers as:

- a file
- an environment variable

The mounting process uses the standard OpenShift mounting mechanism, but it requires additional annotations and labeling.

3.3.1.1. Mounting a Secret or a ConfigMap as a file into a OpenShift Dev Spaces container

Prerequisites

- A running instance of Red Hat OpenShift Dev Spaces.

Procedure

1. Create a new OpenShift Secret or a ConfigMap in the OpenShift project where a OpenShift Dev Spaces is deployed. The labels of the object that is about to be created must match the set of labels:
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>**
   - The **<DEPLOYMENT_NAME>** corresponds to the one following deployments:
     - **postgres**
     - **keycloak**
     - **devfile-registry**
     - **plugin-registry**
     - **devspaces**

       and

- **<OBJECT_KIND>** is either:
  - **secret**
    or
  - **configmap**

Example 3.4. Example:

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

or

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

Annotations must indicate that the given object is mounted as a file.

1. Configure the annotation values:
   - **che.eclipse.org/mount-as: file** - To indicate that a object is mounted as a file.
   - **che.eclipse.org/mount-path: *<TARGET_PATH>*** - To provide a required mount path.

Example 3.5. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
...
```

or

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

```
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
...
```

The OpenShift object may contain several items whose names must match the desired file name mounted into the container.

Example 3.6. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <base64 encoded data content here>
```

or

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <data content here>
```

This results in a file named **ca.crt** being mounted at the **/data** path of OpenShift Dev Spaces container.

IMPORTANT

To make the changes in a OpenShift Dev Spaces container visible, recreate the object entirely.

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”

- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.3.1.2. Mounting a Secret or a ConfigMap as an environment variable into a OpenShift Dev Spaces container

Prerequisites

- A running instance of Red Hat OpenShift Dev Spaces.

Procedure

1. Create a new OpenShift Secret or a ConfigMap in the OpenShift project where a OpenShift Dev Spaces is deployed. The labels of the object that is about to be created must match the set of labels:
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>**
   - The **<DEPLOYMENT_NAME>** corresponds to the one following deployments:
     - **postgres**
     - **keycloak**
     - **devfile-registry**
     - **plugin-registry**
     - **devspaces**
       and
   - **<OBJECT_KIND>** is either:
     - **secret**
       or
     - **configmap**

Example 3.7. Example:

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

or

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

```
    name: custom-settings
    labels:
      app.kubernetes.io/part-of: che.eclipse.org
      app.kubernetes.io/component: devspaces-configmap
  ...
```

Annotations must indicate that the given object is mounted as a environment variable.

1. Configure the annotation values:
   - **che.eclipse.org/mount-as: env** - to indicate that a object is mounted as an environment variable
   - **che.eclipse.org/env-name: *<FOO_ENV>*** - to provide an environment variable name, which is required to mount a object key value

Example 3.8. Example:

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

or

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    ...
data:
  mykey: myvalue
```

This results in two environment variables:

- **FOO_ENV**
- **myvalue**

being provisioned into a OpenShift Dev Spaces container.

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

Example 3.9. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    ...
data:
  mykey: __<base64 encoded data content here>__
  otherkey: __<base64 encoded data content here>__
```

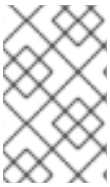
or

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    ...
data:
  mykey: __<data content here>__
  otherkey: __<data content here>__
```

This results in two environment variables:

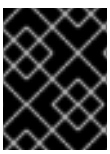
- **FOO_ENV**
- **OTHER_ENV**

being provisioned into a OpenShift Dev Spaces container.



NOTE

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



IMPORTANT

To make the changes in a OpenShift Dev Spaces container visible, recreate the object entirely.

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.3.2. Advanced configuration options for the OpenShift Dev Spaces server component

The following section describes advanced deployment and configuration methods for the OpenShift Dev Spaces server component.

3.3.2.1. Understanding OpenShift Dev Spaces server advanced configuration

The following section describes the OpenShift Dev Spaces server component advanced configuration method for a deployment.

Advanced configuration is necessary to:

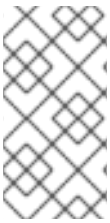
- Add environment variables not automatically generated by the Operator from the standard **CheCluster** Custom Resource fields.
- Override the properties automatically generated by the Operator from the standard **CheCluster** Custom Resource fields.

The **customCheProperties** field, part of the **CheCluster** Custom Resource **server** settings, contains a map of additional environment variables to apply to the OpenShift Dev Spaces server component.

Example 3.10. Override the default memory limit for workspaces

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    customCheProperties:
      CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB: "2048"
```



NOTE

Previous versions of the OpenShift Dev Spaces Operator had a ConfigMap named **custom** to fulfill this role. If the OpenShift Dev Spaces Operator finds a **configMap** with the name **custom**, it adds the data it contains into the **customCheProperties** field, redeploys OpenShift Dev Spaces, and deletes the **custom configMap**.

Additional resources

- Section 3.1.3, "**CheCluster** Custom Resource fields reference".
- Section 3.3.2.2, "OpenShift Dev Spaces server component system properties reference" .

3.3.2.2. OpenShift Dev Spaces server component system properties reference

The following document describes all possible configuration properties of the OpenShift Dev Spaces server component.

3.3.2.2.1. OpenShift Dev Spaces server

3.3.2.2.1.1. CHE_API

API service. Browsers initiate REST communications to OpenShift Dev Spaces server with this URL.

Default

http://${CHE_HOST}:${CHE_PORT}/api

3.3.2.2.1.2. CHE_API_INTERNAL

API service internal network URL. Back-end services should initiate REST communications to OpenShift Dev Spaces server with this URL

Default

NULL

3.3.2.2.1.3. CHE_WEBSOCKET_ENDPOINT

OpenShift Dev Spaces WebSocket major endpoint. Provides basic communication endpoint for major WebSocket interactions and messaging.

Default

ws://${CHE_HOST}:${CHE_PORT}/api/websocket

3.3.2.2.1.4. CHE_WEBSOCKET_INTERNAL_ENDPOINT

OpenShift Dev Spaces WebSocket major internal endpoint. Provides basic communication endpoint for major WebSocket interactions and messaging.

Default

NULL

3.3.2.2.1.5. CHE_WORKSPACE_PROJECTS_STORAGE

Your projects are synchronized from the OpenShift Dev Spaces server into the machine running each workspace. This is the directory in the machine where your projects are placed.

Default

/projects

3.3.2.2.1.6. CHE_WORKSPACE_PROJECTS_STORAGE_DEFAULT_SIZE

Used when OpenShift-type components in a devfile request project PVC creation (Applied in case of **unique** and **per workspace** PVC strategy. In case of the **common** PVC strategy, it is rewritten with the value of the **che.infra.kubernetes.pvc.quantity** property.)

Default

1Gi

3.3.2.2.1.7. CHE_WORKSPACE_LOGS_ROOT__DIR

Defines the directory inside the machine where all the workspace logs are placed. Provide this value into the machine, for example, as an environment variable. This is to ensure that agent developers can use this directory to back up agent logs.

Default

/workspace_logs

3.3.2.2.1.8. CHE_WORKSPACE_HTTP__PROXY

Configures environment variable HTTP_PROXY to a specified value in containers powering workspaces.

Default

empty

3.3.2.2.1.9. CHE_WORKSPACE_HTTPS__PROXY

Configures environment variable HTTPS_PROXY to a specified value in containers powering workspaces.

Default

empty

3.3.2.2.1.10. CHE_WORKSPACE_NO__PROXY

Configures environment variable NO_PROXY to a specified value in containers powering workspaces.

Default

empty

3.3.2.2.1.11. CHE_WORKSPACE_AUTO__START

By default, when users access a workspace with its URL, the workspace automatically starts (if currently stopped). Set this to **false** to disable this behavior.

Default

true

3.3.2.2.1.12. CHE_WORKSPACE_POOL_TYPE

Workspace threads pool configuration. This pool is used for workspace-related operations that require asynchronous execution, for example, starting and stopping. Possible values are **fixed** and **cached**.

Default

fixed

3.3.2.2.1.13. CHE_WORKSPACE_POOL_EXACT__SIZE

This property is ignored when pool type is different from **fixed**. It configures the exact size of the pool. When set, the **multiplier** property is ignored. If this property is not set (**0**, **<0**, **NULL**), then the pool size equals the number of cores. See also **che.workspace.pool.cores_multiplier**.

Default

30

3.3.2.2.1.14. CHE_WORKSPACE_POOL_CORES__MULTIPLIER

This property is ignored when pool type is not set to **fixed**, **che.workspace.pool.exact_size** is set. When set, the pool size is **N_CORES * multiplier**.

Default

2

3.3.2.2.1.15. CHE_WORKSPACE_PROBE__POOL__SIZE

This property specifies how many threads to use for workspace server liveness probes.

Default

10

3.3.2.2.1.16. CHE_WORKSPACE_HTTP__PROXY__JAVA__OPTIONS

HTTP proxy setting for workspace JVM.

Default

NULL

3.3.2.2.1.17. CHE_WORKSPACE_JAVA__OPTIONS

Java command-line options added to JVMs running in workspaces.

Default

-XX:MaxRAM=150m-XX:MaxRAMFraction=2 -XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true -Xms20m -Djava.security.egd=file:/dev/./urandom

3.3.2.2.1.18. CHE_WORKSPACE_MAVEN__OPTIONS

Maven command-line options added to JVMs running agents in workspaces.

Default

-XX:MaxRAM=150m-XX:MaxRAMFraction=2 -XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true -Xms20m -Djava.security.egd=file:/dev/./urandom

3.3.2.2.1.19. CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB

RAM limit default for each machine that has no RAM settings in its environment. Value less or equal to 0 is interpreted as disabling the limit.

Default

1024

3.3.2.2.1.20. CHE_WORKSPACE_DEFAULT__MEMORY__REQUEST__MB

RAM request for each container that has no explicit RAM settings in its environment. This amount is allocated when the workspace container is created. This property may not be supported by all infrastructure implementations. Currently it is supported by OpenShift. A memory request exceeding the memory limit is ignored, and only the limit size is used. Value less or equal to 0 is interpreted as disabling the limit.

Default

200

3.3.2.2.1.21. CHE_WORKSPACE_DEFAULT__CPU__LIMIT__CORES

CPU limit for each container that has no CPU settings in its environment. Specify either in floating point cores number, for example, **0.125**, or using the Kubernetes format, integer millicores, for example, **125m**. Value less or equal to 0 is interpreted as disabling the limit.

Default

-1

3.3.2.2.1.22. CHE_WORKSPACE_DEFAULT__CPU__REQUEST__CORES

CPU request for each container that has no CPU settings in environment. A CPU request exceeding the CPU limit is ignored, and only limit number is used. Value less or equal to 0 is interpreted as disabling the limit.

Default

-1

3.3.2.2.1.23. CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__LIMIT__MB

RAM limit for each sidecar that has no RAM settings in the OpenShift Dev Spaces plug-in configuration. Value less or equal to 0 is interpreted as disabling the limit.

Default

128

3.3.2.2.1.24. CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__REQUEST__MB

RAM request for each sidecar that has no RAM settings in the OpenShift Dev Spaces plug-in configuration.

Default

64

3.3.2.2.1.25. CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES

CPU limit default for each sidecar that has no CPU settings in the OpenShift Dev Spaces plug-in configuration. Specify either in floating point cores number, for example, **0.125**, or using the Kubernetes format, integer millicores, for example, **125m**. Value less or equal to 0 is interpreted as disabling the limit.

Default

-1

3.3.2.2.1.26. CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES

CPU request default for each sidecar that has no CPU settings in the OpenShift Dev Spaces plug-in configuration. Specify either in floating point cores number, for example, **0.125**, or using the Kubernetes format, integer millicores, for example, **125m**.

Default

-1

3.3.2.2.1.27. CHE_WORKSPACE_SIDECAR_IMAGE__PULL__POLICY

Defines image-pulling strategy for sidecars. Possible values are: **Always**, **Never**, **IfNotPresent**. For any other value, **Always** is assumed for images with the **:latest** tag, or **IfNotPresent** for all other cases.

Default

Always

3.3.2.2.1.28. CHE_WORKSPACE_ACTIVITY__CHECK__SCHEDULER__PERIOD__S

Period of inactive workspaces suspend job execution.

Default

60

3.3.2.2.1.29. CHE_WORKSPACE_ACTIVITY__CLEANUP__SCHEDULER__PERIOD__S

The period of the cleanup of the activity table. The activity table can contain invalid or stale data if some unforeseen errors happen, as a server failure at a peculiar point in time. The default is to run the cleanup job every hour.

Default

3600

3.3.2.2.1.30. CHE_WORKSPACE_ACTIVITY__CLEANUP__SCHEDULER__INITIAL__DELAY__S

The delay after server startup to start the first activity clean up job.

Default

60

3.3.2.2.1.31. CHE_WORKSPACE_ACTIVITY__CHECK__SCHEDULER__DELAY__S

Delay before first workspace idleness check job started to avoid mass suspend if OpenShift Dev Spaces server was unavailable for period close to inactivity timeout.

Default

180

3.3.2.2.1.32. CHE_WORKSPACE_CLEANUP__TEMPORARY__INITIAL__DELAY__MIN

Time to delay the first execution of temporary workspaces cleanup job.

Default

5

3.3.2.2.1.33. CHE_WORKSPACE_CLEANUP__TEMPORARY__PERIOD__MIN

Time to delay between the termination of one execution and the commencement of the next execution of temporary workspaces cleanup job

Default

180

3.3.2.2.1.34. CHE_WORKSPACE_SERVER_PING__SUCCESS__THRESHOLD

Number of sequential successful pings to server after which it is treated as available. the OpenShift Dev Spaces Operator: the property is common for all servers, for example, workspace agent, terminal, exec.

Default

1

3.3.2.2.1.35. CHE_WORKSPACE_SERVER_PING__INTERVAL__MILLISECONDS

Interval, in milliseconds, between successive pings to workspace server.

Default

3000

3.3.2.2.1.36. CHE_WORKSPACE_SERVER_LIVENESS__PROBES

List of servers names which require liveness probes

Default

wsagent/http,exec-agent/http,terminal,theia,jupyter,dirigible,cloud-shell,intellij

3.3.2.2.1.37. CHE_WORKSPACE_STARTUP__DEBUG__LOG__LIMIT__BYTES

Limit size of the logs collected from single container that can be observed by che-server when debugging workspace startup. default 10MB=10485760

Default

10485760

3.3.2.2.1.38. CHE_WORKSPACE_STOP_ROLE_ENABLED

If true, 'stop-workspace' role with the edit privileges will be granted to the 'che' ServiceAccount if OpenShift OAuth is enabled. This configuration is mainly required for workspace idling when the OpenShift OAuth is enabled.

Default

true

3.3.2.2.1.39. CHE_DEVWORKSPACES_ENABLED

Specifies whether OpenShift Dev Spaces is deployed with DevWorkspaces enabled. This property is set by the OpenShift Dev Spaces Operator if it also installed the support for DevWorkspaces. This property is used to advertise this fact to the OpenShift Dev Spaces dashboard. It does not make sense to change the value of this property manually.

Default

false

3.3.2.2.2. Authentication parameters

3.3.2.2.2.1. CHE_AUTH_USER__SELF__CREATION

OpenShift Dev Spaces has a single identity implementation, so this does not change the user experience. If true, enables user creation at API level

Default

false

3.3.2.2.2.2. CHE_AUTH_ACCESS__DENIED__ERROR__PAGE

Authentication error page address

Default

/error-oauth

3.3.2.2.2.3. CHE_AUTH_RESERVED__USER__NAMES

Reserved user names

Default

empty

3.3.2.2.2.4. CHE_OAUTH2_GITHUB_CLIENTID__FILEPATH

Configuration of GitHub OAuth2 client. Used to obtain Personal access tokens. Location of the file with GitHub client id.

Default

NULL

3.3.2.2.2.5. CHE_OAUTH2_GITHUB_CLIENTSECRET__FILEPATH

Location of the file with GitHub client secret.

Default

NULL

3.3.2.2.2.6. CHE_OAUTH_GITHUB_AUTHURI

GitHub OAuth authorization URI.

Default

https://github.com/login/oauth/authorize

3.3.2.2.2.7. CHE_OAUTH_GITHUB_TOKENURI

GitHub OAuth token URI.

Default

https://github.com/login/oauth/access_token

3.3.2.2.2.8. CHE_OAUTH_GITHUB_REDIRECTURIS

GitHub OAuth redirect URIs. Separate multiple values with comma, for example: URI,URI,URI

Default

http://localhost:${CHE_PORT}/api/oauth/callback

3.3.2.2.2.9. CHE_OAUTH_OPENSHIFT_CLIENTID

Configuration of OpenShift OAuth client. Used to obtain OpenShift OAuth token. OpenShift OAuth client ID.

Default

NULL

3.3.2.2.2.10. CHE_OAUTH_OPENSHIFT_CLIENTSECRET

Configurationof OpenShift OAuth client. Used to obtain OpenShift OAuth token. OpenShift OAuth client ID. OpenShift OAuth client secret.

Default

NULL

3.3.2.2.2.11. CHE_OAUTH_OPENSHIFT_OAUTH__ENDPOINT

ConfigurationofOpenShift OAuth client. Used to obtain OpenShift OAuth token. OpenShift OAuth client ID. OpenShift OAuth client secret. OpenShift OAuth endpoint.

Default

NULL

3.3.2.2.2.12. CHE_OAUTH_OPENSHIFT_VERIFY__TOKEN__URL

ConfigurationofOpenShiftOAuth client. Used to obtain OpenShift OAuth token. OpenShift OAuth client ID. OpenShift OAuth client secret. OpenShift OAuth endpoint. OpenShift OAuth verification token URL.

Default

NULL

3.3.2.2.2.13. CHE_OAUTH1_BITBUCKET_CONSUMERKEYPATH

Configuration of Bitbucket Server OAuth1 client. Used to obtain Personal access tokens. Location of the file with Bitbucket Server application consumer key (equivalent to a username).

Default

NULL

3.3.2.2.2.14. CHE_OAUTH1_BITBUCKET_PRIVATEKEYPATH

Configurationof Bitbucket Server OAuth1 client. Used to obtain Personal access tokens. Location of the file with Bitbucket Server application consumer key (equivalent to a username). Location of the file with Bitbucket Server application private key

Default

NULL

3.3.2.2.2.15. CHE_OAUTH1_BITBUCKET_ENDPOINT

ConfigurationofBitbucket Server OAuth1 client. Used to obtain Personal access tokens. Location of the file with Bitbucket Server application consumer key (equivalent to a username). Location of the file with Bitbucket Server application private key Bitbucket Server URL. To work correctly with factories the same URL has to be part of **che.integration.bitbucket.server_endpoints** too.

Default

NULL

3.3.2.2.3. Internal

3.3.2.2.3.1. SCHEDULE_CORE__POOL__SIZE

OpenShift Dev Spaces extensions can be scheduled executions on a time basis. This configures the size of the thread pool allocated to extensions that are launched on a recurring schedule.

Default

10

3.3.2.2.3.2. DB_SCHEMA_FLYWAY_BASELINE_ENABLED

DB initialization and migration configuration If true, ignore scripts up to the version configured by baseline.version.

Default

true

3.3.2.2.3.3. DB_SCHEMA_FLYWAY_BASELINE_VERSION

Scripts with version up to this are ignored. Note that scripts with version equal to baseline version are also ignored.

Default

5.0.0.8.1

3.3.2.2.3.4. DB_SCHEMA_FLYWAY_SCRIPTS_PREFIX

Prefix of migration scripts.

Default

empty

3.3.2.2.3.5. DB_SCHEMA_FLYWAY_SCRIPTS_SUFFIX

Suffix of migration scripts.

Default

.sql

3.3.2.2.3.6. DB_SCHEMA_FLYWAY_SCRIPTS_VERSION__SEPARATOR

Separator of version from the other part of script name.

Default

__

3.3.2.2.3.7. DB_SCHEMA_FLYWAY_SCRIPTS_LOCATIONS

Locations where to search migration scripts.

Default

classpath:che-schema

3.3.2.2.4. Kubernetes Infra parameters

3.3.2.2.4.1. CHE_INFRA_KUBERNETES_MASTER__URL

Configuration of Kubernetes client master URL that Infra will use.

Default

empty

3.3.2.2.4.2. CHE_INFRA_KUBERNETES_TRUST__CERTS

Boolean to configure Kubernetes client to use trusted certificates.

Default

false

3.3.2.2.4.3. CHE_INFRA_KUBERNETES_CLUSTER__DOMAIN

Kubernetes cluster domain. If not set, svc names will not contain information about the cluster domain.

Default

NULL

3.3.2.2.4.4. CHE_INFRA_KUBERNETES_SERVER__STRATEGY

Defines the way how servers are exposed to the world in Kubernetes infra. List of strategies implemented in OpenShift Dev Spaces: **default-host**, **multi-host**, **single-host**.

Default

multi-host

3.3.2.2.4.5. CHE_INFRA_KUBERNETES_SINGLEHOST_WORKSPACE_EXPOSURE

Defines the way in which the workspace plugins and editors are exposed in the single-host mode. Supported exposures: **native**: Exposes servers using Kubernetes Ingresses. Works only on Kubernetes. **gateway**: Exposes servers using reverse-proxy gateway.

Default

native

3.3.2.2.4.6. CHE_INFRA_KUBERNETES_SINGLEHOST_WORKSPACE_DEVFILE__ENDPOINT__EXPOSURE

Defines the way how to expose devfile endpoints, as end-user's applications, in single-host server strategy. They can either follow the single-host strategy and be exposed on subpaths, or they can be exposed on subdomains. **multi-host**: expose on subdomains **single-host**: expose on subpaths

Default

multi-host

3.3.2.2.4.7. CHE_INFRA_KUBERNETES_SINGLEHOST_GATEWAY_CONFIGMAP__LABELS

Defines labels which will be set to ConfigMaps configuring single-host gateway.

Default

app=che,component=che-gateway-config

3.3.2.2.4.8. CHE_INFRA_KUBERNETES_INGRESS_DOMAIN

Used to generate domain for a server in a workspace in case property **che.infra.kubernetes.server_strategy** is set to **multi-host**

Default

empty

3.3.2.2.4.9. CHE_INFRA_KUBERNETES_NAMESPACE_CREATION__ALLOWED

Indicates whether OpenShift Dev Spaces server is allowed to create project for user workspaces, or they're intended to be created manually by cluster administrator. This property is also used by the OpenShift infra.

Default

true

3.3.2.2.4.10. CHE_INFRA_KUBERNETES_NAMESPACE_DEFAULT

Defines Kubernetes default namespace in which user's workspaces are created if user does not override it. It's possible to use **<username>** and **<userid>** placeholders (for example: **che-workspace-<username>**). In that case, new namespace will be created for each user. Used by OpenShift infra as well to specify a Project. The **<username>** or **<userid>** placeholder is mandatory.

Default

<username>-che

3.3.2.2.4.11. CHE_INFRA_KUBERNETES_NAMESPACE_LABEL

Defines whether che-server should try to label the workspace namespaces. NOTE: It is strongly recommended to keep the value of this property set to true. If false, the new workspace namespaces will not be labeled automatically and therefore not recognized by the OpenShift Dev Spaces operator making some features of DevWorkspaces not working. If false, an administrator is required to label the namespaces manually using the labels specified in che.infra.kubernetes.namespace.labels. If you want to manage the namespaces yourself, make sure to follow https://www.eclipse.org/che/docs/stable/administration-guide/provisioning-namespaces-in-advance/. Any additional labels present on the namespace are kept in place and do not affect the functionality. Also note that the the administrator is free to pre-create and label the namespaces manually even if this property is true. No updates to the namespaces are done if they already conform to the labeling requirements.

Default

true

3.3.2.2.4.12. CHE_INFRA_KUBERNETES_NAMESPACE_ANNOTATE

Defines whether che-server should try to annotate the workspace namespaces.

Default

true

3.3.2.2.4.13. CHE_INFRA_KUBERNETES_NAMESPACE_LABELS

List of labels to find project that are used for OpenShift Dev Spaces Workspaces. They are used to: - find prepared project for users in combination with **che.infra.kubernetes.namespace.annotations**. - actively label project with any workspace. NOTE: It is strongly recommended not to change the value of this property because the OpenShift Dev Spaces operator relies on these labels and their precise values when reconciling DevWorkspaces. If this configuration is changed, the namespaces will not be automatically recognized by the OpenShift Dev Spaces operator as workspace namespaces unless manually labeled as such using the default labels and values. Additional labels on the namespace do not affect the functionality.

Default

app.kubernetes.io/part-of=che.eclipse.org,app.kubernetes.io/component=workspaces-namespace

3.3.2.2.4.14. CHE_INFRA_KUBERNETES_NAMESPACE_ANNOTATIONS

List of annotations to find project prepared for OpenShift Dev Spaces users workspaces. Only project matching the **che.infra.kubernetes.namespace.labels** will be matched against these annotations. project that matches both **che.infra.kubernetes.namespace.labels** and **che.infra.kubernetes.namespace.annotations** will be preferentially used for User's workspaces. It's possible to use **<username>** placeholder to specify the project to concrete user. They are used to: - find prepared project for users in combination with **che.infra.kubernetes.namespace.labels**. - actively annotate project with any workspace.

Default

che.eclipse.org/username=<username>

3.3.2.2.4.15. CHE_INFRA_KUBERNETES_SERVICE__ACCOUNT__NAME

Defines Kubernetes Service Account name which should be specified to be bound to all workspaces Pods. the OpenShift Dev Spaces Operator that Kubernetes Infrastructure will not create the service

account and it should exist. OpenShift infrastructure will check if project is predefined(if **che.infra.openshift.project** is not empty): - if it is predefined then service account must exist there - if it is 'NULL' or empty string then infrastructure will create new OpenShift project per workspace and prepare workspace service account with needed roles there

Default

NULL

3.3.2.2.4.16. CHE_INFRA_KUBERNETES_WORKSPACE__SA__CLUSTER__ROLES

Specifies optional, additional cluster roles to use with the workspace service account. the OpenShift Dev Spaces Operator that the cluster role names must already exist, and the OpenShift Dev Spaces service account needs to be able to create a Role Binding to associate these cluster roles with the workspace service account. The names are comma separated. This property deprecates **che.infra.kubernetes.cluster_role_name**.

Default

NULL

3.3.2.2.4.17. CHE_INFRA_KUBERNETES_USER__CLUSTER__ROLES

Cluster roles to assign to user in his namespace

Default

NULL

3.3.2.2.4.18. CHE_INFRA_KUBERNETES_WORKSPACE__START__TIMEOUT__MIN

Defines wait time that limits the Kubernetes workspace start time.

Default

8

3.3.2.2.4.19. CHE_INFRA_KUBERNETES_INGRESS__START__TIMEOUT__MIN

Defines the timeout in minutes that limits the period for which Kubernetes Ingress become ready

Default

5

3.3.2.2.4.20. CHE_INFRA_KUBERNETES_WORKSPACE__UNRECOVERABLE__EVENTS

If during workspace startup an unrecoverable event defined in the property occurs, stop the workspace immediately rather than waiting until timeout. the OpenShift Dev Spaces Operator that this SHOULD NOT include a mere "Failed" reason, because that might catch events that are not unrecoverable. A failed container startup is handled explicitly by OpenShift Dev Spaces server.

Default

FailedMount,FailedScheduling,MountVolume.SetUpfailed,Failed to pull image,FailedCreate,ReplicaSetCreateError

3.3.2.2.4.21. CHE_INFRA_KUBERNETES_PVC_ENABLED

Defines whether use the Persistent Volume Claim for OpenShift Dev Spaces workspace needs, for example: backup projects, logs, or disable it.

Default

true

3.3.2.2.4.22. CHE_INFRA_KUBERNETES_PVC_STRATEGY

Defined which strategy will be used while choosing PVC for workspaces. Supported strategies: **common**: All workspaces in the same project will reuse the same PVC. Name of PVC may be configured with **che.infra.kubernetes.pvc.name**. Existing PVC will be used or a new one will be created if it does not exist. **unique**: Separate PVC for each workspace's volume will be used. Name of PVC is evaluated as **'{che.infra.kubernetes.pvc.name} + '-' + {generated_8_chars}'**. Existing PVC will be used or a new one will be created if it does not exist. **per-workspace**: Separate PVC for each workspace will be used. Name of PVC is evaluated as **'{che.infra.kubernetes.pvc.name} + '-' + {WORKSPACE_ID}'**. Existing PVC will be used or a new one will be created if it doesn't exist.

Default

common

3.3.2.2.4.23. CHE_INFRA_KUBERNETES_PVC_PRECREATE__SUBPATHS

Defines whether to run a job that creates workspace's subpath directories in persistent volume for the **common** strategy before launching a workspace. Necessary in some versions of OpenShift as workspace subpath volume mounts are created with root permissions, and therefore cannot be modified by workspaces running as a user (presents an error importing projects into a workspace in OpenShift Dev Spaces). The default is **true**, but should be set to **false** if the version of OpenShift creates subdirectories with user permissions. See: subPath in volumeMount is not writable for non-root users #41638 the OpenShift Dev Spaces Operator that this property has effect only if the **common** PVC strategy used.

Default

true

3.3.2.2.4.24. CHE_INFRA_KUBERNETES_PVC_NAME

Defines the settings of PVC name for OpenShift Dev Spaces workspaces. Each PVC strategy supplies this value differently. See documentation for **che.infra.kubernetes.pvc.strategy** property

Default

claim-che-workspace

3.3.2.2.4.25. CHE_INFRA_KUBERNETES_PVC_STORAGE__CLASS__NAME

Defines the storage class of Persistent Volume Claim for the workspaces. Empty strings means "use default".

Default

empty

3.3.2.2.4.26. CHE_INFRA_KUBERNETES_PVC_QUANTITY

Defines the size of Persistent Volume Claim of OpenShift Dev Spaces workspace. See: Understanding persistent storage

Default

10Gi

3.3.2.2.4.27. CHE_INFRA_KUBERNETES_PVC_JOBS_IMAGE

Pod that is launched when performing persistent volume claim maintenance jobs on OpenShift

Default

registry.access.redhat.com/ubi8-minimal:8.3-230

3.3.2.2.4.28. CHE_INFRA_KUBERNETES_PVC_JOBS_IMAGE_PULL__POLICY

Image pull policy of container that used for the maintenance jobs on OpenShift cluster

Default

IfNotPresent

3.3.2.2.4.29. CHE_INFRA_KUBERNETES_PVC_JOBS_MEMORYLIMIT

Defines Pod memory limit for persistent volume claim maintenance jobs

Default

250Mi

3.3.2.2.4.30. CHE_INFRA_KUBERNETES_PVC_ACCESS__MODE

Defines Persistent Volume Claim access mode. the OpenShift Dev Spaces Operator that for common PVC strategy changing of access mode affects the number of simultaneously running workspaces. If the OpenShift instance running OpenShift Dev Spaces is using Persistent Volumes with RWX access mode, then a limit of running workspaces at the same time is bounded only by OpenShift Dev Spaces limits configuration: RAM, CPU, and so on. See: Understanding persistent storage

Default

ReadWriteOnce

3.3.2.2.4.31. CHE_INFRA_KUBERNETES_PVC_WAIT__BOUND

Defines if OpenShift Dev Spaces Server should wait workspaces Persistent Volume Claims to become bound after creating. Default value is **true**. The parameter is used by all Persistent Volume Claim strategies. It should be set to **false** when **volumeBindingMode** is configured to **WaitForFirstConsumer** otherwise workspace starts will hangs up on phase of waiting PVCs.

Default

true

3.3.2.2.4.32. CHE_INFRA_KUBERNETES_INGRESS_ANNOTATIONS__JSON

Defines annotations for ingresses which are used for servers exposing. Value depends on the kind of ingress controller. OpenShift infrastructure ignores this property because it uses Routes rather than Ingresses. the OpenShift Dev Spaces Operator that for a single-host deployment strategy to work, a controller supporting URL rewriting has to be used (so that URLs can point to different servers while the servers do not need to support changing the app root). The **che.infra.kubernetes.ingress.path.rewrite_transform** property defines how the path of the ingress

should be transformed to support the URL rewriting and this property defines the set of annotations on the ingress itself that instruct the chosen ingress controller to actually do the URL rewriting, potentially building on the path transformation (if required by the chosen ingress controller). For example for Nginx ingress controller 0.22.0 and later the following value is recommended: **{"ingress.kubernetes.io/rewrite-target": "/$1","ingress.kubernetes.io/ssl-redirect": "false",\ "ingress.kubernetes.io/proxy-connect-timeout": "3600","ingress.kubernetes.io/proxy-read-timeout": "3600", "nginx.org/websocket-services": "<service-name>"}** and the **che.infra.kubernetes.ingress.path.rewrite_transform** should be set to **"%s(.*)"**. For nginx ingress controller older than 0.22.0, the rewrite-target should be set to merely / and the path transform to **%s** (see the **che.infra.kubernetes.ingress.path.rewrite_transform** property). See the Nginx ingress controller documentation for the explanation of how the ingress controller uses the regular expression available in the ingress path and how it achieves the URL rewriting.

Default

NULL

3.3.2.2.4.33. CHE_INFRA_KUBERNETES_INGRESS_PATH__TRANSFORM

Defines a recipe on how to declare the path of the ingress that should expose a server. The **%s** represents the base public URL of the server and is guaranteed to end with a forward slash. This property must be a valid input to the **String.format()** method and contain exactly one reference to **%s**. See the description of the **che.infra.kubernetes.ingress.annotations_json** property to see how these two properties interplay when specifying the ingress annotations and path. If not defined, this property defaults to **%s** (without the quotes) which means that the path is not transformed in any way for use with the ingress controller.

Default

NULL

3.3.2.2.4.34. CHE_INFRA_KUBERNETES_INGRESS_LABELS

Additional labels to add into every Ingress created by OpenShift Dev Spaces server to allow clear identification.

Default

NULL

3.3.2.2.4.35. CHE_INFRA_KUBERNETES_POD_SECURITY__CONTEXT_RUN__AS__USER

Defines security context for Pods that will be created by Kubernetes Infra This is ignored by OpenShift infra

Default

NULL

3.3.2.2.4.36. CHE_INFRA_KUBERNETES_POD_SECURITY__CONTEXT_FS__GROUP

Defines security context for Pods that will be created by Kubernetes Infra. A special supplemental group that applies to all containers in a Pod. This is ignored by OpenShift infra.

Default

NULL

3.3.2.2.4.37. CHE_INFRA_KUBERNETES_POD_TERMINATION__GRACE__PERIOD__SEC

Defines grace termination period for Pods that will be created by OpenShift infrastructures. Default value: **0**. It allows to stop Pods quickly and significantly decrease the time required for stopping a workspace. the OpenShift Dev Spaces Operator: if **terminationGracePeriodSeconds** have been explicitly set in OpenShift recipe it will not be overridden.

Default

0

3.3.2.2.4.38. CHE_INFRA_KUBERNETES_TLS__ENABLED

Creates Ingresses with Transport Layer Security (TLS) enabled. In OpenShift infrastructure, Routes will be TLS-enabled.

Default

false

3.3.2.2.4.39. CHE_INFRA_KUBERNETES_TLS__SECRET

Name of a secret that should be used when creating workspace ingresses with TLS. This property is ignored by OpenShift infrastructure.

Default

empty

3.3.2.2.4.40. CHE_INFRA_KUBERNETES_TLS__KEY

Data for TLS Secret that should be used for workspaces Ingresses. **cert** and **key** should be encoded with Base64 algorithm. These properties are ignored by OpenShift infrastructure.

Default

NULL

3.3.2.2.4.41. CHE_INFRA_KUBERNETES_TLS__CERT

Certificate data for TLS Secret that should be used for workspaces Ingresses. Certificate should be encoded with Base64 algorithm. This property is ignored by OpenShift infrastructure.

Default

NULL

3.3.2.2.4.42. CHE_INFRA_KUBERNETES_RUNTIMES__CONSISTENCY__CHECK__PERIOD__MIN

Defines the period with which runtimes consistency checks will be performed. If runtime has inconsistent state then runtime will be stopped automatically. Value must be more than 0 or **-1**, where **-1** means that checks won't be performed at all. It is disabled by default because there is possible OpenShift Dev Spaces Server configuration when OpenShift Dev Spaces Server doesn't have an ability to interact with Kubernetes API when operation is not invoked by user. It DOES work on the following configurations: - workspaces objects are created in the same namespace where OpenShift Dev Spaces Server is located; - **cluster-admin** service account token is mounted to OpenShift Dev Spaces Server Pod. It DOES NOT work on the following configurations: - OpenShift Dev Spaces Server communicates with Kubernetes API using token from OAuth provider.

Default

-1

3.3.2.2.4.43. CHE_INFRA_KUBERNETES_TRUSTED__CA_SRC__CONFIGMAP

Name of the ConfigMap in OpenShift Dev Spaces server namespace with additional CA TLS certificates to be propagated into all user's workspaces. If the property is set on OpenShift 4 infrastructure, and **che.infra.openshift.trusted_ca.dest_configmap_labels** includes the **config.openshift.io/inject-trusted-cabundle=true** label, then cluster CA bundle will be propagated too.

Default

NULL

3.3.2.2.4.44. CHE_INFRA_KUBERNETES_TRUSTED__CA_DEST__CONFIGMAP

Name of the ConfigMap in a workspace namespace with additional CA TLS certificates. Holds the copy of **che.infra.kubernetes.trusted_ca.src_configmap** but in a workspace namespace. Content of this ConfigMap is mounted into all workspace containers including plugin brokers. Do not change the ConfigMap name unless it conflicts with the already existing ConfigMap. the OpenShift Dev Spaces Operator that the resulting ConfigMap name can be adjusted eventually to make it unique in project. The original name would be stored in **che.original_name** label.

Default

ca-certs

3.3.2.2.4.45. CHE_INFRA_KUBERNETES_TRUSTED__CA_MOUNT__PATH

Configures path on workspace containers where the CA bundle should be mounted. Content of ConfigMap specified by **che.infra.kubernetes.trusted_ca.dest_configmap** is mounted.

Default

/**public-certs**

3.3.2.2.4.46. CHE_INFRA_KUBERNETES_TRUSTED__CA_DEST__CONFIGMAP__LABELS

Comma separated list of labels to add to the CA certificates ConfigMap in user workspace. See the **che.infra.kubernetes.trusted_ca.dest_configmap** property.

Default

empty

3.3.2.2.5. OpenShift Infra parameters

3.3.2.2.5.1. CHE_INFRA_OPENSHIFT_TRUSTED__CA_DEST__CONFIGMAP__LABELS

Comma separated list of labels to add to the CA certificates ConfigMap in user workspace. See **che.infra.kubernetes.trusted_ca.dest_configmap** property. This default value is used for automatic cluster CA bundle injection in OpenShift 4.

Default

config.openshift.io/inject-trusted-cabundle=true

3.3.2.2.5.2. CHE_INFRA_OPENSHIFT_ROUTE_LABELS

Additional labels to add into every Route created by OpenShift Dev Spaces server to allow clear identification.

Default

NULL

3.3.2.2.5.3. CHE_INFRA_OPENSHIFT_ROUTE_HOST_DOMAIN__SUFFIX

The hostname that should be used as a suffix for the workspace routes. For example: Using **domain_suffix=*<devspaces-__<openshift_deployment_name>__.__<domain_name>__>***, the route resembles: **routed3qrtk.*<devspaces-__<openshift_deployment_name>__.__<domain_name>__>***. It has to be a valid DNS name.

Default

NULL

3.3.2.2.5.4. CHE_INFRA_OPENSHIFT_PROJECT_INIT__WITH__SERVER__SA

Initialize OpenShift project with OpenShift Dev Spaces server's service account if OpenShift OAuth is enabled.

Default

true

3.3.2.2.6. Experimental properties

3.3.2.2.6.1. CHE_WORKSPACE_PLUGIN__BROKER_METADATA_IMAGE

Docker image of OpenShift Dev Spaces plugin broker app that resolves workspace tools configuration and copies plugins dependencies to a workspace. The OpenShift Dev Spaces Operator overrides these images by default. Changing the images here will not have an effect if OpenShift Dev Spaces is installed using the Operator.

Default

quay.io/eclipse/che-plugin-metadata-broker:v3.4.0

3.3.2.2.6.2. CHE_WORKSPACE_PLUGIN__BROKER_ARTIFACTS_IMAGE

Docker image of OpenShift Dev Spaces plugin artifacts broker. This broker runs as an init container on the workspace Pod. Its job is to take in a list of plugin identifiers (either references to a plugin in the registry or a link to a plugin meta.yaml) and ensure that the correct .vsix and .theia extensions are downloaded into the /plugins directory, for each plugin requested for the workspace.

Default

quay.io/eclipse/che-plugin-artifacts-broker:v3.4.0

3.3.2.2.6.3. CHE_WORKSPACE_PLUGIN__BROKER_DEFAULT__MERGE__PLUGINS

Configures the default behavior of the plugin brokers when provisioning plugins into a workspace. If set to true, the plugin brokers will attempt to merge plugins when possible: they run in the same sidecar image and do not have conflicting settings. This value is the default setting used when the devfile does

not specify the **mergePlugins** attribute.

Default

false

3.3.2.2.6.4. CHE_WORKSPACE_PLUGIN__BROKER_PULL__POLICY

Docker image of OpenShift Dev Spaces plugin broker app that resolves workspace tools configuration and copies plugins dependencies to a workspace

Default

Always

3.3.2.2.6.5. CHE_WORKSPACE_PLUGIN__BROKER_WAIT__TIMEOUT__MIN

Defines the timeout in minutes that limits the max period of result waiting for plugin broker.

Default

3

3.3.2.2.6.6. CHE_WORKSPACE_PLUGIN__REGISTRY__URL

Workspace plug-ins registry endpoint. Should be a valid HTTP URL. Example: http://che-plugin-registry-eclipse-che.192.168.65.2.nip.io In case OpenShift Dev Spaces plug-ins registry is not needed value 'NULL' should be used

Default

https://che-plugin-registry.prod-preview.openshift.io/v3

3.3.2.2.6.7. CHE_WORKSPACE_PLUGIN__REGISTRY__INTERNAL__URL

Workspace plugins registry internal endpoint. Should be a valid HTTP URL. Example: http://devfile-registry.che.svc.cluster.local:8080 In case OpenShift Dev Spaces plug-ins registry is not needed value 'NULL' should be used

Default

NULL

3.3.2.2.6.8. CHE_WORKSPACE_DEVFILE__REGISTRY__URL

Devfile Registry endpoint. Should be a valid HTTP URL. Example: http://che-devfile-registry-eclipse-che.192.168.65.2.nip.io In case OpenShift Dev Spaces plug-ins registry is not needed value 'NULL' should be used

Default

https://che-devfile-registry.prod-preview.openshift.io/

3.3.2.2.6.9. CHE_WORKSPACE_DEVFILE__REGISTRY__INTERNAL__URL

Devfile Registry "internal" endpoint. Should be a valid HTTP URL. Example: http://plugin-registry.che.svc.cluster.local:8080 In case OpenShift Dev Spaces plug-ins registry is not needed value 'NULL' should be used

Default

NULL

3.3.2.2.6.10. CHE_WORKSPACE_STORAGE_AVAILABLE__TYPES

The configuration property that defines available values for storage types that clients such as the Dashboard should propose to users during workspace creation and update. Available values: - **persistent**: Persistent Storage slow I/O but persistent. - **ephemeral**: Ephemeral Storage allows for faster I/O but may have limited storage and is not persistent. - **async**: Experimental feature: Asynchronous storage is combination of Ephemeral and Persistent storage. Allows for faster I/O and keep your changes, will backup on stop and restore on start workspace. Will work only if: - **che.infra.kubernetes.pvc.strategy='common'** - **che.limits.user.workspaces.run.count=1** - **che.infra.kubernetes.namespace.default** contains **<username>** in other cases remove **async** from the list.

Default

persistent,ephemeral,async

3.3.2.2.6.11. CHE_WORKSPACE_STORAGE_PREFERRED__TYPE

The configuration property that defines a default value for storage type that clients such as the Dashboard should propose to users during workspace creation and update. The **async** value is an experimental feature, not recommended as default type.

Default

persistent

3.3.2.2.6.12. CHE_SERVER_SECURE__EXPOSER

Configures in which way secure servers will be protected with authentication. Suitable values: - **default**: **jwtproxy** is configured in a pass-through mode. Servers should authenticate requests themselves. - **jwtproxy**: **jwtproxy** will authenticate requests. Servers will receive only authenticated requests.

Default

jwtproxy

3.3.2.2.6.13. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_TOKEN_ISSUER

Jwtproxy issuer string, token lifetime, and optional auth page path to route unsigned requests to.

Default

wsmaster

3.3.2.2.6.14. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_TOKEN_TTL

JWTProxy issuer token lifetime.

Default

8800h

3.3.2.2.6.15. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_AUTH_LOADER_PATH

Optional authentication page path to route unsigned requests to.

Default

/_app/loader.html

3.3.2.2.6.16. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_IMAGE

JWTProxy image.

Default

quay.io/eclipse/che-jwtproxy:0.10.0

3.3.2.2.6.17. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_MEMORY__REQUEST

JWTProxy memory request.

Default

15mb

3.3.2.2.6.18. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_MEMORY__LIMIT

JWTProxy memory limit.

Default

128mb

3.3.2.2.6.19. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_CPU__REQUEST

JWTProxy CPU request.

Default

0.03

3.3.2.2.6.20. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_CPU__LIMIT

JWTProxy CPU limit.

Default

0.5

3.3.2.2.7. Configuration of the major WebSocket endpoint

3.3.2.2.7.1. CHE_CORE_JSONRPC_PROCESSOR__MAX__POOL__SIZE

Maximum size of the JSON RPC processing pool in case if pool size would be exceeded message execution will be rejected

Default

50

3.3.2.2.7.2. CHE_CORE_JSONRPC_PROCESSOR__CORE__POOL__SIZE

Initial JSON processing pool. Minimum number of threads that used to process major JSON RPC messages.

Default

5

3.3.2.2.7.3. CHE_CORE_JSONRPC_PROCESSOR__QUEUE__CAPACITY

Configuration of queue used to process JSON RPC messages.

Default

100000

3.3.2.2.7.4. CHE_METRICS_PORT

Port the HTTP server endpoint that would be exposed with Prometheus metrics.

Default

8087

3.3.2.2.8. CORS settings

3.3.2.2.8.1. CHE_CORS_ALLOWED__ORIGINS

Indicates which request origins are allowed. CORS filter on WS Master is turned off by default. Use environment variable "CHE_CORS_ENABLED=true" to turn it on.

Default

*

3.3.2.2.8.2. CHE_CORS_ALLOW__CREDENTIALS

Indicates if it allows processing of requests with credentials (in cookies, headers, TLS client certificates).

Default

false

3.3.2.2.9. Factory defaults

3.3.2.2.9.1. CHE_FACTORY_DEFAULT__PLUGINS

Editor and plugin which will be used for factories that are created from a remote Git repository which does not contain any OpenShift Dev Spaces-specific workspace descriptor Multiple plugins must be comma-separated, for example:
pluginFooPublisher/pluginFooName/pluginFooVersion,pluginBarPublisher/pluginBarName/pluginBarVersion

Default

redhat/vscode-commons/latest

3.3.2.2.9.2. CHE_FACTORY_DEFAULT__DEVFILE__FILENAMES

Devfile filenames to look on repository-based factories (for example GitHub). Factory will try to locate those files in the order they enumerated in the property.

Default

devfile.yaml,.devfile.yaml

3.3.2.2.10. Devfile defaults

3.3.2.2.10.1. CHE_FACTORY_DEFAULT__EDITOR

Editor that will be used for factories that are created from a remote Git repository which does not contain any OpenShift Dev Spaces-specific workspace descriptor.

Default

eclipse/che-theia/latest

3.3.2.2.10.2. CHE_FACTORY_SCM__FILE__FETCHER__LIMIT__BYTES

File size limit for the URL fetcher which fetch files from the SCM repository.

Default

102400

3.3.2.2.10.3. CHE_FACTORY_DEVFILE2__FILES__RESOLUTION__LIST

Additional files which may be present in repository to complement devfile v2, and should be referenced as links to SCM resolver service in factory to retrieve them.

Default

.che/che-editor.yaml,.che/che-theia-plugins.yaml,.vscode/extensions.json

3.3.2.2.10.4. CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR

Default Editor that should be provisioned into Devfile if there is no specified Editor Format is **editorPublisher/editorName/editorVersion** value. **NULL** or absence of value means that default editor should not be provisioned.

Default

eclipse/che-theia/latest

3.3.2.2.10.5. CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS

Default Plug-ins which should be provisioned for Default Editor. All the plugins from this list that are not explicitly mentioned in the user-defined devfile will be provisioned but only when the default editor is used or if the user-defined editor is the same as the default one (even if in different version). Format is comma-separated **pluginPublisher/pluginName/pluginVersion** values, and URLs. For example: **eclipse/che-theia-exec-plugin/0.0.1,eclipse/che-theia-terminal-plugin/0.0.1,https://cdn.pluginregistry.com/vi-mode/meta.yaml** If the plugin is a URL, the plugin's **meta.yaml** is retrieved from that URL.

Default

NULL

3.3.2.2.10.6. CHE_WORKSPACE_PROVISION_SECRET_LABELS

Defines comma-separated list of labels for selecting secrets from a user namespace, which will be mount into workspace containers as a files or environment variables. Only secrets that match ALL given labels will be selected.

Default

app.kubernetes.io/part-of=che.eclipse.org,app.kubernetes.io/component=workspace-secret

3.3.2.2.10.7. CHE_WORKSPACE_DEVFILE_ASYNC_STORAGE_PLUGIN

Plugin is added in case asynchronous storage feature will be enabled in workspace configuration and supported by environment

Default

eclipse/che-async-pv-plugin/latest

3.3.2.2.10.8. CHE_INFRA_KUBERNETES_ASYNC_STORAGE_IMAGE

Docker image for the OpenShift Dev Spaces asynchronous storage

Default

quay.io/eclipse/che-workspace-data-sync-storage:0.0.1

3.3.2.2.10.9. CHE_WORKSPACE_POD_NODE__SELECTOR

Optionally configures node selector for workspace Pod. Format is comma-separated key=value pairs, for example: **disktype=ssd,cpu=xlarge,foo=bar**

Default

NULL

3.3.2.2.10.10. CHE_WORKSPACE_POD_TOLERATIONS__JSON

Optionally configures tolerations for workspace Pod. Format is a string representing a JSON Array of taint tolerations, or **NULL** to disable it. The objects contained in the array have to follow the toleration v1 core specifications. Example:
[{"effect":"NoExecute","key":"aNodeTaint","operator":"Equal","value":"aValue"}]

Default

NULL

3.3.2.2.10.11. CHE_INFRA_KUBERNETES_ASYNC_STORAGE_SHUTDOWN__TIMEOUT__MIN

The timeout for the Asynchronous Storage Pod shutdown after stopping the last used workspace. Value less or equal to 0 interpreted as disabling shutdown ability.

Default

120

3.3.2.2.10.12. CHE_INFRA_KUBERNETES_ASYNC_STORAGE_SHUTDOWN__CHECK__PERIOD__MIN

Defines the period with which the Asynchronous Storage Pod stopping ability will be performed (once in 30 minutes by default)

Default

30

3.3.2.2.10.13. CHE_INTEGRATION_BITBUCKET_SERVER__ENDPOINTS

Bitbucket endpoints used for factory integrations. Comma separated list of Bitbucket server URLs or NULL if no integration expected.

Default

NULL

3.3.2.2.10.14. CHE_INTEGRATION_GITLAB_SERVER__ENDPOINTS

GitLab endpoints used for factory integrations. Comma separated list of GitLab server URLs or NULL if no integration expected.

Default

NULL

3.3.2.2.10.15. CHE_INTEGRATION_GITLAB_OAUTH__ENDPOINT

Address of the GitLab server with configured OAuth 2 integration

Default

NULL

3.3.2.2.10.16. CHE_OAUTH2_GITLAB_CLIENTID__FILEPATH

Configuration of GitLab OAuth2 client. Used to obtain Personal access tokens. Location of the file with GitLab client id.

Default

NULL

3.3.2.2.10.17. CHE_OAUTH2_GITLAB_CLIENTSECRET__FILEPATH

Location of the file with GitLab client secret.

Default

NULL#

3.3.2.2.11. Che system

3.3.2.2.11.1. CHE_SYSTEM_SUPER__PRIVILEGED__MODE

System Super Privileged Mode. Grants users with the manageSystem permission additional permissions for getByKey, getByNameSpace, stopWorkspaces, and getResourcesInformation. These are not given to admins by default and these permissions allow admins gain visibility to any workspace along with naming themselves with administrator privileges to those workspaces.

Default

false

3.3.2.2.11.2. CHE_SYSTEM_ADMIN__NAME

Grant system permission for **che.admin.name** user. If the user already exists it'll happen on component startup, if not - during the first login when user is persisted in the database.

Default

admin

3.3.2.2.12. Workspace limits

3.3.2.2.12.1. CHE_LIMITS_WORKSPACE_ENV_RAM

Workspaces are the fundamental runtime for users when doing development. You can set parameters that limit how workspaces are created and the resources that are consumed. The maximum amount of RAM that a user can allocate to a workspace when they create a new workspace. The RAM slider is adjusted to this maximum value.

Default

16gb

3.3.2.2.12.2. CHE_LIMITS_WORKSPACE_IDLE_TIMEOUT

The length of time in milliseconds that a user is idle with their workspace when the system will suspend the workspace and then stopping it. Idleness is the length of time that the user has not interacted with the workspace, meaning that one of the agents has not received interaction. Leaving a browser window open counts toward idleness.

Default

1800000

3.3.2.2.12.3. CHE_LIMITS_WORKSPACE_RUN_TIMEOUT

The length of time in milliseconds that a workspace will run, regardless of activity, before the system will suspend it. Set this property if you want to automatically stop workspaces after a period of time. The default is zero, meaning that there is no run timeout.

Default

0

3.3.2.2.13. Users workspace limits

3.3.2.2.13.1. CHE_LIMITS_USER_WORKSPACES_RAM

The total amount of RAM that a single user is allowed to allocate to running workspaces. A user can allocate this RAM to a single workspace or spread it across multiple workspaces.

Default

-1

3.3.2.2.13.2. CHE_LIMITS_USER_WORKSPACES_COUNT

The maximum number of workspaces that a user is allowed to create. The user will be presented with an error message if they try to create additional workspaces. This applies to the total number of both running and stopped workspaces.

Default

-1

3.3.2.2.13.3. CHE_LIMITS_USER_WORKSPACES_RUN_COUNT

The maximum number of running workspaces that a single user is allowed to have. If the user has reached this threshold and they try to start an additional workspace, they will be prompted with an error message. The user will need to stop a running workspace to activate another.

Default

1

3.3.2.2.14. Organizations workspace limits

3.3.2.2.14.1. CHE_LIMITS_ORGANIZATION_WORKSPACES_RAM

The total amount of RAM that a single organization (team) is allowed to allocate to running workspaces. An organization owner can allocate this RAM however they see fit across the team's workspaces.

Default

-1

3.3.2.2.14.2. CHE_LIMITS_ORGANIZATION_WORKSPACES_COUNT

The maximum number of workspaces that a organization is allowed to own. The organization will be presented an error message if they try to create additional workspaces. This applies to the total number of both running and stopped workspaces.

Default

-1

3.3.2.2.14.3. CHE_LIMITS_ORGANIZATION_WORKSPACES_RUN_COUNT

The maximum number of running workspaces that a single organization is allowed. If the organization has reached this threshold and they try to start an additional workspace, they will be prompted with an error message. The organization will need to stop a running workspace to activate another.

Default

-1

3.3.2.2.15. Multi-user-specific OpenShift infrastructure configuration

3.3.2.2.15.1. CHE_INFRA_OPENSHIFT_OAUTH__IDENTITY__PROVIDER

Alias of the OpenShift identity provider registered in Keycloak, that should be used to create workspace OpenShift resources in OpenShift namespaces owned by the current OpenShift Dev Spaces user. Should be set to NULL if **che.infra.openshift.project** is set to a non-empty value. See: OpenShift identity provider

Default

NULL

3.3.2.2.16. OIDC configuration

3.3.2.2.16.1. **CHE_OIDC_AUTH__SERVER__URL**

Url to OIDC identity provider server Can be set to NULL only if **che.oidc.oidcProvider** is used

Default

http://${CHE_HOST}:5050/auth

3.3.2.2.16.2. **CHE_OIDC_AUTH__INTERNAL__SERVER__URL**

Internal network service Url to OIDC identity provider server

Default

NULL

3.3.2.2.16.3. **CHE_OIDC_ALLOWED__CLOCK__SKEW__SEC**

The number of seconds to tolerate for clock skew when verifying **exp** or **nbf** claims.

Default

3

3.3.2.2.16.4. **CHE_OIDC_USERNAME__CLAIM**

Username claim to be used as user display name when parsing JWT token if not defined the fallback value is 'preferred_username' in Keycloak installations and **name** in Dex installations.

Default

NULL

3.3.2.2.16.5. **CHE_OIDC_OIDC__PROVIDER**

Base URL of an alternate OIDC provider that provides a discovery endpoint as detailed in the following specification Obtaining OpenID Provider Configuration Information Deprecated, use **che.oidc.auth_server_url** and **che.oidc.auth_internal_server_url** instead.

Default

NULL

3.3.2.2.17. Keycloak configuration

3.3.2.2.17.1. **CHE_KEYCLOAK_REALM**

Keycloak realm is used to authenticate users Can be set to NULL only if **che.keycloak.oidcProvider** is used

Default

che

3.3.2.2.17.2. CHE_KEYCLOAK_CLIENT__ID

Keycloak client identifier in **che.keycloak.realm** to authenticate users in the dashboard, the IDE, and the CLI.

Default

che-public

3.3.2.2.17.3. CHE_KEYCLOAK_OSO_ENDPOINT

URL to access OSO OAuth tokens

Default

NULL

3.3.2.2.17.4. CHE_KEYCLOAK_GITHUB_ENDPOINT

URL to access Github OAuth tokens

Default

NULL

3.3.2.2.17.5. CHE_KEYCLOAK_USE__NONCE

Use the OIDC optional **nonce** feature to increase security.

Default

true

3.3.2.2.17.6. CHE_KEYCLOAK_JS__ADAPTER__URL

URL to the Keycloak Javascript adapter to use. if set to NULL, then the default used value is **${che.keycloak.auth_server_url}/js/keycloak.js**, or **<che-server>/api/keycloak/OIDCKeycloak.js** if an alternate **oidc_provider** is used

Default

NULL

3.3.2.2.17.7. CHE_KEYCLOAK_USE__FIXED__REDIRECT__URLS

Set to true when using an alternate OIDC provider that only supports fixed redirect Urls This property is ignored when **che.keycloak.oidc_provider** is NULL

Default

false

3.3.2.2.17.8. CHE_OAUTH_SERVICE__MODE

Configuration of OAuth Authentication Service that can be used in "embedded" or "delegated" mode. If set to "embedded", then the service work as a wrapper to OpenShift Dev Spaces's OAuthAuthenticator (as in Single User mode). If set to "delegated", then the service will use Keycloak IdentityProvider mechanism. Runtime Exception **wii** be thrown, in case if this property is not set properly.

Default

delegated

3.3.2.2.17.9. CHE_KEYCLOAK_CASCADE__USER__REMOVAL__ENABLED

Configuration for enabling removing user from Keycloak server on removing user from OpenShift Dev Spaces database. By default it's disabled. Can be enabled in some special cases when deleting a user in OpenShift Dev Spaces database should execute removing related-user from Keycloak. For correct work need to set administrator username ${che.keycloak.admin_username} and password ${che.keycloak.admin_password}.

Default

false

3.3.2.2.17.10. CHE_KEYCLOAK_ADMIN__USERNAME

Keycloak administrator username. Will be used for deleting user from Keycloak on removing user from OpenShift Dev Spaces database. Make sense only in case ${che.keycloak.cascade_user_removal_enabled} set to 'true'

Default

NULL

3.3.2.2.17.11. CHE_KEYCLOAK_ADMIN__PASSWORD

Keycloak administrator password. Will be used for deleting user from Keycloak on removing user from OpenShift Dev Spaces database. Make sense only in case ${che.keycloak.cascade_user_removal_enabled} set to 'true'

Default

NULL

3.3.2.2.17.12. CHE_KEYCLOAK_USERNAME_REPLACEMENT__PATTERNS

User name adjustment configuration. OpenShift Dev Spaces needs to use the usernames as part of Kubernetes object names and labels and therefore has stricter requirements on their format than the identity providers usually allow (it needs them to be DNS-compliant). The adjustment is represented by comma-separated key-value pairs. These are sequentially used as arguments to the String.replaceAll function on the original username. The keys are regular expressions, values are replacement strings that replace the characters in the username that match the regular expression. The modified username will only be stored in the OpenShift Dev Spaces database and will not be advertised back to the identity provider. It is recommended to use DNS-compliant characters as replacement strings (values in the key-value pairs). Example: **=-,@=-at-** changes \ to **-** and @ to **-at-** so the username **org\user@com** becomes **org-user-at-com.**

Default

NULL

3.4. CONFIGURING WORKSPACES GLOBALLY

This section describes how an administrator can configure workspaces globally.

- Section 3.4.1, "Configuring the number of workspaces that a user can create"

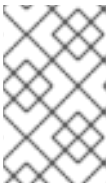
- Section 3.4.2, “Deploying OpenShift Dev Spaces with support for Git repositories with self-signed certificates”
- Section 3.4.3, “Configuring workspaces nodeSelector”

3.4.1. Configuring the number of workspaces that a user can create

This procedure describes how to configure the number of workspaces that a user can create. By creating multiple workspaces, users can have access to workspaces with different configurations simultaneously.

Prerequisites

- You have installed an instance of **OpenShift Dev Spaces** by using the Operator.
- You have determined the value of the ***<number-of-workspaces>*** placeholder.



NOTE

If the value is **-1**, users can create an unlimited number of workspaces. If the value is a positive integer, users can create as many workspaces as the value of the integer. The default value is **-1**.

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”.

```
spec:
  server:
    customCheProperties:
      CHE_LIMITS_USER_WORKSPACES_COUNT: "<number-of-workspaces>"
```

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.4.2. Deploying OpenShift Dev Spaces with support for Git repositories with self-signed certificates

You can configure OpenShift Dev Spaces to support operations on Git providers that use self-signed certificates.

Prerequisites

- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.
- Git version 2 or later

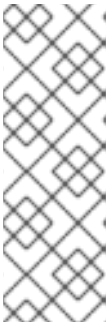
Procedure

1. Create a new **ConfigMap** with details about the Git server:

```
$ oc create configmap che-git-self-signed-cert \
  --from-file=ca.crt=<path_to_certificate> \ (1)
  --from-literal=githost=<host:port> -n openshift-devspaces (2)
```

(1) Path to self-signed certificate

(2) The host and port of the HTTPS connection on the Git server (optional).



NOTE

- When **githost** is not specified, the given certificate is used for all HTTPS repositories.
- Certificate files are typically stored as Base64 ASCII files, such as. **.pem**, **.crt**, **.ca-bundle**. Also, they can be encoded as binary data, for example, **.cer**. All **Secrets** that hold certificate files should use the Base64 ASCII certificate rather than the binary data certificate.

2. Add the required labels to the ConfigMap:

```
$ oc label configmap che-git-self-signed-cert \
  app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

3. Configure OpenShift Dev Spaces operand to use self-signed certificates for Git repositories. See Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource” .

```
spec:
  server:
    gitSelfSignedCert: true
```

Verification steps

- Create and start a new workspace. Every container used by the workspace mounts a special volume that contains a file with the self-signed certificate. The repository’s **.git/config** file contains information about the Git server host (its URL) and the path to the certificate in the **http** section (see Git documentation about git-config).

Example 3.11. A .git/config file example

```
[http "https://10.33.177.118:3000"]
sslCAInfo = /etc/che/git/cert/ca.crt
```

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

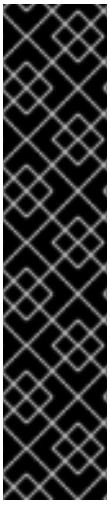
3.4.3. Configuring workspaces nodeSelector

This section describes how to configure **nodeSelector** for Pods of OpenShift Dev Spaces workspaces.

Procedure

OpenShift Dev Spaces uses the **CHE_WORKSPACE_POD_NODE__SELECTOR** environment variable to configure **nodeSelector**. This variable may contain a set of comma-separated **key=value** pairs to form the nodeSelector rule, or **NULL** to disable it.

```
CHE_WORKSPACE_POD_NODE__SELECTOR=disktype=ssd,cpu=xlarge,[key=value]
```



IMPORTANT

nodeSelector must be configured during OpenShift Dev Spaces installation. This prevents existing workspaces from failing to run due to volumes affinity conflict caused by existing workspace PVC and Pod being scheduled in different zones.

To avoid Pods and PVCs to be scheduled in different zones on large, multizone clusters, create an additional **StorageClass** object (pay attention to the **allowedTopologies** field), which will coordinate the PVC creation process.

Pass the name of this newly created **StorageClass** to OpenShift Dev Spaces through the **CHE_INFRA_KUBERNETES_PVC_STORAGE__CLASS__NAME** environment variable. A default empty value of this variable instructs OpenShift Dev Spaces to use the cluster's default **StorageClass**.

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.5. CACHING IMAGES FOR FASTER WORKSPACE START

To improve the start time performance of OpenShift Dev Spaces workspaces, use the Image Puller, a OpenShift Dev Spaces-agnostic component that can be used to pre-pull images for OpenShift clusters. The Image Puller is an additional OpenShift deployment which creates a *DaemonSet* that can be configured to pre-pull relevant OpenShift Dev Spaces workspace images on each node. These images would already be available when a OpenShift Dev Spaces workspace starts, therefore improving the workspace start time.

The Image Puller provides the following parameters for configuration.

Table 3.8. Image Puller parameters

Parameter	Usage	Default
CACHING_INTERVAL_HOURS	DaemonSets health checks interval in hours	**"1"**
CACHING_MEMORY_REQUEST	The memory request for each cached image while the puller is running. See Section 3.5.2, "Defining the memory parameters for the Image Puller".	**10Mi**

Parameter	Usage	Default
CACHING_MEMORY_LIMIT	The memory limit for each cached image while the puller is running. See Section 3.5.2, "Defining the memory parameters for the Image Puller".	**20Mi**
CACHING_CPU_REQUEST	The processor request for each cached image while the puller is running	**.05** or 50 millicores
CACHING_CPU_LIMIT	The processor limit for each cached image while the puller is running	**.2** or 200 millicores
DAEMONSET_NAME	Name of DaemonSet to create	**kubernetes-image-puller**
DEPLOYMENT_NAME	Name of the Deployment to create	**kubernetes-image-puller**
NAMESPACE	OpenShift project containing DaemonSet to create	**k8s-image-puller**
IMAGES	Semicolon-separated list of images to pull, in the format ***<name1>=<image1>;<name2>=<image2>***. See Section 3.5.1, "Defining the list of images to pull".	
NODE_SELECTOR	Node selector to apply to the pods created by the DaemonSet	**'{}'**
AFFINITY	Affinity applied to pods created by the DaemonSet	**'{}'**
IMAGE_PULL_SECRETS	List of image pull secrets, in the format **pullsecret1;...** to add to pods created by the DaemonSet. Those secrets need to be in the image puller's namespace and a cluster administrator must create them.	**""**

Additional resources

- Section 3.5.1, "Defining the list of images to pull"
- Section 3.5.2, "Defining the memory parameters for the Image Puller".

- Section 3.5.3, “Installing Image Puller on OpenShift by using the web console”
- Section 3.5.4, “Installing Image Puller on OpenShift by using the CLI”
- Kubernetes Image Puller source code repository

3.5.1. Defining the list of images to pull

The Image Puller can pre-pull most images, including scratch images such as **che-machine-exec**. However, images that mount volumes in the Dockerfile, such as **traefik**, are not supported for pre-pulling on OpenShift 3.11.

Procedure

1. Gather a list of relevant container images for prepulling by navigating to the **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/plugin-registry/v3/external_images.txt** URL.
2. Determine images from the list for pre-pulling. For faster workspace startup times, consider pre-pulling workspace related images such as **che-theia**, **che-machine-exec**, **che-theia-endpoint-runtime-binary**, and plug-in sidecar images.

Additional resources

- Section 3.5.3, “Installing Image Puller on OpenShift by using the web console”
- Section 3.5.4, “Installing Image Puller on OpenShift by using the CLI”

3.5.2. Defining the memory parameters for the Image Puller

Define the memory requests and limits parameters to ensure pulled containers and the platform have enough memory to run.

Prerequisites

- Section 3.5.1, “Defining the list of images to pull”

Procedure

1. To define the minimal value for **CACHING_MEMORY_REQUEST** or **CACHING_MEMORY_LIMIT**, consider the necessary amount of memory required to run each of the container images to pull.
2. To define the maximal value for **CACHING_MEMORY_REQUEST** or **CACHING_MEMORY_LIMIT**, consider the total memory allocated to the DaemonSet Pods in the cluster:

   ```
   (memory limit) * (number of images) * (number of nodes in the cluster)
   ```

   Pulling 5 images on 20 nodes, with a container memory limit of **20Mi** requires **2000Mi** of memory.

Additional resources

- Section 3.5.3, “Installing Image Puller on OpenShift by using the web console”

- Section 3.5.4, “Installing Image Puller on OpenShift by using the CLI”

3.5.3. Installing Image Puller on OpenShift by using the web console

You can install the community supported Kubernetes Image Puller Operator on OpenShift using the OpenShift web console.

Prerequisites

- Section 3.5.1, “Defining the list of images to pull”
- Section 3.5.2, “Defining the memory parameters for the Image Puller”.
- An OpenShift web console session by a cluster administrator. See Accessing the web console.

Procedure

1. Install the community supported Kubernetes Image Puller Operator. See Installing from OperatorHub using the web console.
2. Create a kubernetes-image-puller **KubernetesImagePuller** operand from the community supported Kubernetes Image Puller Operator. See Creating applications from installed Operators.

3.5.4. Installing Image Puller on OpenShift by using the CLI

You can install the Kubernetes Image Puller on OpenShift by using OpenShift **oc** management tool.

Prerequisites

- Section 3.5.1, “Defining the list of images to pull” .
- Section 3.5.2, “Defining the memory parameters for the Image Puller”.
- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.

Procedure

1. Clone the Image Puller repository and get in the directory containing the OpenShift templates:

```
$ git clone https://github.com/che-incubator/kubernetes-image-puller
$ cd kubernetes-image-puller/deploy/openshift
```

2. Configure the **app.yaml**, **configmap.yaml** and **serviceaccount.yaml** OpenShift templates using following parameters:

Table 3.9. Image Puller OpenShift templates parameters in app.yaml

Value	Usage	Default

Value	Usage	Default
DEPLOYMENT_NAME	The value of **DEPLOYMENT_NAME** in the ConfigMap	**kubernetes-image-puller**
IMAGE	Image used for the **kubernetes-image-puller** deployment	**registry.redhat.io/devspaces/imagepuller-rhel8:3.0**
IMAGE_TAG	The image tag to pull	**latest**
SERVICEACCOUNT_NAME	The name of the ServiceAccount created and used by the deployment	**kubernetes-image-puller**

Table 3.10. Image Puller OpenShift templates parameters in**configmap.yaml**

Value	Usage	Default
CACHING_CPU_LIMIT	The value of **CACHING_CPU_LIMIT** in the ConfigMap	**.2**
CACHING_CPU_REQUEST	The value of **CACHING_CPU_REQUEST** in the ConfigMap	**.05**
CACHING_INTERVAL_HOURS	The value of **CACHING_INTERVAL_HOURS** in the ConfigMap	**"1"**
CACHING_MEMORY_LIMIT	The value of **CACHING_MEMORY_LIMIT** in the ConfigMap	**"20Mi"**
CACHING_MEMORY_REQUEST	The value of **CACHING_MEMORY_REQUEST** in the ConfigMap	**"10Mi"**
DAEMONSET_NAME	The value of **DAEMONSET_NAME** in the ConfigMap	**kubernetes-image-puller**
DEPLOYMENT_NAME	The value of **DEPLOYMENT_NAME** in the ConfigMap	**kubernetes-image-puller**

Value	Usage	Default
IMAGES	The value of **IMAGES** in the ConfigMap	**"undefined"**
NAMESPACE	The value of **NAMESPACE** in the ConfigMap	**k8s-image-puller**
NODE_SELECTOR	The value of **NODE_SELECTOR** in the ConfigMap	**"{}"**

Table 3.11. Image Puller OpenShift templates parameters in serviceaccount.yaml

Value	Usage	Default
SERVICEACCOUNT_NAME	The name of the ServiceAccount created and used by the deployment	**kubernetes-image-puller**

3. Create an OpenShift project to host the Image Puller:

```
$ oc new-project <k8s-image-puller>
```

4. Process and apply the templates to install the puller:

```
$ oc process -f serviceaccount.yaml | oc apply -f -
$ oc process -f configmap.yaml | oc apply -f -
$ oc process -f app.yaml | oc apply -f -
```

Verification steps

1. Verify the existence of a *<kubernetes-image-puller>* deployment and a *<kubernetes-image-puller>* DaemonSet. The DaemonSet needs to have a Pod for each node in the cluster:

```
$ oc get deployment,daemonset,pod --namespace <k8s-image-puller>
```

2. Verify the values of the *<kubernetes-image-puller>* **ConfigMap**.

```
$ oc get configmap <kubernetes-image-puller> --output yaml
```

3.6. CONFIGURING OBSERVABILITY

To configure OpenShift Dev Spaces observability features, see:

- Section 3.6.1, “Che-Theia workspaces”
- Section 3.6.2, “Configuring server logging”

- Section 3.6.3, “Collecting logs using dsc”
- Section 3.6.4.3, “Monitoring OpenShift Dev Spaces Server”
- Section 3.6.4.2, “Monitoring the DevWorkspace Operator”

3.6.1. Che-Theia workspaces

3.6.1.1. Telemetry overview

Telemetry is the explicit and ethical collection of operation data. By default, telemetry is not available in Red Hat OpenShift Dev Spaces, but in the Che-Theia editor there is an abstract API that allows enabling telemetry using the plug-in mechanism and in the **chectl** command line tool usage data can be collected using segment. This approach is used in the "Eclipse Che hosted by Red Hat " service where telemetry is enabled for every Che-Theia workspace.

This documentation includes a guide describing how to make your own telemetry client for Red Hat OpenShift Dev Spaces, followed by an overview of the Red Hat OpenShift Dev Spaces Woopra Telemetry Plugin.

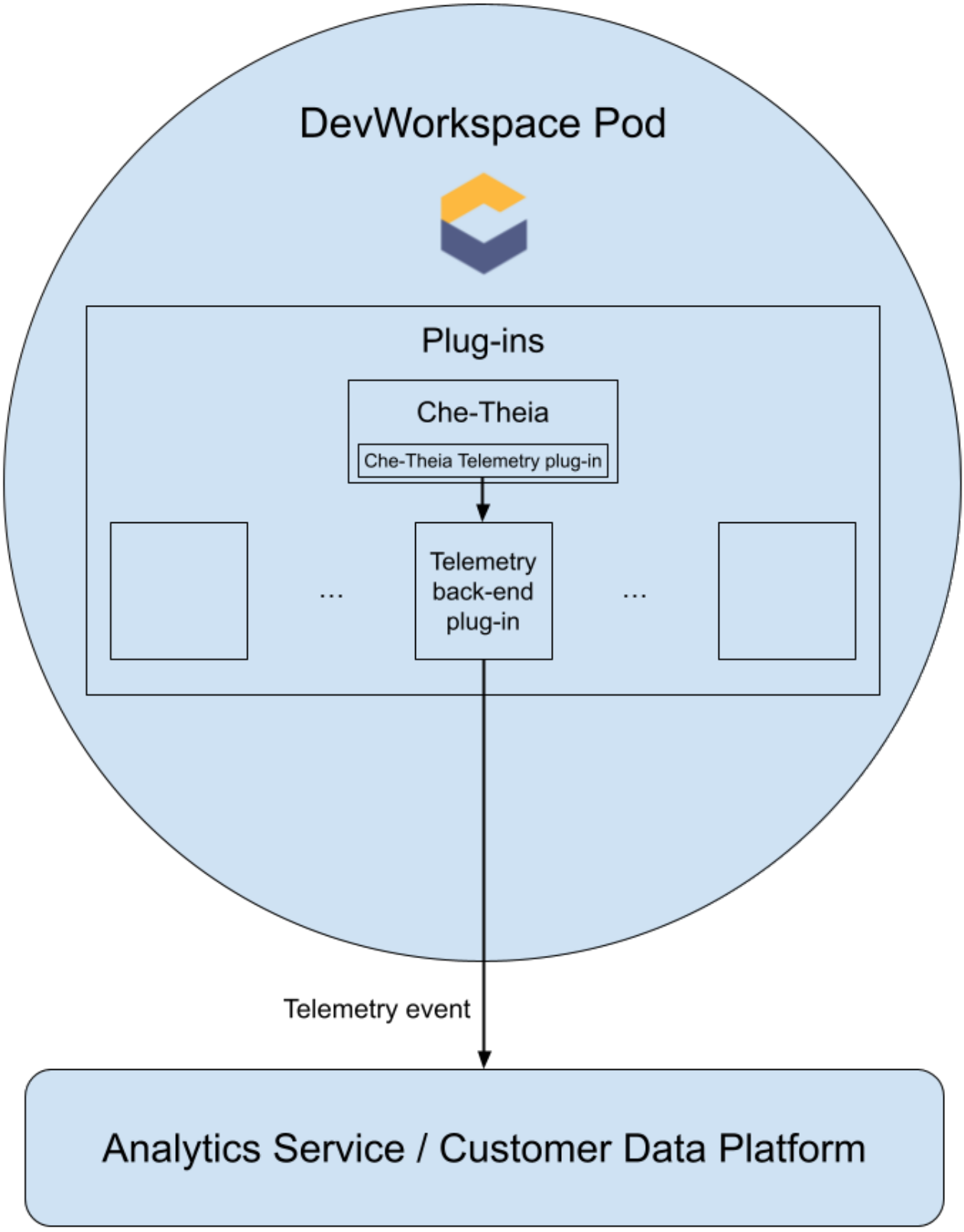
3.6.1.2. Use cases

Red Hat OpenShift Dev Spaces telemetry API allows tracking:

- Duration of a workspace utilization
- User-driven actions such as file editing, committing, and pushing to remote repositories.
- Programming languages and devfiles used in workspaces.

3.6.1.3. How it works

When a DevWorkspace starts, the **che-theia** container starts the telemetry plug-in which is responsible for sending telemetry events to a backend. If the **$DEVWORKSPACE_TELEMETRY_BACKEND_PORT** environment variable is set in the DevWorkspace Pod, the telemetry plug-in sends events to a backend listening at that port. The backend turns received events into a backend-specific representation of the events and sends them to the configured analytics backend (for example, Segment or Woopra).



3.6.1.4. Events sent to the backend by the Che-Theia telemetry plug-in

Event	Description
WORKSPACE_OPENED	Sent when Che-Theia starts running

Event	Description
COMMIT_LOCALLY	Sent when a commit was made locally with the **git.commit** Theia command
PUSH_TO_REMOTE	Sent when a Git push was made with the **git.push** Theia command
EDITOR_USED	Sent when a file was changed within the editor

Other events such as **WORKSPACE_INACTIVE** and **WORKSPACE_STOPPED** can be detected within the back-end plug-in.

3.6.1.5. The Woopra telemetry plug-in

The Woopra Telemetry Plugin is a plug-in built to send telemetry from a Red Hat OpenShift Dev Spaces installation to Segment and Woopra. This plug-in is used by Eclipse Che hosted by Red Hat , but any Red Hat OpenShift Dev Spaces deployment can take advantage of this plug-in. There are no dependencies other than a valid Woopra domain and Segment Write key. The devfile v2 for the plug-in, plugin.yaml, has four environment variables that can be passed to the plug-in:

- **WOOPRA_DOMAIN** - The Woopra domain to send events to.
- **SEGMENT_WRITE_KEY** - The write key to send events to Segment and Woopra.
- **WOOPRA_DOMAIN_ENDPOINT** - If you prefer not to pass in the Woopra domain directly, the plug-in will get it from a supplied HTTP endpoint that returns the Woopra Domain.
- **SEGMENT_WRITE_KEY_ENDPOINT** - If you prefer not to pass in the Segment write key directly, the plug-in will get it from a supplied HTTP endpoint that returns the Segment write key.

To enable the Woopra plug-in on the Red Hat OpenShift Dev Spaces installation:

Procedure

- Deploy the **plugin.yaml** devfile v2 file to an HTTP server with the environment variables set correctly.

1. Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    workspacesDefaultPlugins:
    - editor: eclipse/che-theia/next    (1)
      plugins:                          (2)
      - 'https://your-web-server/plugin.yaml'
```

(1) The **editorId** to set the telemetry plug-in for.

(2) The URL to the telemetry plug-in's devfile v2 definition.

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.6.1.6. Creating a telemetry plug-in

This section shows how to create an **AnalyticsManager** class that extends **AbstractAnalyticsManager** and implements the following methods:

- **isEnabled()** - determines whether the telemetry backend is functioning correctly. This can mean always returning **true**, or have more complex checks, for example, returning **false** when a connection property is missing.
- **destroy()** - cleanup method that is run before shutting down the telemetry backend. This method sends the **WORKSPACE_STOPPED** event.
- **onActivity()** - notifies that some activity is still happening for a given user. This is mainly used to send **WORKSPACE_INACTIVE** events.
- **onEvent()** - submits telemetry events to the telemetry server, such as **WORKSPACE_USED** or **WORKSPACE_STARTED**.
- **increaseDuration()** - increases the duration of a current event rather than sending many events in a small frame of time.

The following sections cover:

- Creating a telemetry server to echo events to standard output.
- Extending the OpenShift Dev Spaces telemetry client and implementing a user's custom backend.
- Creating a **plugin.yaml** file representing a DevWorkspace plug-in for the custom backend.
- Specifying of a location of a custom plug-in to OpenShift Dev Spaces by setting the **workspacesDefaultPlugins** attribute from the **CheCluster** custom resource.

3.6.1.6.1. Getting started

This document describes the steps required to extend the OpenShift Dev Spaces telemetry system to communicate with to a custom backend:

1. Creating a server process that receives events
2. Extending OpenShift Dev Spaces libraries to create a backend that sends events to the server
3. Packaging the telemetry backend in a container and deploying it to an image registry
4. Adding a plug-in for your backend and instructing OpenShift Dev Spaces to load the plug-in in your DevWorkspaces

A finished example of the telemetry backend is available here.

Creating a server that receives events
For demonstration purposes, this example shows how to create a server that receives events from our telemetry plug-in and writes them to standard output.

For production use cases, consider integrating with a third-party telemetry system (for example, Segment, Woopra) rather than creating your own telemetry server. In this case, use your provider's APIs to send events from your custom backend to their system.

The following Go code starts a server on port **8080** and writes events to standard output:

Example 3.12. main.go

```
package main

import (
 "io/ioutil"
 "net/http"

 "go.uber.org/zap"
)

var logger *zap.SugaredLogger

func event(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /event")
 case "POST":
  logger.Info("POST /event")
 }
 body, err := req.GetBody()
 if err != nil {
  logger.With("err", err).Info("error getting body")
  return
 }
 responseBody, err := ioutil.ReadAll(body)
 if err != nil {
  logger.With("error", err).Info("error reading response body")
  return
 }
 logger.With("body", string(responseBody)).Info("got event")
}

func activity(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /activity, doing nothing")
 case "POST":
  logger.Info("POST /activity")
  body, err := req.GetBody()
  if err != nil {
   logger.With("error", err).Info("error getting body")
   return
  }
  responseBody, err := ioutil.ReadAll(body)
  if err != nil {
   logger.With("error", err).Info("error reading response body")
   return
  }
  logger.With("body", string(responseBody)).Info("got activity")
```

```
 }
}

func main() {

 log, _ := zap.NewProduction()
 logger = log.Sugar()

 http.HandleFunc("/event", event)
 http.HandleFunc("/activity", activity)
 logger.Info("Added Handlers")

 logger.Info("Starting to serve")
 http.ListenAndServe(":8080", nil)
}
```

Create a container image based on this code and expose it as a deployment in OpenShift in the openshift-devspaces project. The code for the example telemetry server is available at telemetry-server-example. To deploy the telemetry server, clone the repository and build the container:

```
$ git clone https://github.com/che-incubator/telemetry-server-example
$ cd telemetry-server-example
$ podman build -t registry/organization/telemetry-server-example:latest .
$ podman push registry/organization/telemetry-server-example:latest
```

Both **manifest_with_ingress.yaml** and **manifest_with_route** contain definitions for a Deployment and Service. The former also defines a Kubernetes Ingress, while the latter defines an OpenShift Route.

In the manifest file, replace the **image** and **host** fields to match the image you pushed, and the public hostname of your OpenShift cluster. Then run:

```
$ kubectl apply -f manifest_with_[ingress|route].yaml -n {prod-namespace}
```

3.6.1.6.2. Creating the back-end project

NOTE

For fast feedback when developing, it is recommended to do development inside a DevWorkspace. This way, you can run the application in a cluster and receive events from the front-end telemetry plug-in.

1. Maven Quarkus project scaffolding:

```
mvn io.quarkus:quarkus-maven-plugin:2.7.1.Final:create \
    -DprojectGroupId=mygroup -DprojectArtifactId=devworkspace-telemetry-example-plugin \
-DprojectVersion=1.0.0-SNAPSHOT
```

2. Remove the files under **src/main/java/mygroup** and **src/test/java/mygroup**.

3. Consult the GitHub packages for the latest version and Maven coordinates of **backend-base**.

4. Add the following dependencies to your **pom.xml**:

Example 3.13. **pom.xml**

```
<!-- Required -->
<dependency>
    <groupId>org.eclipse.che.incubator.workspace-telemetry</groupId>
    <artifactId>backend-base</artifactId>
    <version>LATEST VERSION FROM PREVIOUS STEP</version>
</dependency>


<!-- Used to make http requests to the telemetry server -->
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client-jackson</artifactId>
</dependency>
```

5. Create a personal access token with **read:packages** permissions to download the **org.eclipse.che.incubator.workspace-telemetry:backend-base** dependency from GitHub packages.

6. Add your GitHub username, personal access token and **che-incubator** repository details in your **~/.m2/settings.xml** file:

Example 3.14. **settings.xml**

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>che-incubator</id>
      <username>YOUR GITHUB USERNAME</username>
      <password>YOUR GITHUB TOKEN</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>github</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>false</enabled></snapshots>
          </repository>
          <repository>
```

```
            <id>che-incubator</id>
            <url>https://maven.pkg.github.com/che-incubator/che-workspace-telemetry-
client</url>
          </repository>
        </repositories>
      </profile>
    </profiles>
  </settings>
```

3.6.1.6.3. Creating a concrete implementation of AnalyticsManager and adding specialized logic

Create two files in your project under **src/main/java/mygroup**:

- **MainConfiguration.java** - contains configuration provided to **AnalyticsManager**.
- **AnalyticsManager.java** - contains logic specific to the telemetry system.

Example 3.15. MainConfiguration.java

```
package org.my.group;

import java.util.Optional;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;

import org.eclipse.che.incubator.workspace.telemetry.base.BaseConfiguration;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Dependent
@Alternative
public class MainConfiguration extends BaseConfiguration {
    @ConfigProperty(name = "welcome.message")  1
    Optional<String> welcomeMessage;  2
}
```

1 A MicroProfile configuration annotation is used to inject the **welcome.message** configuration.

For more details on how to set configuration properties specific to your backend, see the Quarkus Configuration Reference Guide.

Example 3.16. AnalyticsManager.java

```
package org.my.group;

import java.util.HashMap;
import java.util.Map;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;
import javax.inject.Inject;
```

```
import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.AnalyticsEvent;
import org.eclipse.che.incubator.workspace.telemetry.finder.DevWorkspaceFinder;
import org.eclipse.che.incubator.workspace.telemetry.finder.UsernameFinder;
import org.eclipse.microprofile.rest.client.inject.RestClient;
import org.slf4j.Logger;

import static org.slf4j.LoggerFactory.getLogger;

@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {

    private static final Logger LOG = getLogger(AbstractAnalyticsManager.class);

    public AnalyticsManager(MainConfiguration mainConfiguration, DevWorkspaceFinder
devworkspaceFinder, UsernameFinder usernameFinder) {
        super(mainConfiguration, devworkspaceFinder, usernameFinder);

        mainConfiguration.welcomeMessage.ifPresentOrElse(      (1)
            (str) -> LOG.info("The welcome message is: {}", str),
            () -> LOG.info("No welcome message provided")
        );
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public void destroy() {}

    @Override
    public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent, String
resolution, Map<String, Object> properties) {
        LOG.info("The received event is: {}", event);          (2)
    }

    @Override
    public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) { }

    @Override
    public void onActivity() {}
}
```

1. Log the welcome message if it was provided.
2. Log the event received from the front-end plug-in.

Since **org.my.group.AnalyticsManager** and **org.my.group.MainConfiguration** are alternative beans, specify them using the **quarkus.arc.selected-alternatives** property in **src/main/resources/application.properties**.

Example 3.17. application.properties

```
quarkus.arc.selected-alternatives=MainConfiguration,AnalyticsManager
```

3.6.1.6.4. Running the application within a DevWorkspace

1. Set the **DEVWORKSPACE_TELEMETRY_BACKEND_PORT** environment variable in the DevWorkspace. Here, the value is set to **4167**.

```
spec:
  template:
    attributes:
      workspaceEnv:
        - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
          value: '4167'
```

2. Restart the DevWorkspace from the Red Hat OpenShift Dev Spaces dashboard.

3. Run the following command within a DevWorkspace's terminal window to start the application. Use the **--settings** flag to specify path to the location of the **settings.xml** file that contains the GitHub access token.

```
$ mvn --settings=settings.xml quarkus:dev -
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}
```

The application now receives telemetry events through port **4167** from the front-end plug-in.

Verification steps

1. Verify that the following output is logged:

```
INFO  [org.ecl.che.inc.AnalyticsManager] (Quarkus Main Thread) No welcome message
provided
INFO  [io.quarkus] (Quarkus Main Thread) devworkspace-telemetry-example-plugin 1.0.0-
SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 0.323s. Listening on:
http://localhost:4167
INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, kubernetes-client, rest-
client, rest-client-jackson, resteasy, resteasy-jsonb, smallrye-context-propagation, smallrye-
openapi, swagger-ui, vertx]
```

2. To verify that the **onEvent()** method of **AnalyticsManager** receives events from the front-end plug-in, press the **l** key to disable Quarkus live coding and edit any file within the IDE. The following output should be logged:

```
INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (Aesh InputStream Reader) Live reload
disabled
INFO  [org.ecl.che.inc.AnalyticsManager] (executor-thread-2) The received event is: Edit
Workspace File in Che
```

3.6.1.6.5. Implementing isEnabled()

For the purposes of the example, this method always returns **true** whenever it is called.

Example 3.18. AnalyticsManager.java

```
@Override
public boolean isEnabled() {
    return true;
}
```

It is possible to put more complex logic in **isEnabled()**. For example, the hosted OpenShift Dev Spaces Woopra backend checks that a configuration property exists before determining if the backend is enabled.

3.6.1.6.6. Implementing onEvent()

onEvent() sends the event received by the backend to the telemetry system. For the example application, it sends an HTTP POST payload to the /**event** endpoint from the telemetry server.

3.6.1.6.6.1. Sending a POST request to the example telemetry server

For the following example, the telemetry server application is deployed to OpenShift at the following URL: **http://little-telemetry-server-che.apps-crc.testing**, where **apps-crc.testing** is the ingress domain name of the OpenShift cluster.

1. Set up the RESTEasy REST Client by creating **TelemetryService.java**

   Example 3.19. TelemetryService.java

   ```
   package org.my.group;

   import java.util.Map;

   import javax.ws.rs.Consumes;
   import javax.ws.rs.POST;
   import javax.ws.rs.Path;
   import javax.ws.rs.core.MediaType;
   import javax.ws.rs.core.Response;

   import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

   @RegisterRestClient
   public interface TelemetryService {
       @POST
       @Path("/event") (1)
       @Consumes(MediaType.APPLICATION_JSON)
       Response sendEvent(Map<String, Object> payload);
   }
   ```

   (1) The endpoint to make the **POST** request to.

2. Specify the base URL for **TelemetryService** in the **src/main/resources/application.properties** file:

Example 3.20. **application.properties**

```
org.my.group.TelemetryService/mp-rest/url=http://little-telemetry-server-che.apps-crc.testing
```

3. Inject **TelemetryService** into **AnalyticsManager** and send a **POST** request in **onEvent()**

Example 3.21. **AnalyticsManager.java**

```
@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {
  @Inject
  @RestClient
  TelemetryService telemetryService;

...

@Override
public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent,
String resolution, Map<String, Object> properties) {
  Map<String, Object> payload = new HashMap<String, Object>(properties);
  payload.put("event", event);
  telemetryService.sendEvent(payload);
}
```

This sends an HTTP request to the telemetry server and automatically delays identical events for a small period of time. The default duration is 1500 milliseconds.

3.6.1.6.7. Implementing **increaseDuration()**

Many telemetry systems recognize event duration. The **AbstractAnalyticsManager** merges similar events that happen in the same frame of time into one event. This implementation of **increaseDuration()** is a no-op. This method uses the APIs of the user's telemetry provider to alter the event or event properties to reflect the increased duration of an event.

Example 3.22. **AnalyticsManager.java**

```
@Override
public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) {}
```

3.6.1.6.8. Implementing **onActivity()**

Set an inactive timeout limit, and use **onActivity()** to send a **WORKSPACE_INACTIVE** event if the last event time is longer than the timeout.

Example 3.23. **AnalyticsManager.java**

```
public class AnalyticsManager extends AbstractAnalyticsManager {
```

```
    ...

    private long inactiveTimeLimit = 60000 * 3;

    ...

    @Override
    public void onActivity() {
        if (System.currentTimeMillis() - lastEventTime >= inactiveTimeLimit) {
            onEvent(WORKSPACE_INACTIVE, lastOwnerId, lastIp, lastUserAgent, lastResolution,
commonProperties);
        }
    }
```

3.6.1.6.9. Implementing destroy()

When **destroy()** is called, send a **WORKSPACE_STOPPED** event and shutdown any resources such as connection pools.

Example 3.24. AnalyticsManager.java

```
@Override
public void destroy() {
    onEvent(WORKSPACE_STOPPED, lastOwnerId, lastIp, lastUserAgent, lastResolution,
commonProperties);
}
```

Running **mvn quarkus:dev** as described in Section 3.6.1.6.4, "Running the application within a DevWorkspace" and terminating the application with **Ctrl+C** sends a **WORKSPACE_STOPPED** event to the server.

3.6.1.6.10. Packaging the Quarkus application

See the Quarkus documentation for the best instructions to package the application in a container. Build and push the container to a container registry of your choice.

3.6.1.6.10.1. Sample Dockerfile for building a Quarkus image running with JVM

Example 3.25. Dockerfile.jvm

```
FROM registry.access.redhat.com/ubi8/openjdk-11:1.11

ENV LANG='en_US.UTF-8' LANGUAGE='en_US:en'

COPY --chown=185 target/quarkus-app/lib/ /deployments/lib/
COPY --chown=185 target/quarkus-app/*.jar /deployments/
COPY --chown=185 target/quarkus-app/app/ /deployments/app/
COPY --chown=185 target/quarkus-app/quarkus/ /deployments/quarkus/

EXPOSE 8080
USER 185
```

```
ENTRYPOINT ["java", "-Dquarkus.http.host=0.0.0.0", "-
Djava.util.logging.manager=org.jboss.logmanager.LogManager", "-
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}", "-jar",
"/deployments/quarkus-run.jar"]
```

To build the image, run:

```
mvn package && \
podman build -f src/main/docker/Dockerfile.jvm -t image:tag .
```

3.6.1.6.10.2. Sample Dockerfile for building a Quarkus native image

Example 3.26. Dockerfile.native

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
WORKDIR /work/
RUN chown 1001 /work \
    && chmod "g+rwX" /work \
    && chown 1001:root /work
COPY --chown=1001:root target/*-runner /work/application

EXPOSE 8080
USER 1001

CMD ["./application", "-Dquarkus.http.host=0.0.0.0", "-
Dquarkus.http.port=$DEVWORKSPACE_TELEMETRY_BACKEND_PORT}"]
```

To build the image, run:

```
mvn package -Pnative -Dquarkus.native.container-build=true && \
podman build -f src/main/docker/Dockerfile.native -t image:tag .
```

3.6.1.6.11. Creating a plugin.yaml for your plug-in

Create a **plugin.yaml** devfile v2 file representing a DevWorkspace plug-in that runs your custom backend in a DevWorkspace Pod. For more information about devfile v2, see Devfile v2 documentation

Example 3.27. plugin.yaml

```
schemaVersion: 2.1.0
metadata:
  name: devworkspace-telemetry-backend-plugin
  version: 0.0.1
  description: A Demo telemetry backend
  displayName: Devworkspace Telemetry Backend
components:
  - name: devworkspace-telemetry-backend-plugin
    attributes:
      workspaceEnv:
```

```
      - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
        value: '4167'
    container:
      image: YOUR IMAGE          (1)
      env:
        - name: WELCOME_MESSAGE  (2)
          value: 'hello world!'
```

1. Specify the container image built from Section 3.6.1.6.10, “Packaging the Quarkus application”.
2. Set the value for the **welcome.message** optional configuration property from Example 4.

Typically, the user deploys this file to a corporate web server. This guide demonstrates how to create an Apache web server on OpenShift and host the plug-in there.

Create a ConfigMap referencing the new **plugin.yaml** file.

```
$ oc create configmap --from-file=plugin.yaml -n openshift-devspaces telemetry-plugin-yaml
```

Create a deployment, a service, and a route to expose the web server. The deployment references this ConfigMap and places it in the **/var/www/html** directory.

Example 3.28. manifest.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      volumes:
        - name: plugin-yaml
          configMap:
            name: telemetry-plugin-yaml
            defaultMode: 420
      containers:
        - name: apache
          image: 'registry.redhat.io/rhscl/httpd-24-rhel7:latest'
          ports:
            - containerPort: 8080
              protocol: TCP
          resources: {}
          volumeMounts:
            - name: plugin-yaml
              mountPath: /var/www/html
```

```
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxUnavailable: 25%
        maxSurge: 25%
    revisionHistoryLimit: 10
    progressDeadlineSeconds: 600
  ---
  kind: Service
  apiVersion: v1
  metadata:
    name: apache
  spec:
    ports:
      - protocol: TCP
        port: 8080
        targetPort: 8080
    selector:
      app: apache
    type: ClusterIP
  ---
  kind: Route
  apiVersion: route.openshift.io/v1
  metadata:
    name: apache
  spec:
    host: apache-che.apps-crc.testing
    to:
      kind: Service
      name: apache
      weight: 100
    port:
      targetPort: 8080
    wildcardPolicy: None
```

```
$ oc apply -f manifest.yaml
```

Verification steps

After the deployment has started, confirm that **plugin.yaml** is available in the web server:

```
$ curl apache-che.apps-crc.testing/plugin.yaml
```

3.6.1.6.12. Specifying the telemetry plug-in in a DevWorkspace

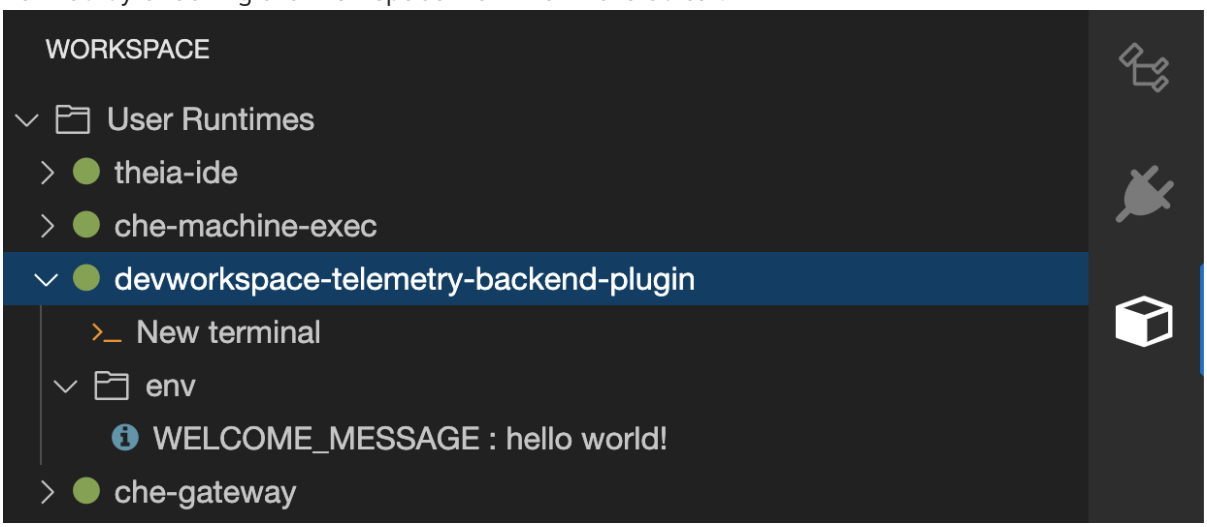
1. Add the following to the **components** field of an existing DevWorkspace:

```
components:
  ...
  - name: telemetry-plug-in
    plugin:
      uri: http://apache-che.apps-crc.testing/plugin.yaml
```

2. Start the DevWorkspace from the OpenShift Dev Spaces dashboard.

Verification steps

1. Verify that the **telemetry-plug-in** container is running in the DevWorkspace pod. Here, this is verified by checking the Workspace view within the editor.



2. Edit files within the editor and observe their events in the example telemetry server's logs.

3.6.1.6.13. Applying the telemetry plug-in for all DevWorkspaces

Set the telemetry plug-in as a default plug-in. Default plug-ins are applied on DevWorkspace startup for new and existing DevWorkspaces.

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    workspacesDefaultPlugins:
    - editor: eclipse/che-theia/next      (1)
      plugins:                            (2)
      - 'http://apache-che.apps-crc.testing/plugin.yaml'
```

(1) The editorId to set default plug-ins for.

(2) List of URLs to devfile v2 plug-ins.

Additional resources

- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource" .

Verification steps

1. Start a new or existing DevWorkspace from the Red Hat OpenShift Dev Spaces dashboard.

2. Verify that the telemetry plug-in is working by following the verification steps for Section 3.6.1.6.12, "Specifying the telemetry plug-in in a DevWorkspace" .

3.6.2. Configuring server logging

It is possible to fine-tune the log levels of individual loggers available in the OpenShift Dev Spaces server.

The log level of the whole OpenShift Dev Spaces server is configured globally using the **cheLogLevel** configuration property of the Operator. See Section 3.1.3, "**CheCluster** Custom Resource fields reference". To set the global log level in installations not managed by the Operator, specify the **CHE_LOG_LEVEL** environment variable in the **che** ConfigMap.

It is possible to configure the log levels of the individual loggers in the OpenShift Dev Spaces server using the **CHE_LOGGER_CONFIG** environment variable.

3.6.2.1. Configuring log levels

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    customCheProperties:
      CHE_LOGGER_CONFIG: "<key1=value1,key2=value2>" 1
```

1 Comma-separated list of key-value pairs, where keys are the names of the loggers as seen in the OpenShift Dev Spaces server log output and values are the required log levels.

Example 3.29. Configuring debug mode for the WorkspaceManager

```
spec:
  server:
    customCheProperties:
      CHE_LOGGER_CONFIG:
"org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.6.2.2. Logger naming

The names of the loggers follow the class names of the internal server classes that use those loggers.

3.6.2.3. Logging HTTP traffic

Procedure

- To log the HTTP traffic between the OpenShift Dev Spaces server and the API server of the Kubernetes or OpenShift cluster, configure the **CheCluster** Custom Resource. See Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource” .

```
spec:
  server:
    customCheProperties:
      CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.6.3. Collecting logs using dsc

An installation of Red Hat OpenShift Dev Spaces consists of several containers running in the OpenShift cluster. While it is possible to manually collect logs from each running container, **dsc** provides commands which automate the process.

Following commands are available to collect Red Hat OpenShift Dev Spaces logs from the OpenShift cluster using the **dsc** tool:

dsc server:logs

Collects existing Red Hat OpenShift Dev Spaces server logs and stores them in a directory on the local machine. By default, logs are downloaded to a temporary directory on the machine. However, this can be overwritten by specifying the **-d** parameter. For example, to download Che logs to the **/home/user/che-logs/** directory, use the command

```
dsc server:logs -d /home/user/che-logs/
```

When run, **dsc server:logs** prints a message in the console specifying the directory that will store the log files:

```
Red Hat OpenShift Dev Spaces logs will be available in '/tmp/chectl-logs/1648575098344'
```

If Red Hat OpenShift Dev Spaces is installed in a non-default project, **dsc server:logs** requires the **-n <NAMESPACE>** paremeter, where **<NAMESPACE>** is the OpenShift project in which Red Hat OpenShift Dev Spaces was installed. For example, to get logs from OpenShift Dev Spaces in the **my-namespace** project, use the command

```
dsc server:logs -n my-namespace
```

dsc server:deploy

Logs are automatically collected during the OpenShift Dev Spaces installation when installed using **dsc**. As with **dsc server:logs**, the directory logs are stored in can be specified using the **-d** parameter.

Additional resources

- "`dsc` reference documentation"

3.6.4. Monitoring OpenShift Dev Spaces with Prometheus and Grafana

You can collect and view the OpenShift Dev Spaces metrics with a running instance of Prometheus and Grafana on the cluster.

- Section 3.6.4.1, “Installing Prometheus and Grafana”
- Section 3.6.4.2, “Monitoring the DevWorkspace Operator”
- Section 3.6.4.3, “Monitoring OpenShift Dev Spaces Server”

3.6.4.1. Installing Prometheus and Grafana

You can install Prometheus and Grafana by applying **template.yaml** that consists of a Deployment and Service for both Prometheus and Grafana.

Alternatively, you can use the Prometheus Operator and Grafana Operator.

Prerequisites

- oc

Procedure

To install Prometheus and Grafana by using **template.yaml**:

- Apply **template.yaml** to the cluster by running **oc apply -f template.yaml**.

Example 3.30. template.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  labels:
    app: grafana
spec:
  ports:
  - name: 3000-tcp
    port: 3000
    protocol: TCP
    targetPort: 3000
  selector:
    app: grafana
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  labels:
    app: prometheus
spec:
  ports:
  - name: 9090-tcp
    port: 9090
```

```
    protocol: TCP
    targetPort: 9090
  selector:
    app: prometheus
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: grafana
  name: grafana
spec:
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - image: registry.redhat.io/rhel8/grafana:7
        name: grafana
        ports:
        - containerPort: 3000
          protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus
  name: prometheus
spec:
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - image: quay.io/prometheus/prometheus:v2.36.0
        name: prometheus
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: /prometheus
          name: volume-data
        - mountPath: /etc/prometheus/prometheus.yml
          name: volume-config
          subPath: prometheus.yml
      volumes:
      - emptyDir: {}
```

```
        name: volume-data
      - configMap:
          defaultMode: 420
          name: prometheus-config
        name: volume-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
---
```

Additional resources

- First steps with Prometheus
- Installing Grafana

3.6.4.2. Monitoring the DevWorkspace Operator

You can configure an example monitoring stack to process metrics exposed by the DevWorkspace Operator.

3.6.4.2.1. Collecting DevWorkspace Operator metrics with Prometheus

To use Prometheus to collect, store, and query metrics about the DevWorkspace Operator:

Prerequisites

- The **devworkspace-controller-metrics** Service is exposing metrics on port **8443**.
- The **devworkspace-webhookserver** Service is exposing metrics on port **9443**. By default, the Service exposes metrics on port **9443**.
- Prometheus 2.26.0 or later is running. The Prometheus console is running on port **9090** with a corresponding **service** and **route**. See First steps with Prometheus.

Procedure

1. Create a **ClusterRoleBinding** to bind the **ServiceAccount** associated with Prometheus to the devworkspace-controller-metrics-reader **ClusterRole**.

NOTE

Without the **ClusterRoleBinding**, you cannot access DevWorkspace metrics because access is protected with role-based access control (RBAC).

Example 3.31. ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: devworkspace-controller-metrics-reader
rules:
- nonResourceURLs:
  - /metrics
  verbs:
  - get
```

Example 3.32. ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: devworkspace-controller-metrics-binding
subjects:
  - kind: ServiceAccount
    name: <ServiceAccount_name_associated_with_the_Prometheus_Pod>
    namespace: <Prometheus_namespace>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: devworkspace-controller-metrics-reader
```

2. Configure Prometheus to scrape metrics from port **8443** exposed by the **devworkspace-controller-metrics** Service and from port **9443** exposed by the **devworkspace-webhookserver** Service.

Example 3.33. Prometheus configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval: 5s (1)
      evaluation_interval: 5s (2)
    scrape_configs: (3)
      - job_name: 'DevWorkspace'
        scheme: https
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-controller-metrics:8443'] (4)
      - job_name: 'DevWorkspace webhooks'
        scheme: https
        authorization:
```

```
      type: Bearer
      credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
    tls_config:
      insecure_skip_verify: true
    static_configs:
      - targets: ['devworkspace-webhookserver:9443'] 5
```

1. The rate at which a target is scraped.
2. The rate at which the recording and alerting rules are re-checked.
3. The resources that Prometheus monitors. In the default configuration, two jobs, **DevWorkspace** and **DevWorkspace webhooks**, scrape the time series data exposed by the **devworkspace-controller-metrics** and **devworkspace-webhookserver** Services.
4. The scrape metrics from port **8443**.
5. The scrape metrics from port **9443**.

Verification steps

1. Use the Prometheus console to view and query metrics:
   - View the metrics at **http://*<prometheus_url>*/metrics**.
   - Query metrics from **http://*<prometheus_url>*/graph**. For more information, see Using the expression browser.
2. Verify that all targets are up by viewing the targets endpoint at **http://*<prometheus-url>*/targets**.

Additional resources

- Configuring Prometheus
- Querying Prometheus
- Prometheus metric types

3.6.4.2.2. DevWorkspace-specific metrics

The following tables describe the DevWorkspace-specific metrics exposed by the **devworkspace-controller-metrics** Service.

Table 3.12. Metrics

Name	Type	Description	Labels
devworkspace_started_total	Counter	Number of DevWorkspace starting events.	**source**, **routingclass**

Name	Type	Description	Labels
devworkspace_started_success_total	Counter	Number of DevWorkspaces successfully entering the **Running** phase.	**source**, **routingclass**
devworkspace_fail_total	Counter	Number of failed DevWorkspaces.	**source**, **reason**
devworkspace_startup_time	Histogram	Total time taken to start a DevWorkspace, in seconds.	**source**, **routingclass**

Table 3.13. Labels

Name	Description	Values
source	The **controller.devfile.io/devworkspace-source** label of the DevWorkspace.	**string**
routingclass	The **spec.routingclass** of the DevWorkspace.	**"basic|cluster|cluster-tls|web-terminal"**
reason	The workspace startup failure reason.	**"BadRequest|InfrastructureFailure|Unknown"**

Table 3.14. Startup failure reasons

Name	Description
BadRequest	Startup failure due to an invalid devfile used to create a DevWorkspace.
InfrastructureFailure	Startup failure due to the following errors: **CreateContainerError**, **RunContainerError**, **FailedScheduling**, **FailedMount**.
Unknown	Unknown failure reason.

3.6.4.2.3. Viewing DevWorkspace Operator metrics on Grafana dashboards

To view the DevWorkspace Operator metrics on Grafana with the example dashboard:

Prerequisites

- Prometheus is collecting metrics. See Section 3.6.4.2.1, "Collecting DevWorkspace Operator metrics with Prometheus".
- Grafana version 7.5.3 or later.
- Grafana is running on port **3000** with a corresponding **service** and **route**. See Installing Grafana.

Procedure

1. Add the data source for the Prometheus instance. See Creating a Prometheus data source .
2. Import the example **grafana-dashboard.json** dashboard.

Verification steps

- Use the Grafana console to view the DevWorkspace Operator metrics dashboard. See Section 3.6.4.2.4, "Grafana dashboard for the DevWorkspace Operator" .

Additional resources

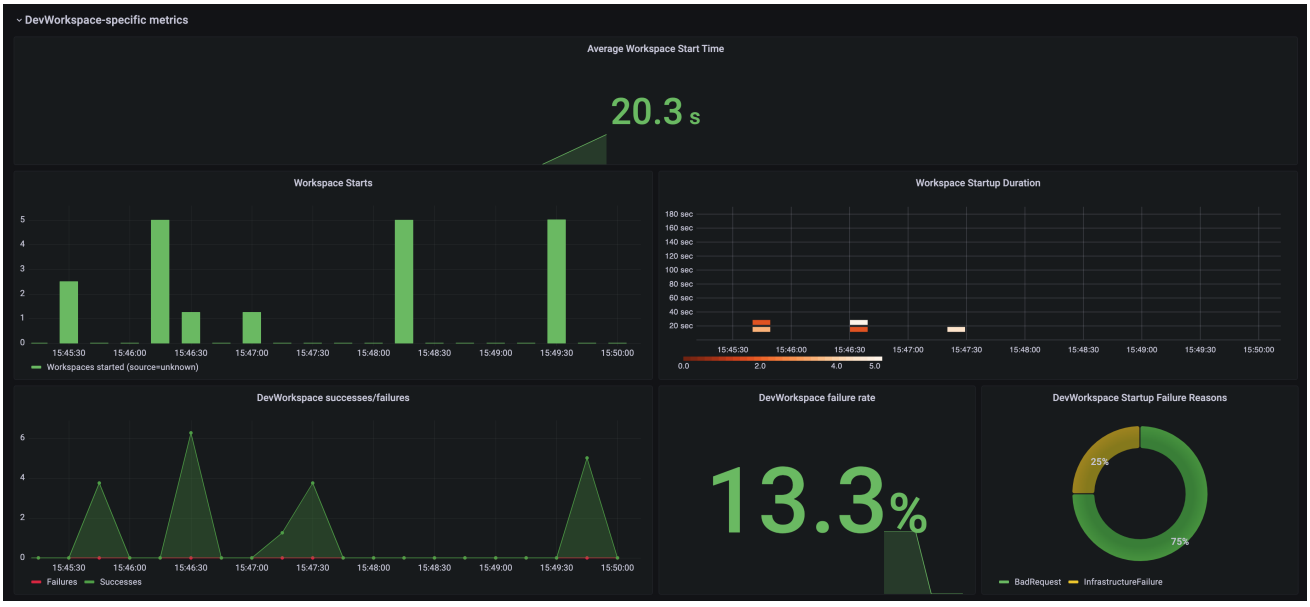
- Prometheus data source
- Import dashboard

3.6.4.2.4. Grafana dashboard for the DevWorkspace Operator

The example Grafana dashboard based on **grafana-dashboard.json** displays the following metrics from the DevWorkspace Operator.

3.6.4.2.4.1. The DevWorkspace-specific metrics panel

Figure 3.1. The DevWorkspace-specific metrics panel



Average workspace start time

The average workspace startup duration.

Workspace starts

The number of successful and failed workspace startups.

Workspace startup duration

A heatmap that displays workspace startup duration.

DevWorkspace successes / failures

A comparison between successful and failed DevWorkspace startups.

DevWorkspace failure rate

The ratio between the number of failed workspace startups and the number of total workspace startups.

DevWorkspace startup failure reasons

A pie chart that displays the distribution of workspace startup failures:

- **BadRequest**
- **InfrastructureFailure**
- **Unknown**

3.6.4.2.4.2. The Operator metrics panel (part 1)

Figure 3.2. The Operator metrics panel (part 1)

Webhooks in flight

A comparison between the number of different webhook requests.

Work queue duration

A heatmap that displays how long the reconcile requests stay in the work queue before they are handled.

Webhooks latency (/mutate)

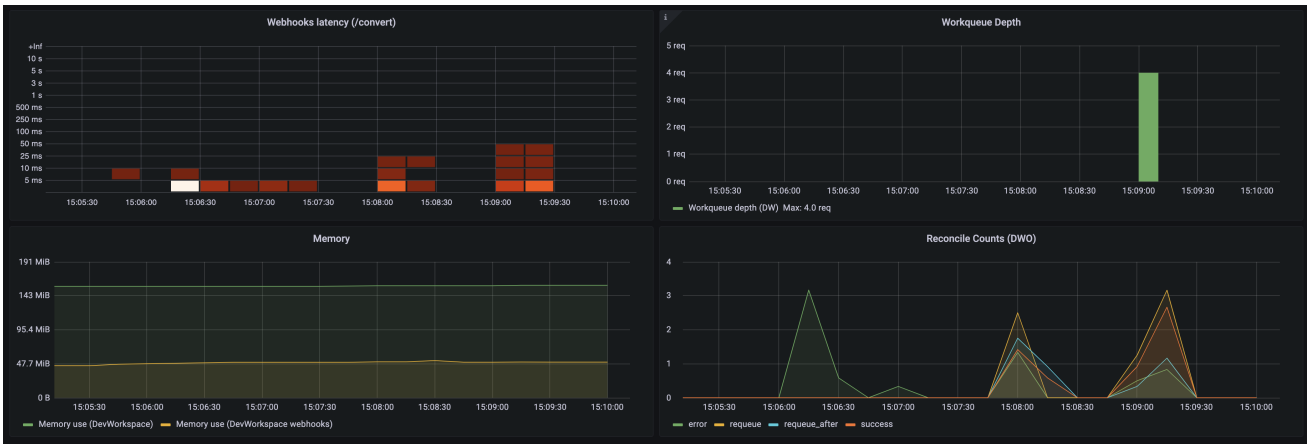
A heatmap that displays the **/mutate** webhook latency.

Reconcile time

A heatmap that displays the reconcile duration.

3.6.4.2.4.3. The Operator metrics panel (part 2)

Figure 3.3. The Operator metrics panel (part 2)



Webhooks latency (/convert)

A heatmap that displays the /**convert** webhook latency.

Work queue depth

The number of reconcile requests that are in the work queue.

Memory

Memory usage for the DevWorkspace controller and the DevWorkspace webhook server.

Reconcile counts (DWO)

The average per-second number of reconcile counts for the DevWorkspace controller.

3.6.4.3. Monitoring OpenShift Dev Spaces Server

You can configure OpenShift Dev Spaces to expose JVM metrics such as JVM memory and class loading for OpenShift Dev Spaces Server.

3.6.4.3.1. Enabling and exposing OpenShift Dev Spaces Server metrics

OpenShift Dev Spaces exposes the JVM metrics on port **8087** of the **che-host** Service. You can configure this behaviour.

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”.

```
spec:
  metrics:
    enable: <boolean> 1
```

1 **true** to enable, **false** to disable.

3.6.4.3.2. Collecting OpenShift Dev Spaces Server metrics with Prometheus

To use Prometheus to collect, store, and query JVM metrics for OpenShift Dev Spaces Server:

Prerequisites

- OpenShift Dev Spaces is exposing metrics on port **8087**. See Enabling and exposing OpenShift Dev Spaces server JVM metrics.
- Prometheus 2.26.0 or later is running. The Prometheus console is running on port **9090** with a corresponding **service** and **route**. See First steps with Prometheus.

Procedure

- Configure Prometheus to scrape metrics from port **8087**.

Example 3.34. Prometheus configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval:     5s          (1)
      evaluation_interval: 5s          (2)
    scrape_configs:                    (3)
      - job_name: 'che'
        static_configs:
          - targets: ['[che-host]:8087'] (4)
```

1. The rate at which a target is scraped.
2. The rate at which the recording and alerting rules are re-checked.
3. The Resources that Prometheus monitors. In the default configuration, a single job, **che**, scrapes the time series data exposed by OpenShift Dev Spaces Server.
4. The scrape metrics from port **8087**.

Verification steps

1. View the metrics in the Prometheus console at **http://*<prometheus-url>*/metrics**.
2. Query the metrics in the Prometheus console from **http://*<prometheus-url>*/graph**. For more information, see Using the expression browser.
3. Verify that all targets are up by viewing the targets endpoint at **http://*<prometheus-url>*/targets**.

Additional resources

- Configuring Prometheus
- Querying Prometheus
- Prometheus metric types

3.6.4.3.3. Viewing OpenShift Dev Spaces Server metrics on Grafana dashboards

To view the OpenShift Dev Spaces Server metrics on Grafana:

Prerequisites

- Prometheus is collecting metrics on the OpenShift Dev Spaces cluster. See Section 3.6.4, "Monitoring OpenShift Dev Spaces with Prometheus and Grafana".
- Grafana 6.0 or later is running on port **3000** with a corresponding **service** and **route**. See Installing Grafana.

Procedure

1. Add the data source for the Prometheus instance. See Creating a Prometheus data source .
2. Import the example dashboard. See Import dashboard.
3. View the OpenShift Dev Spaces JVM metrics in the Grafana console:

Figure 3.4. OpenShift Dev Spaces server JVM dashboard

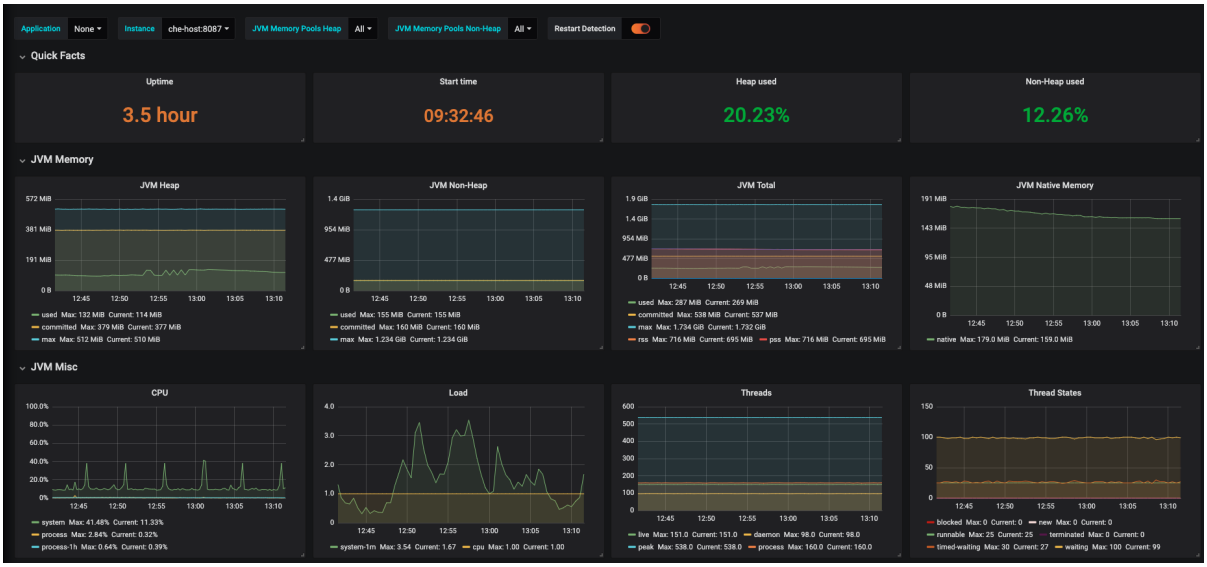


Figure 3.5. Quick Facts

Figure 3.6. JVM Memory

Figure 3.7. JVM Misc

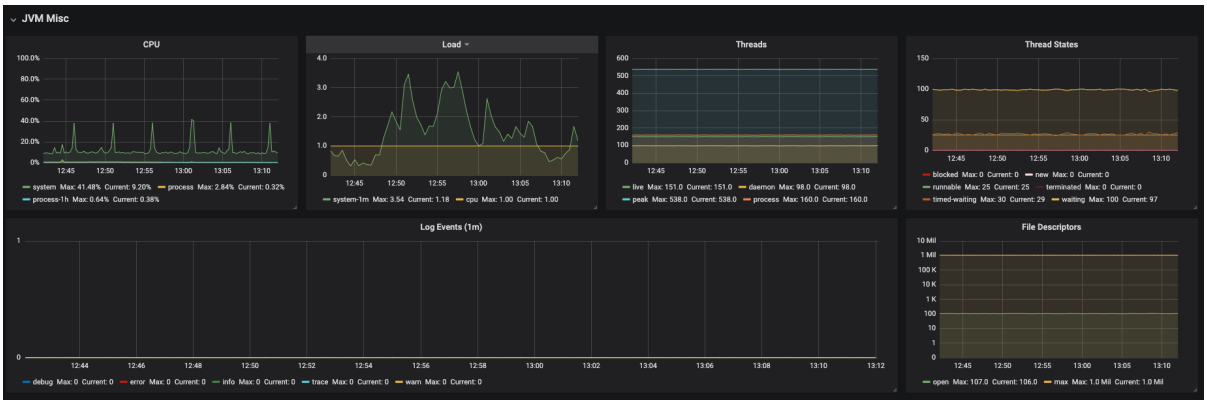


Figure 3.8. JVM Memory Pools (heap)

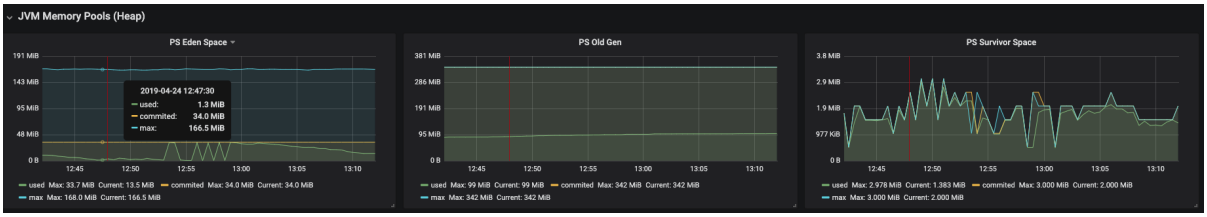


Figure 3.9. JVM Memory Pools (Non-Heap)

Figure 3.10. Garbage Collection

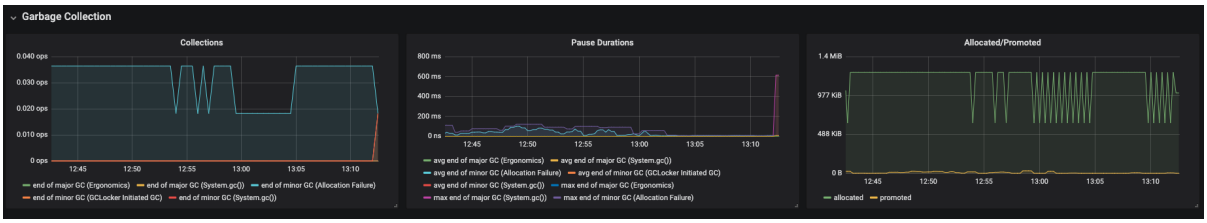


Figure 3.11. Class loading

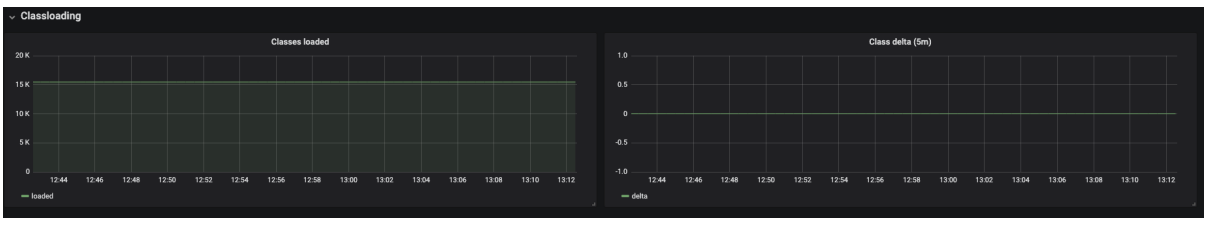
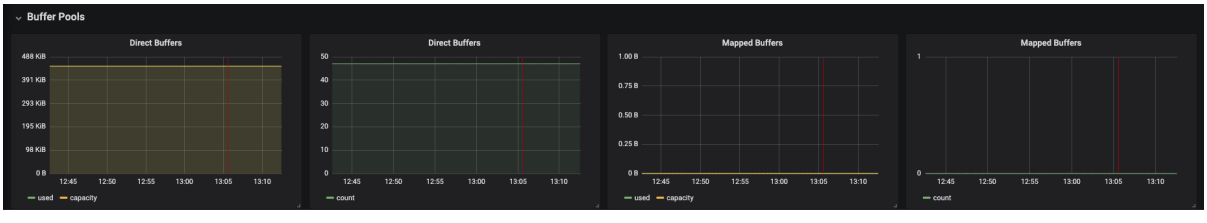


Figure 3.12. Buffer Pools



3.7. CONFIGURING NETWORKING

3.7.1. Configuring Red Hat OpenShift Dev Spaces server hostname

This procedure describes how to configure OpenShift Dev Spaces to use custom hostname.

Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.
- The certificate and the private key files are generated.

IMPORTANT

To generate the pair of a private key and certificate, the same certification authority (CA) must be used as for other OpenShift Dev Spaces hosts.

IMPORTANT

Ask a DNS provider to point the custom hostname to the cluster ingress.

Procedure

1. Pre-create a project for OpenShift Dev Spaces:

```
$ oc create project openshift-devspaces
```

2. Create a TLS secret:

```
$ oc create secret TLS <tls-secret-name> \ 1
--key <key-file> \ 2
--cert <cert-file> \ 3
-n openshift-devspaces
```

1. The TLS secret name
2. A file with the private key
3. A file with the certificate

3. Add the required labels to the secret:

```
$ oc label secret <tls-secret-name> \ 1
app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

```
```

1 The TLS secret name

4. Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    cheHost: <hostname>          1
    cheHostTLSSecret: <secret>   2
```

1 Custom Red Hat OpenShift Dev Spaces server hostname

2 The TLS secret name

5. If OpenShift Dev Spaces has been already deployed, wait until the rollout of all OpenShift Dev Spaces components finishes.

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

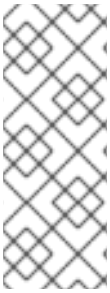
3.7.2. Importing untrusted TLS certificates to OpenShift Dev Spaces

By default, external communications between OpenShift Dev Spaces components are encrypted with TLS. Communications of OpenShift Dev Spaces components with external services such as proxies, source code repositories, and identity provider might also require TLS. All communications encrypted with TLS require the use of TLS certificates signed by trusted Certificate Authorities (CA).

When the certificates used by OpenShift Dev Spaces components or by an external service are signed by an untrusted CA, you must import the CA certificate into the OpenShift Dev Spaces instance so that every OpenShift Dev Spaces component treats the certificates as signed by a trusted CA. You have to do this in the following cases:

- The underlying OpenShift cluster uses TLS certificates signed by an untrusted CA. OpenShift Dev Spaces server or workspace components connect to external OIDC providers or a Git server that use TLS certificates signed by an untrusted CA.

OpenShift Dev Spaces uses labeled ConfigMaps in project as sources for TLS certificates. The ConfigMaps can have an arbitrary number of keys with a random number of certificates each.



NOTE

When an OpenShift cluster contains cluster-wide trusted CA certificates added through the cluster-wide-proxy configuration, OpenShift Dev Spaces Operator detects them and automatically injects them into a ConfigMap. OpenShift Dev Spaces automatically labels the ConfigMap with the **config.openshift.io/inject-trusted-cabundle="true"** label. Based on this annotation, OpenShift automatically injects the cluster-wide trusted CA certificates inside the **ca-bundle.crt** key of ConfigMap.

IMPORTANT

Some OpenShift Dev Spaces components require a full certificate chain to trust the endpoint. If the cluster is configured with an intermediate certificate, add the whole chain, including self-signed root, to OpenShift Dev Spaces.

3.7.2.1. Adding new CA certificates into OpenShift Dev Spaces

The following procedure is applicable for already installed and running instances and for instances that are to be installed.

Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.
- Namespace for OpenShift Dev Spaces exists.

Procedure

1. Save the certificates you need to import to a local file system.

   CAUTION

   - A certificate with the introductory phrase **BEGIN TRUSTED CERTIFICATE** is likely in the PEM **TRUSTED CERTIFICATE** format, which is not supported by Java. Convert it to the supported **CERTIFICATE** format with the following command:
     - **openssl x509 -in cert.pem -out cert.cer**

2. Create a new ConfigMap with the required TLS certificates:

   ```
   $ oc create configmap custom-certs --from-file=<bundle-file-path> -n=openshift-devspaces
   ```

   To apply more than one bundle, add another **-from-file=<*bundle-file-path*>**. Alternatively, create another ConfigMap.

3. Label created ConfigMaps with the **app.kubernetes.io/part-of=che.eclipse.org** and **app.kubernetes.io/component=ca-bundle** labels:

   ```
   $ oc label configmap custom-certs app.kubernetes.io/part-of=che.eclipse.org
   app.kubernetes.io/component=ca-bundle -n <devspaces-namespace-name>
   ```

4. Deploy OpenShift Dev Spaces if it hasn't been deployed before. Otherwise wait until the rollout of OpenShift Dev Spaces components finishes.

5. Restart running workspaces for the changes to take effect.

3.7.2.2. Troubleshooting imported certificate issues

If issues occur after adding the certificates, verify the specified values at the OpenShift Dev Spaces instance level and workspace level.

Verifying imported certificates at the OpenShift Dev Spaces instance level

- In case of a OpenShift Dev Spaces Operator deployment, the namespace where the **CheCluster** is located contains labeled ConfigMaps with the correct content:

```
$ oc get cm --selector=app.kubernetes.io/component=ca-bundle,app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

  Check the content of ConfigMap by entering:

```
$ oc get cm <name> -n openshift-devspaces -o yaml
```

- OpenShift Dev Spaces Pod Volumes list contains a volume that uses **ca-certs-merged** ConfigMap as data-source. To get the list of Volumes of the OpenShift Dev Spaces Pod, run:

```
$ oc get pod -o json <devspaces-pod-name> -n openshift-devspaces | jq .spec.volumes
```

- OpenShift Dev Spaces mounts certificates in the /**public-certs**/ folder of the OpenShift Dev Spaces server container. To view the list of files in this folder, enter:

```
$ oc exec -t <devspaces-pod-name> -n openshift-devspaces -- ls /public-certs/
```

- In the OpenShift Dev Spaces server logs, there is a line for every certificate added to the Java truststore, including configured OpenShift Dev Spaces certificates. View them:

```
$ oc logs <devspaces-pod-name> -n openshift-devspaces
```

- OpenShift Dev Spaces server Java truststore contains the certificates. The certificates SHA1 fingerprints are among the list of the SHA1 of the certificates included in the truststore. View the list:

```
$ oc exec -t <devspaces-pod-name> -n openshift-devspaces -- keytool -list -keystore /home/user/cacerts
Your keystore contains 141 entries:
+
(...)
```

  To get the SHA1 hash of a certificate on the local filesystem, run:

```
$ openssl x509 -in <certificate-file-path> -fingerprint -noout
SHA1 Fingerprint=3F:DA:BF:E7:A7:A7:90:62:CA:CF:C7:55:0E:1D:7D:05:16:7D:45:60
```

Verifying imported certificates at the workspace level

- Start a workspace, obtain the project name in which it has been created and wait for the workspace to be started.

- Get the name of the workspace Pod:

```
$ oc get pods -o=jsonpath='{.items[0].metadata.name}' -n <workspace namespace> | grep '^workspace.*'
```

- Get the name of the Che-Theia IDE container in the workspace Pod:

```
$ oc get -o json pod <workspace pod name>  -n <workspace namespace> | \
    jq -r '.spec.containers[] | select(.name | startswith("theia-ide")).name'
```

- Look for a **ca-certs** ConfigMap inside the workspace namespace:

```
$ oc get cm ca-certs <workspace namespace>
```

- Check that the entries in the **ca-certs** ConfigMap contain all the additional entries you added before. In addition, it can contain **ca-bundle.crt** reserved entry. View the entries:

```
$ oc get cm ca-certs -n <workspace namespace> -o json | jq -r '.data | keys[]'
ca-bundle.crt
source-config-map-name.data-key.crt
```

- Confirm that the **ca-certs** ConfigMap is added as a volume in the workspace Pod:

```
$ oc get -o json pod <workspace pod name> -n <workspace namespace> | \
    jq '.spec.volumes[] | select(.configMap.name == "ca-certs")'
{
  "configMap": {
    "defaultMode": 420,
    "name": "ca-certs"
  },
  "name": "che-self-signed-certs"
}
```

- Confirm that the volume is mounted into containers, especially in the Che-Theia IDE container:

```
$ oc get -o json pod <workspace pod name> -n <workspace namespace> | \
   jq '.spec.containers[] | select(.name == "<theia ide container name>").volumeMounts[] |
select(.name == "che-self-signed-certs")'
{
  "mountPath": "/public-certs",
  "name": "che-self-signed-certs",
  "readOnly": true
}
```

- Inspect the /**public-certs** folder in the Che-Theia IDE container and check if its contents match the list of entries in the **ca-certs** ConfigMap:

```
$ oc exec <workspace pod name> -c <theia ide container name> -n <workspace
namespace> -- ls /public-certs
ca-bundle.crt
source-config-map-name.data-key.crt
```

3.7.3. Adding labels and annotations to OpenShift Route

You can configure OpenShift Route labels and annotations, if your organization requires them.

Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.

- An instance of OpenShift Dev Spaces running in OpenShift.

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    cheServerIngress:
      labels: <labels> (1)
      annotations: <annotations> (2)
    customCheProperties:
      CHE_INFRA_KUBERNETES_INGRESS_LABELS: <labels> (3)
      CHE_INFRA_KUBERNETES_INGRESS_ANNOTATIONS__JSON: "<annotations>" (4)
```

(1) (3) A comma-separated list of labels for OpenShift Route: **key1=value1,key2=value2**.

(2) (4) Annotations for OpenShift Route in JSON format: **{"key1": "value1", "key2" : "value2"}**.

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.7.4. Configuring OpenShift Route to work with Router Sharding

You can configure labels, annotations, and domains for OpenShift Route to work with Router Sharding.

Prerequisites

- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.
- **dsc**. See: Section 2.1, "Install the dsc management tool" .

Procedure

- Configure the **CheCluster** Custom Resource. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource".

```
spec:
  server:
    cheServerRoute:
      labels: <labels> (1)
      domain: <domain> (2)
      annotations: (3)
        key1: value1
        key2: value2
    customCheProperties:
      CHE_INFRA_OPENSHIFT_ROUTE_LABELS: <labels> (4)
      CHE_INFRA_OPENSHIFT_ROUTE_HOST_DOMAIN__SUFFIX: <domain> (5)
```

-

1 4 A comma-separated list of labels that the target ingress controller uses to filter the set of Routes to service.

2 5 The DNS name serviced by the target ingress controller.

3 An unstructured key value map stored with a resource.`

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.8. CONFIGURING STORAGE

- Section 3.8.1, “Configuring storage classes”

3.8.1. Configuring storage classes

To configure OpenShift Dev Spaces to use a configured infrastructure storage, install OpenShift Dev Spaces using storage classes. This is especially useful when a user wants to bind a persistent volume provided by a non-default provisioner. To do so, a user binds this storage for the OpenShift Dev Spaces data saving and sets the parameters for that storage. These parameters can determine the following:

- A special host path
- A storage capacity
- A volume mod
- Mount options
- A file system
- An access mode
- A storage type
- And many others

OpenShift Dev Spaces has two components that require persistent volumes to store data:

- A PostgreSQL database.
- A OpenShift Dev Spaces workspaces. OpenShift Dev Spaces workspaces store source code using volumes, for example /**projects** volume.

NOTE

OpenShift Dev Spaces workspaces source code is stored in the persistent volume only if a workspace is not ephemeral.

Persistent volume claims facts:

- OpenShift Dev Spaces does not create persistent volumes in the infrastructure.
- OpenShift Dev Spaces uses persistent volume claims (PVC) to mount persistent volumes.
- The OpenShift Dev Spaces server creates persistent volume claims.
  A user defines a storage class name in the OpenShift Dev Spaces configuration to use the storage classes feature in the OpenShift Dev Spaces PVC. With storage classes, a user configures infrastructure storage in a flexible way with additional storage parameters. It is also possible to bind a static provisioned persistent volumes to the OpenShift Dev Spaces PVC using the class name.

Procedure

Use CheCluster Custom Resource definition to define storage classes:

1. Define storage class names: configure the **CheCluster** Custom Resource, and install OpenShift Dev Spaces. See Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”.

```
spec:
  storage:
    # keep blank unless you need to use a non default storage class for PostgreSQL PVC
    postgresPVCStorageClassName: 'postgres-storage'
    # keep blank unless you need to use a non default storage class for workspace PVC(s)
    workspacePVCStorageClassName: 'workspace-storage'
```

2. Define the persistent volume for a PostgreSQL database in a **che-postgres-pv.yaml** file:

   che-postgres-pv.yaml file

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv-volume
  labels:
    type: local
spec:
  storageClassName: postgres-storage
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/che/postgres"
```

3. Define the persistent volume for a OpenShift Dev Spaces workspace in a **che-postgres-pv.yaml** file:

   che-workspace-pv.yaml file

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: workspace-pv-volume
  labels:
```

```
    type: local
spec:
  storageClassName: workspace-storage
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/che/workspace"
```

4. Bind the two persistent volumes:

```
$ kubectl apply -f che-workspace-pv.yaml -f che-postgres-pv.yaml
```

NOTE

You must provide valid file permissions for volumes. You can do it using storage class configuration or manually. To manually define permissions, define **storageClass#mountOptions uid** and **gid**. PostgreSQL volume requires **uid=26** and **gid=26**.

Additional resources

- Section 3.1.1, “Using dsc to configure the **CheCluster** Custom Resource during installation”
- Section 3.1.2, “Using the CLI to configure the CheCluster Custom Resource”

3.9. BRANDING

- Section 3.9.1, “Branding Che-Theia”

3.9.1. Branding Che-Theia

This chapter describes how to customize the Che-Theia interface and branding. Customization is possible for the following elements:

- **Welcome** page and **About** dialog:
  - Product name
  - Product logo
  - Description
  - List of helpful resources (**Help** section of the **Welcome** page)

To start using the customized Che-Theia:

1. Build a container image with the customized Che-Theia.
2. Define an editor **meta.yaml** that uses the custom image.
3. Create a workspace from a devfile using the custom editor.

3.9.1.1. Defining custom branding values for Che-Theia

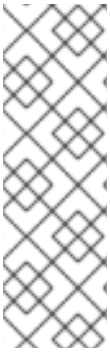
This section describes how to customize definitions of basic branding elements of Che-Theia.

Procedure

Create a **product.json** file with a new name of the product, logo, description, and list of hyperlinks on the **Welcome** page (an example of **product.json**:

```
{
   "name": "Red Hat OpenShift Dev Spaces", 1
   "icon": "icon.png", 2
   "logo": { 3
      "dark": "logo-light.png",
      "light": "logo-dark.png"
   },
   "welcome": { 4
      "title": "Welcome to Your Workspace",
      "links": ["website", "documentation"]
   },
   "links": { 5
      "website": {
         "name": "Discover Red Hat OpenShift Dev Spaces",
         "url": "https://developers.redhat.com/products/openshift_dev_spaces/overview"
      },
      "documentation": {
         "name": "Browse Documentation",
         "url": "https://www.redhat.com/docs"
      }
   }
}
```

1. **name**: tab title for the **Welcome** page and the **About** dialog.
2. **icon**: icon for the **Welcome** page tab title.
3. **logo**: product logo for dark and light themes on the **Welcome** page (maximum height 80 pixels) and in the **About** dialog (maximum height 100 pixels). Use an image with a transparent background. Define a relative path, an absolute path, or a URL to an image.
4. **welcome**: the behavior of the **Welcome** page. Customize the invitation title and the links in the **Help** section. When the **welcome/links** property is not defined, the **Welcome** page displays the links from the **links** section.
5. **links**: list of helpful resources for the product. Use tags to group links to make them easier to find.



NOTE

To use only one logo image for both dark and light themes:

```
{
    ...
    "logo": "product-logo.png"
    ...
}
```

3.9.1.2. Building a Che-Theia container image with custom branding

This section describes how to build a Che-Theia container image with custom branding applied.

Prerequisites

- A **product.json** file with custom branding definitions.

Procedure

1. Download an example **Dockerfile**.
2. In the **Dockerfile** directory, create a **branding**/ sub-directory. Place the custom **product.json** file and logo images into the **branding**/ directory.
3. Build the container image with Che-Theia and push the image to a container registry:

```
$ podman build -t username/che-theia-devspaces:next .
$ podman push username/che-theia-devspaces:next
```

3.9.1.3. Testing Che-Theia with custom branding

This section describes how to test a customized Che-Theia by opening a new workspace with custom branding.

Prerequisites

- Che-Theia container image built with custom branding definitions.

Procedure

To test a custom Che-Theia image, create a new **meta.yaml** file describing a custom **cheEditor**, and use it in a devfile for the testing workspace.

1. Clone the **che-plugin-registry** repository and check out the version to deploy. See, administration-guide:examples/snip_devspaces-clone-the-plug-in-registry-repository.adoc
2. Open the **che-editors.yaml** file.
3. Edit the entry where **id** equals **eclipse/che-theia/next** and replace the **image** value in the **containers** section to point to the custom Che-Theia container image.
4. Build the registry: administration-guide:examples/snip_devspaces-build-a-custom-plug-in-registry.adoc

5. Navigate to the **./dependencies/che-plugin-registry/v3/plugins/eclipse/che-theia/next** directory.

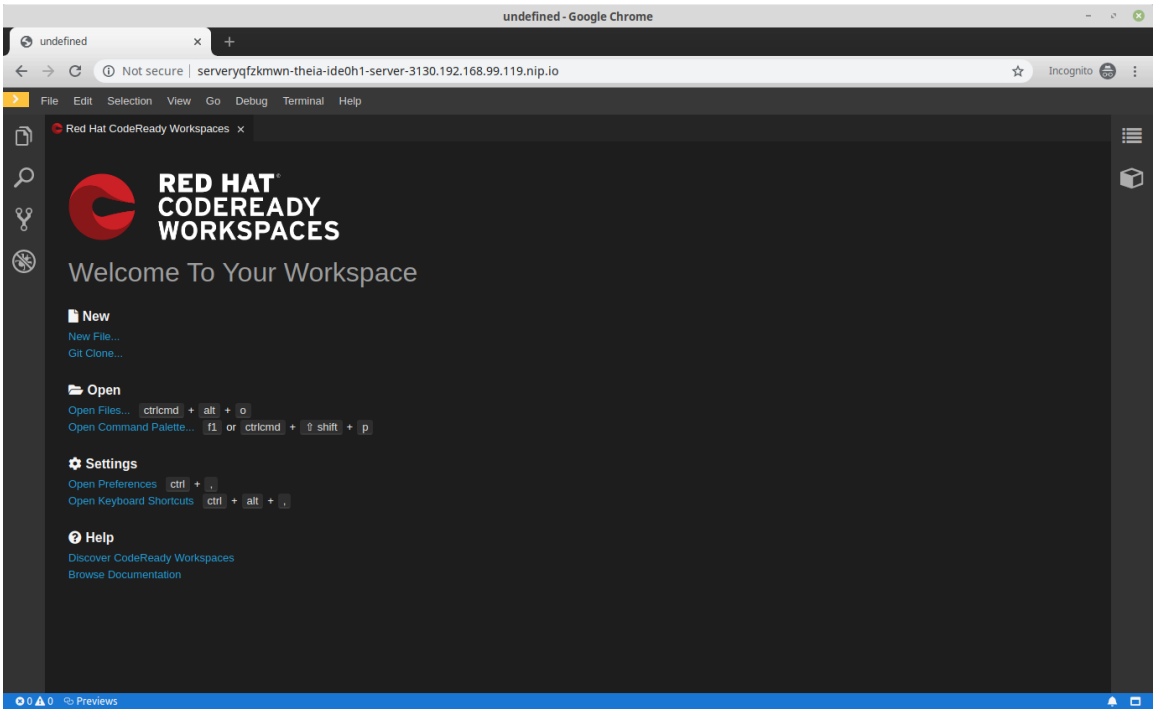
6. Publish the **meta.yaml** file in this directory to a publicly accessible location where it can be used as an HTTP resource.

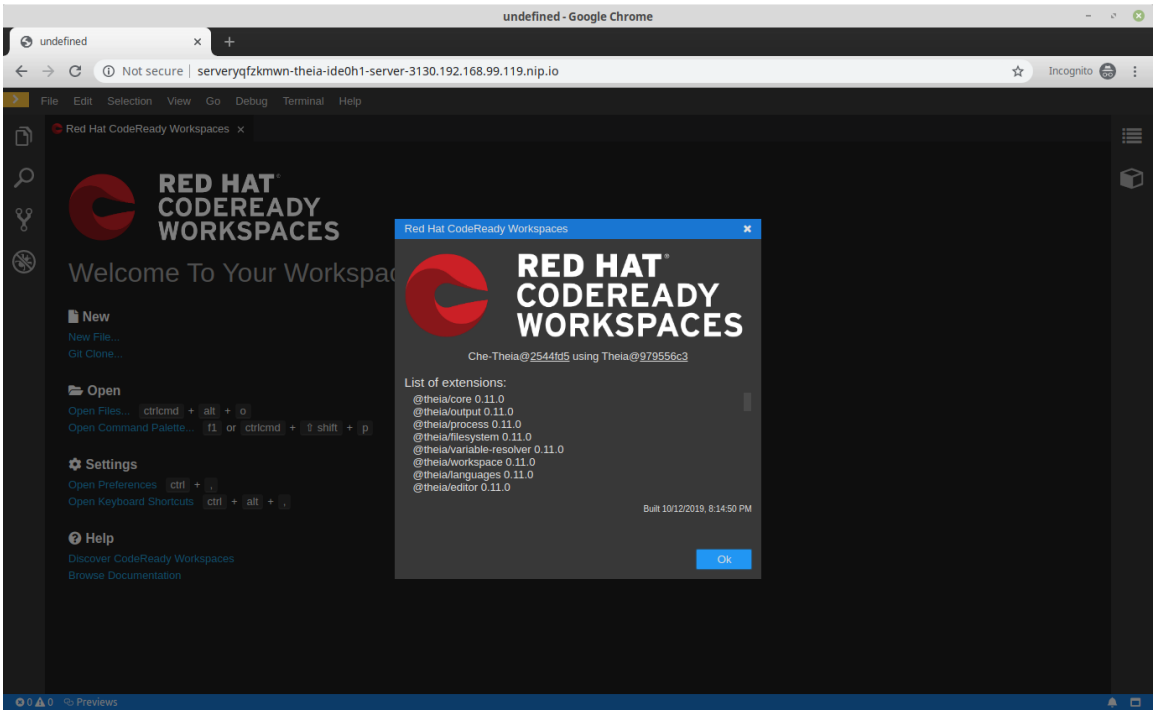
7. Create a workspace using the sample che-theia-branding-example devfile to apply the changes.
   Verify the **reference** field points to your published **meta.yaml** file:


```
metadata:
  name: che-theia-all
projects:
  - name: che-cheia-branding-example
    source:
      location: 'https://github.com/che-samples/che-theia-branding-example.git'
      type: git
      branch: master
components:
  - type: cheEditor
    reference: >-
      https://raw.githubusercontent.com/che-samples/che-theia-branding-example/master/che-
editor.meta.yaml
apiVersion: 1.0.0
```

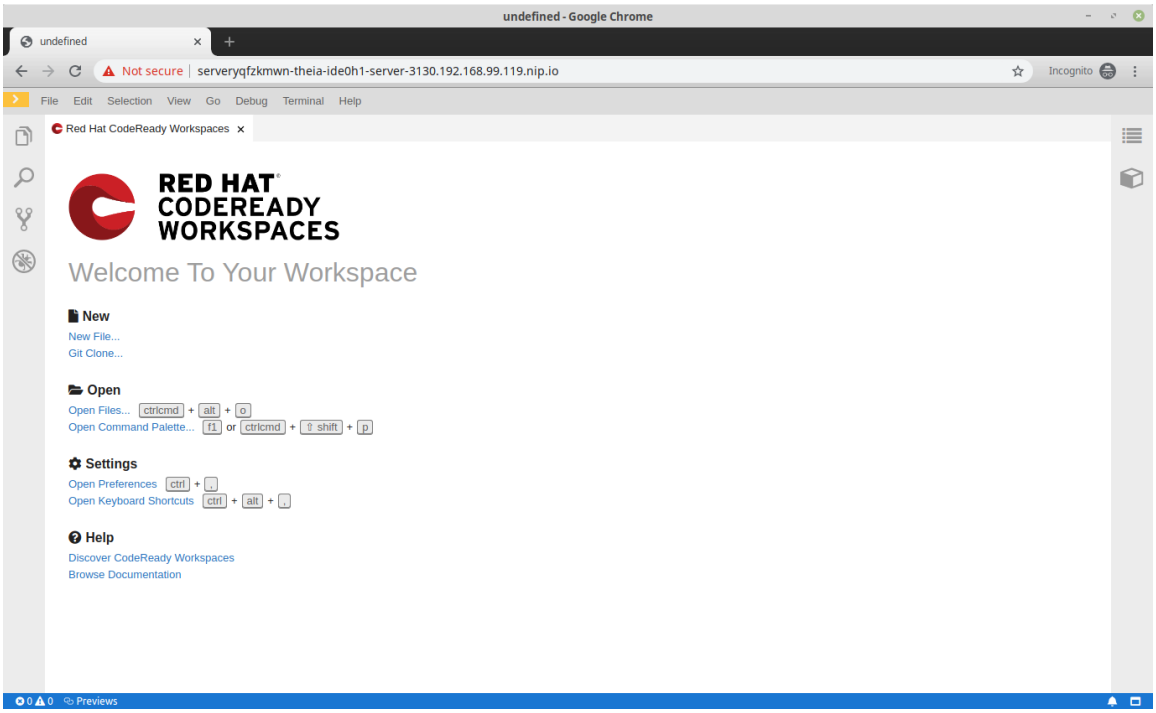
8. Run the workspace to see the changes:

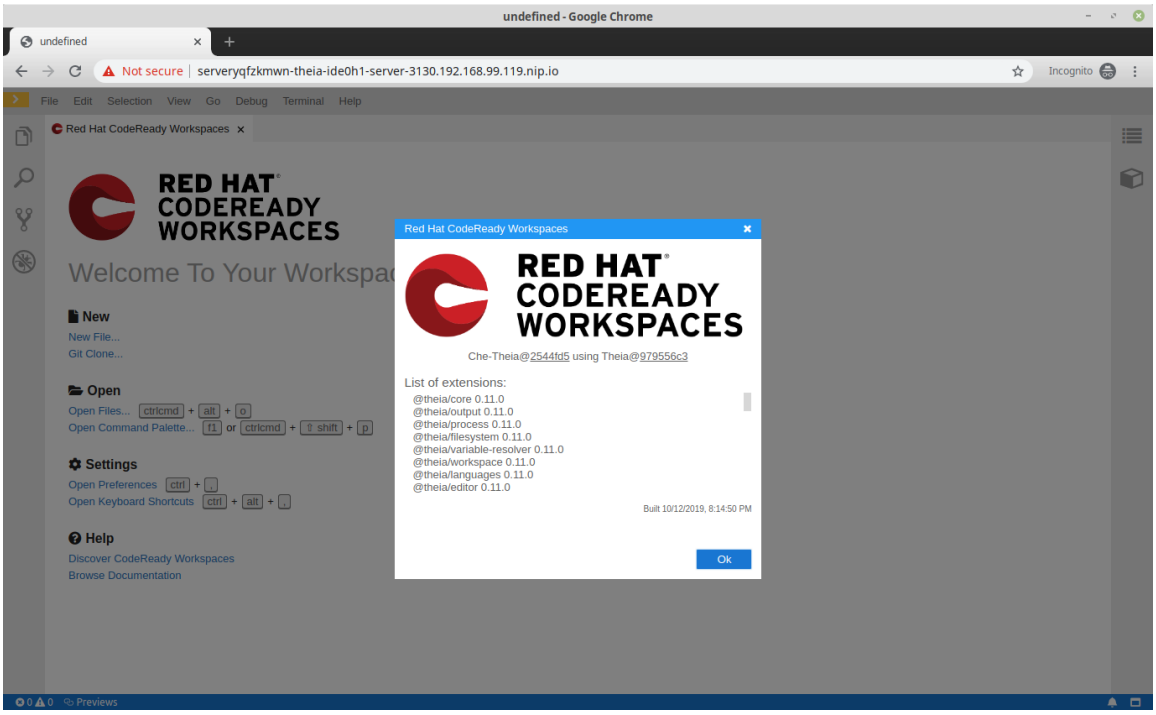
   - The dark theme of Che-Theia:





- The light theme of Che-Theia:





3.10. MANAGING IDENTITIES AND AUTHORIZATIONS

This section describes different aspects of managing identities and authorizations of Red Hat OpenShift Dev Spaces.

- Section 3.10.3, “Removing user data”

3.10.1. OAuth for GitHub, GitLab, or Bitbucket

To enable users to work with remote Git repositories:

- Section 3.10.1.1, “Configuring OAuth 2.0 for GitHub”
- Section 3.10.1.2, “Configuring OAuth 2.0 for GitLab”

- Section 3.10.1.3, “Configuring OAuth 1.0 for Bitbucket”

3.10.1.1. Configuring OAuth 2.0 for GitHub

To enable users to work with a remote Git repository that is hosted on GitHub:

1. Set up the GitHub OAuth App (OAuth 2.0).
2. Apply the GitHub OAuth App Secret.

3.10.1.1.1. Setting up the GitHub OAuth App

Set up a GitHub OAuth App using OAuth 2.0.

Prerequisites

- You are logged in to GitHub.
- **base64** is installed in the operating system you are using.

Procedure

1. Go to https://github.com/settings/applications/new.
2. Enter the following values:
   a. **Application name**: **OpenShift Dev Spaces**.
   b. **Homepage URL**: **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/**
   c. **Authorization callback URL**: **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/api/oauth/callback**
3. Click **Register application**.
4. Click **Generate new client secret**
5. Copy the **GitHub OAuth Client ID** and encode it to Base64 for use when applying the GitHub OAuth App Secret:

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

6. Copy the **GitHub OAuth Client Secret** and encode it to Base64 for use when applying the GitHub OAuth App Secret:

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

Additional resources

- GitHub Docs: Creating an OAuth App

3.10.1.1.2. Applying the GitHub OAuth App Secret

Prepare and apply the GitHub OAuth App Secret.

Prerequisites

- Setting up the GitHub OAuth App is completed.
- The Base64-encoded values, which were generated when setting up the GitHub OAuth App, are prepared:
  - **GitHub OAuth Client ID**
  - **GitHub OAuth Client Secret**
- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.

Procedure

1. Prepare the Secret:

```
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: github
type: Opaque
data:
  id: <Base64_GitHub_OAuth_Client_ID> (2)
  secret: <Base64_GitHub_OAuth_Client_Secret> (3)
```

(1) The OpenShift Dev Spaces namespace. The default is **openshift-devspaces**.

(2) The Base64-encoded **GitHub OAuth Client ID**.

(3) The Base64-encoded **GitHub OAuth Client Secret**

2. Apply the Secret:

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

3. Verify in the output that the Secret is created.

3.10.1.2. Configuring OAuth 2.0 for GitLab

To enable users to work with a remote Git repository that is hosted using a GitLab instance:

1. Set up the GitLab authorized application (OAuth 2.0).
2. Apply the GitLab authorized application Secret.

3.10.1.2.1. Setting up the GitLab authorized application

Set up a GitLab authorized application using OAuth 2.0.

Prerequisites

- You are logged in to GitLab.
- **base64** is installed in the operating system you are using.

Procedure

1. Click your avatar and go to **Edit profile → Applications**.
2. Enter **OpenShift Dev Spaces** as the **Name**.
3. Enter **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/api/oauth/callback** as the **Redirect URI**.
4. Check the **Confidential** and **Expire access tokens** checkboxes.
5. Under **Scopes**, check the **api**, **write_repository**, and **openid** checkboxes.
6. Click **Save application**.
7. Copy the **GitLab Application ID** and encode it to Base64 for use when applying the GitLab-authorized application Secret:

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

8. Copy the **GitLab Client Secret** and encode it to Base64 for use when applying the GitLab-authorized application Secret:

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

Additional resources

- GitLab Docs: Authorized applications

3.10.1.2.2. Applying the GitLab-authorized application Secret

Prepare and apply the GitLab-authorized application Secret.

Prerequisites

- Setting up the GitLab authorized application is completed.
- The Base64-encoded values, which were generated when setting up the GitLab authorized application, are prepared:
  - **GitLab Application ID**
  - **GitLab Client Secret**

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.

Procedure

1. Prepare the Secret:

```
kind: Secret
apiVersion: v1
metadata:
  name: gitlab-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: gitlab
    che.eclipse.org/scm-server-endpoint: <gitlab_server_url> (2)
type: Opaque
data:
  id: <Base64_GitLab_Application_ID> (3)
  secret: <Base64_GitLab_Client_Secret> (4)
```

(1) The OpenShift Dev Spaces namespace. The default is **openshift-devspaces**.

(2) The **GitLab server URL** Use **https://gitlab.com** for the **SAAS** version.

(3) The Base64-encoded **GitLab Application ID**.

(4) The Base64-encoded **GitLab Client Secret**.

2. Apply the Secret:

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

3. Verify in the output that the Secret is created.

3.10.1.3. Configuring OAuth 1.0 for Bitbucket

To enable users to work with a remote Git repository that is hosted on a Bitbucket server:

1. Set up the Bitbucket application link (OAuth 1.0).
2. Apply the Bitbucket application link Secret.

3.10.1.3.1. Setting up the Bitbucket application link

Set up a Bitbucket application link using OAuth 1.0.

Prerequisites

- You are logged in to Bitbucket.

- **openssl** is installed in the operating system you are using.
- **base64** is installed in the operating system you are using.

Procedure

1. On a command line, run the commands to create the necessary files for the next steps and for use when applying the Bitbucket application link Secret:

```
$ openssl genrsa -out private.pem 2048 && \
openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in private.pem -out
privatepkcs8.pem && \
cat privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' | sed 's/-----END PRIVATE
KEY-----//g' | tr -d '\n' | base64 | tr -d '\n' > privatepkcs8-stripped.pem && \
openssl rsa -in private.pem -pubout > public.pub && \
cat public.pub | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY-----//g'
| tr -d '\n' > public-stripped.pub && \
openssl rand -base64 24 > bitbucket-consumer-key && \
openssl rand -base64 24 > bitbucket-shared-secret
```

2. Go to **Administration → Application Links**.
3. Enter **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/** into the URL field and click **Create new link**.
4. Under **The supplied Application URL has redirected once**, check the **Use this URL** checkbox and click **Continue**.
5. Enter **OpenShift Dev Spaces** as the **Application Name**.
6. Select **Generic Application** as the **Application Type**.
7. Enter **OpenShift Dev Spaces** as the **Service Provider Name**.
8. Paste the content of the **bitbucket-consumer-key** file as the **Consumer key**.
9. Paste the content of the **bitbucket-shared-secret** file as the **Shared secret**.
10. Enter ***<bitbucket_server_url>*/plugins/servlet/oauth/request-token** as the **Request Token URL**.
11. Enter ***<bitbucket_server_url>*/plugins/servlet/oauth/access-token** as the **Access token URL**.
12. Enter ***<bitbucket_server_url>*/plugins/servlet/oauth/authorize** as the **Authorize URL**.
13. Check the **Create incoming link** checkbox and click **Continue**.
14. Paste the content of the **bitbucket_consumer_key** file as the **Consumer Key**.
15. Enter **OpenShift Dev Spaces** as the **Consumer name**.
16. Paste the content of the **public-stripped.pub** file as the **Public Key** and click **Continue**.

Additional resources

- Atlassian Documentation: Link to other applications

3.10.1.3.2. Applying the Bitbucket application link Secret

Prepare and apply the Bitbucket application link Secret.

Prerequisites

- Setting up the Bitbucket application link is completed.
- The following Base64-encoded files, which were created when setting up the Bitbucket application link, are prepared:
  - **privatepkcs8-stripped.pem**
  - **bitbucket_consumer_key**
  - **bitbucket-shared-secret**
- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.

Procedure

1. Prepare the Secret:

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/component: oauth-scm-configuration
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: <bitbucket_server_url> (2)
type: Opaque
data:
  private.key: <Base64_content_of_privatepkcs8-stripped.pem> (3)
  consumer.key: <Base64_content_of_bitbucket_server_consumer_key> (4)
  shared_secret: <Base64_content_of_bitbucket-shared-secret> (5)
```

(1) The OpenShift Dev Spaces namespace. The default is **openshift-devspaces**.

(2) The Bitbucket server URL.

(3) The Base64-encoded content of the **privatepkcs8-stripped.pem** file.

(4) The Base64-encoded content of the **bitbucket_consumer_key** file.

(5) The Base64-encoded content of the **bitbucket-shared-secret** file.

2. Apply the Secret:

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

3. Verify in the output that the Secret is created.

3.10.2. Configuring the administrative user

To execute actions that require administrative privileges on OpenShift Dev Spaces server, such as deleting user data, activate a user with administrative privileges. The default installation enables the administrative privileges for the **admin** user, regardless of its existence on OpenShift.

Procedure

- Configure the **CheCluster** Custom Resource to set the *<admin>* user with administrative privileges. See Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource" .

```
spec:
 server:
   customCheProperties:
     CHE_SYSTEM_ADMIN__NAME: '<admin>'
```

Additional resources

- Section 3.1.1, "Using dsc to configure the **CheCluster** Custom Resource during installation"
- Section 3.1.2, "Using the CLI to configure the CheCluster Custom Resource"

3.10.3. Removing user data

3.10.3.1. Removing user data according to GDPR

You can remove the OpenShift Dev Spaces user's data using the OpenShift Dev Spaces API. Following this procedure makes the service compliant to EU General Data Protection Regulation (GDPR) that enforces the right for individuals to have personal data erased.

Prerequisites

- An active session with administrative permissions to OpenShift Dev Spaces. See Section 3.10.2, "Configuring the administrative user".
- An active **oc** session with administrative permissions to the OpenShift cluster. See Getting started with the OpenShift CLI.

Procedure

1. Get the *<username>* user *<id>* **id**: navigate to https://*<devspaces-*<openshift_deployment_name>.<domain_name>>/swagger/#/user/find_1, click **Try it out**, set **name**: *<username>*, and click **Execute**. Scroll down the **Response body** to find the **id** value.

2. Remove the *<id>* user data that OpenShift Dev Spaces server manages, such as user preferences: navigate to

https://<*devspaces*-<openshift_deployment_name>.<domain_name>>/swagger/#/user/remove, click **Try it out**, set **id**: <*id*>, and click **Execute**. Expect a **204** response code:

3. Delete the user project to remove all OpenShift resources bound to the user, such as workspaces, secrets, and configmaps.

```
$ oc delete namespace <username>-devspaces
```

Additional resources

- Chapter 4, *Managing OpenShift Dev Spaces server workloads using the OpenShift Dev Spaces server API*.
- Section 3.2.1, “Configuring a user project name for automatic provisioning” .
- To remove the data of all users, see Chapter 6, *Uninstalling OpenShift Dev Spaces*.

CHAPTER 4. MANAGING OPENSHIFT DEV SPACES SERVER WORKLOADS USING THE OPENSHIFT DEV SPACES SERVER API

To manage OpenShift Dev Spaces server workloads, use the Swagger web user interface to navigate OpenShift Dev Spaces server API.

Procedure

- Navigate to the Swagger API web user interface: **https://devspaces-*<openshift_deployment_name>*.*<domain_name>*/swagger**.

Additional resources

- Swagger

CHAPTER 5. UPGRADING OPENSHIFT DEV SPACES

This chapter describes how to upgrade from CodeReady Workspaces 2.15 to OpenShift Dev Spaces 3.0.

5.1. UPGRADING THE DSC MANAGEMENT TOOL

This section describes how to upgrade the **dsc** management tool.

Procedure

- Section 2.1, “Install the dsc management tool” .

5.2. UPGRADING CODEREADY WORKSPACES 2.15 ON RED HAT OPENSHIFT

The workspace engine and authentication system used in CodeReady Workspaces 2.15 and earlier versions are deprecated. Due to this deprecation, upgrading CodeReady Workspaces 2.15 involves running upgrade scripts.

5.2.1. Manually upgrading CodeReady Workspaces 2.15 to OpenShift Dev Spaces 3.0.1 on Red Hat OpenShift

You can manually upgrade CodeReady Workspaces 2.15 to OpenShift Dev Spaces 3.0.1 on Red Hat OpenShift.

Prerequisites

- OpenShift Container Platform 4.10 or OpenShift Dedicated 4.10 or Red Hat OpenShift Service on AWS (ROSA) 4.10.
- An instance of CodeReady Workspaces deployed on one of the Section 1.1, “Supported platforms”. The instance uses the default internal PostgreSQL database and has OAuth enabled on Red Hat OpenShift. See Red Hat CodeReady Workspaces 2.15 - Configuring OpenShift OAuth.
- The following command line tools are available:
  - **oc**
  - **curl**
  - **jq**
- The host running the upgrade commands is running on Linux.
- Optional:
  a. All changes from all workspaces have been committed and pushed to their Git remotes.
  b. All workspaces have been stopped to avoid UX degradation.
  c. The CodeReady Workspaces data have been backed up. See Red Hat CodeReady Workspaces 2.15 - Backup and recovery.

Procedure

1. Download 1-prepare.sh.
   1-prepare.sh shuts down CodeReady Workspaces and RH-SSO, fetches the existing users' data, and dumps the CodeReady Workspaces database.

2. Download 2-migrate.sh.
   2-migrate.sh fetches CodeReady Workspaces RH-SSO and database data, and repopulates the database with updated data.

3. Download 3-subscribe.sh.
   3-subscribe.sh deletes CodeReady Workspaces Operator and RH-SSO resources, updates the CheCluster CR, and creates a new OpenShift Dev Spaces Operator subscription.

4. Download 4-wait.sh.
   4-wait.sh waits until OpenShift Dev Spaces is ready, which can take more than 5 minutes.

5. Set the environment variables to use in the upgrade scripts:

```
export INSTALLATION_NAMESPACE=openshift-workspaces (1)
export PRODUCT_ID=red-hat-openshift-devspaces
export PRODUCT_DEPLOYMENT_NAME=devspaces
export PRODUCT_OPERATOR_NAME=devspaces-operator
export PRODUCT_OLM_STABLE_CHANNEL=stable
export PRODUCT_OLM_CATALOG_SOURCE=redhat-operators
export PRODUCT_OLM_PACKAGE=devspaces
export PRODUCT_OLM_STARTING_CSV=devspacesoperator.v3.0.1
export PRE_MIGRATION_PRODUCT_OPERATOR_NAMESPACE=openshift-workspaces
(2)
export PRE_MIGRATION_PRODUCT_SHORT_ID=codeready
export PRE_MIGRATION_PRODUCT_DEPLOYMENT_NAME=codeready
export PRE_MIGRATION_PRODUCT_OPERATOR_NAME=codeready-operator
export PRE_MIGRATION_PRODUCT_CHE_CLUSTER_CR_NAME=codeready-workspaces
export
PRE_MIGRATION_PRODUCT_IDENTITY_PROVIDER_DEPLOYMENT_NAME=keycloak
export PRE_MIGRATION_PRODUCT_SUBSCRIPTION_NAME=codeready-workspaces
```

   (1) **openshift-workspaces** or another project where CodeReady Workspaces was previously installed.

   (2) **openshift-workspaces** or another project where CodeReady Workspaces was previously installed.

6. Run the upgrade scripts:

```
$ chmod +x ./1-prepare.sh ./2-migrate.sh ./3-subscribe.sh ./4-wait.sh; \
  ./1-prepare.sh && ./2-migrate.sh && ./3-subscribe.sh && ./4-wait.sh
```

Verification

- In the OpenShift Dev Spaces dashboard, go to **About → Server Version** to verify that it is **3.0**.

5.2.2. Rolling the upgrade back to CodeReady Workspaces 2.15 on Red Hat OpenShift

If upgrading CodeReady Workspaces 2.15 to OpenShift Dev Spaces 3.0.1 on Red Hat OpenShift fails, you can run a rollback script to restore CodeReady Workspaces 2.15.

Prerequisites

- OpenShift Container Platform 4.10 or OpenShift Dedicated 4.10 or Red Hat OpenShift Service on AWS (ROSA) 4.10.

Procedure

1. Download the rollback.sh script.
2. Set the environment variables to use in the **rollback.sh** script:

```
export INSTALLATION_NAMESPACE=openshift-workspaces (1)
export PRODUCT_ID=red-hat-openshift-devspaces
export PRODUCT_SHORT_ID=devspaces
export PRODUCT_DEPLOYMENT_NAME=devspaces
export PRE_MIGRATION_PRODUCT_OPERATOR_NAMESPACE=openshift-workspaces
(2)
export PRE_MIGRATION_PRODUCT_DEPLOYMENT_NAME=codeready
export PRE_MIGRATION_PRODUCT_SUBSCRIPTION_NAME=codeready-workspaces
export PRE_MIGRATION_PRODUCT_CHE_CLUSTER_CR_NAME=codeready-workspaces
export PRE_MIGRATION_PRODUCT_OPERATOR_NAME=codeready-operator
export PRE_MIGRATION_PRODUCT_OLM_PACKAGE=codeready-workspaces
export PRE_MIGRATION_PRODUCT_OLM_CHANNEL=latest
export PRE_MIGRATION_PRODUCT_OLM_CATALOG_SOURCE=redhat-operators
export PRE_MIGRATION_PRODUCT_OLM_STARTING_CSV=crwoperator.v2.15.4
```

(1) **openshift-workspaces** or another project where CodeReady Workspaces was previously installed.

(2) **openshift-workspaces** or another project where CodeReady Workspaces was previously installed.

3. Run the **rollback.sh** script.

```
$ chmod +x ./rollback.sh; ./rollback.sh
```

Verification

- In the CodeReady Workspaces dashboard, go to **About → Server Version** to verify that it is 2.15.

5.3. SPECIFYING THE UPDATE APPROVAL STRATEGY FOR THE RED HAT OPENSHIFT DEV SPACES OPERATOR

The Red Hat OpenShift Dev Spaces Operator supports two upgrade strategies:

Automatic

The Operator installs new updates when they become available.

Manual

New updates need to be manually approved before installation begins.

You can specify the update approval strategy for the Red Hat OpenShift Dev Spaces Operator by using the OpenShift web console.

Prerequisites

- An OpenShift web console session by a cluster administrator. See Accessing the web console.
- An instance of OpenShift Dev Spaces that was installed by using Red Hat Ecosystem Catalog.

Procedure

1. In the OpenShift web console, navigate to **Operators → Installed Operators**.
2. Click **Red Hat OpenShift Dev Spaces**in the list of installed Operators.
3. Navigate to the **Subscription** tab.
4. Configure the **Update approval** strategy to **Automatic** or **Manual**.

Additional resources

- Changing the update channel for an Operator

5.4. UPGRADING OPENSHIFT DEV SPACES USING THE OPENSHIFT WEB CONSOLE

You can manually approve an upgrade from an earlier minor version using the Red Hat OpenShift Dev Spaces Operator from the Red Hat Ecosystem Catalog in the OpenShift web console.

Prerequisites

- An OpenShift web console session by a cluster administrator. See Accessing the web console.
- An instance of OpenShift Dev Spaces that was installed by using the Red Hat Ecosystem Catalog.
- The approval strategy in the subscription is **Manual**. See Section 5.3, "Specifying the update approval strategy for the Red Hat OpenShift Dev Spaces Operator".

Procedure

- Manually approve the pending Red Hat OpenShift Dev Spaces Operator upgrade. See Manually approving a pending Operator upgrade.

Verification steps

1. Navigate to the OpenShift Dev Spaces instance.
2. The 3.0 version number is visible at the bottom of the page.

Additional resources

- Manually approving a pending Operator upgrade

5.5. REPAIRING THE DEVWORKSPACE OPERATOR ON OPENSHIFT

Under certain conditions, such as OLM restart or cluster upgrade, the Dev Spaces Operator for OpenShift Dev Spaces might automatically install the DevWorkspace Operator even when it is already present on the cluster. In that case, you can repair the DevWorkspace Operator on OpenShift as follows:

Prerequisites

- An active **oc** session as a cluster administrator to the destination OpenShift cluster. See Getting started with the CLI.
- On the **Installed Operators** page of the OpenShift web console, you see multiple entries for the DevWorkspace Operator or one entry that is stuck in a loop of **Replacing** and **Pending**.

Procedure

1. Delete the **devworkspace-controller** namespace that contains the failing pod.
2. Update **DevWorkspace** and **DevWorkspaceTemplate** Custom Resource Definitions (CRD) by setting the conversion strategy to **None** and removing the entire **webhook** section:

```
spec:
  ...
  conversion:
    strategy: None
status:
...
```

TIP

You can find and edit the **DevWorkspace** and **DevWorkspaceTemplate** CRDs in the **Administrator** perspective of the OpenShift web console by searching for **DevWorkspace** in **Administration → CustomResourceDefinitions**.

NOTE

The **DevWorkspaceOperatorConfig** and **DevWorkspaceRouting** CRDs have the conversion strategy set to **None** by default.

3. Remove the DevWorkspace Operator subscription:

```
$ oc delete sub devworkspace-operator \
-n openshift-operators 1
```

1 **openshift-operators** or an OpenShift project where the DevWorkspace Operator is installed.

4. Get the DevWorkspace Operator CSVs in the *<devworkspace-operator.vX.Y.Z>* format:

```
$ oc get csv | grep devworkspace
```

5. Remove each DevWorkspace Operator CSV:

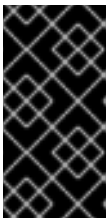
```
$ oc delete csv <devworkspace-operator.vX.Y.Z> \
-n openshift-operators (1)
```

(1) **openshift-operators** or an OpenShift project where the DevWorkspace Operator is installed.

6. Re-create the DevWorkspace Operator subscription:

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: devworkspace-operator
  namespace: openshift-operators
spec:
  channel: fast
  name: devworkspace-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic (1)
  startingCSV: devworkspace-operator.v0.15.2
EOF
```

(1) **Automatic** or **Manual**.



IMPORTANT

For **installPlanApproval: Manual**, in the **Administrator** perspective of the OpenShift web console, go to **Operators → Installed Operators** and select the following for the **DevWorkspace Operator**: **Upgrade available → Preview InstallPlan → Approve**.

7. In the **Administrator** perspective of the OpenShift web console, go to **Operators → Installed Operators** and verify the **Succeeded** status of the **DevWorkspace Operator**.

CHAPTER 6. UNINSTALLING OPENSHIFT DEV SPACES

WARNING

Uninstalling OpenShift Dev Spaces removes all OpenShift Dev Spaces-related user data!

To uninstall an instance of Red Hat OpenShift Dev Spaces 3.0:

Prerequisites

- An active **oc** session with administrative permissions to the destination OpenShift cluster. See Getting started with the CLI.
- **dsc**. See: Section 2.1, “Install the dsc management tool” .

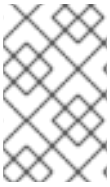
Procedure

1. Obtain the name of the OpenShift Dev Spaces project (default: **openshift-devspaces**):

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. Remove the OpenShift Dev Spaces instance from the *<openshift-devspaces>* project:

```
$ dsc server:delete -n <openshift-devspaces>
```



NOTE

If OpenShift Dev Spaces was installed from the OpenShift web console, **dsc** will not uninstall the DevWorkspace operator. To uninstall the DevWorkspace Operator, see Deleting the DevWorkspace Operator dependency.