Red Hat CodeReady Workspaces 1.0

Administration Guide

Installing and administering Red Hat CodeReady Workspaces 1.0

Last Updated: 2019-04-26

Red Hat CodeReady Workspaces 1.0 Administration Guide

Installing and administering Red Hat CodeReady Workspaces 1.0

Supriya Takkhi
sbharadw@redhat.com

Robert Kratky
rkratky@redhat.com

Legal Notice

Abstract

Information for administrators installing and operating Red Hat CodeReady Workspaces.

Table of Contents

CHAPTER 1. UNDERSTANDING RED HAT CODEREADY WORKSPACES

Red Hat CodeReady Workspaces is a developer workspace server and cloud IDE. Workspaces are defined as project code files and all of their dependencies neccessary to edit, build, run, and debug them. Each workspace has its own private IDE hosted within it. The IDE is accessible through a browser. The browser downloads the IDE as a single-page web application.

Red Hat CodeReady Workspaces provides:

- Workspaces that include runtimes and IDEs
- RESTful workspace server
- A browser-based IDE
- Plugins for languages, framework, and tools
- An SDK for creating plugins and assemblies

Additional resources

See Red Hat CodeReady Workspaces CodeReady Workspaces 1.0.2 Release Notes and Known Issues for more details about the current version.

CHAPTER 2. INSTALLING CODEREADY WORKSPACES

This section describes how to obtain installation files for Red Hat CodeReady Workspaces and how to use them to deploy the product on an instance of OpenShift (such as Red Hat OpenShift Container Platform).

2.1. DOWNLOADING THE CODEREADY WORKSPACES DEPLOYMENT SCRIPT

This procedure describes how to obtain and unpack the archive with the CodeReady Workspaces deployment shell script.

The CodeReady Workspaces deployment script uses the OpenShift Operator to deploy Red Hat Single Sign-On, the PostgreSQL database, and the CodeReady Workspaces server container images on an instance of Red Hat OpenShift Container Platform. The images are available in the Red Hat Container Catalogue.

Prerequisites

- Registered for the free Red Hat Developer Program.
- Logged in to the Red Hat Developer Portal.

Procedure

1. Change to a temporary directory. Create it if necessary. For example:

   ```
   $ mkdir ~/tmp
   $ cd ~/tmp
   ```

2. Download the archive with the deployment script and the **`config.yaml`** file using the browser with which you logged into the Red Hat Developer Portal: codeready-workspaces-1.0.2.GA-operator-installer.tar.gz.

3. Unpack the downloaded archive and change to the created directory:

   ```
   $ tar xvf codeready-workspaces-1.0.2.GA-operator-installer.tar.gz \
     && cd codeready-workspaces-operator-installer/
   ```

Next steps

Continue by configuring and running the deployment script. See Section 2.2, “Running the CodeReady Workspaces deployment script”.

2.2. RUNNING THE CODEREADY WORKSPACES DEPLOYMENT SCRIPT

The CodeReady Workspaces deployment script uses command-line arguments and the **`config.yaml`** file to populate a set of configuration environment variables for the OpenShift Operator used for the actual deployment.

Prerequisites

- Downloaded and unpacked the deployment script and the configuration file. See Section 2.1, “Downloading the CodeReady Workspaces deployment script”.
- A running instance of Red Hat OpenShift Container Platform 3.11 or OpenShift Dedicated 3.11. To install OpenShift Container Platform, see the Getting Started with OpenShift Container Platform guide. To install OpenShift Dedicated 3.11, see OKD documentation.
- The OpenShift command-line client tool, **`oc`**, is in the path.
- The user is logged in to the OpenShift instance (using, for example, **`oc login`**).
- CodeReady Workspaces is supported for use with Google Chrome 70.0.3538.110 (Official Build) (64bit).

2.2.1. Deploying CodeReady Workspaces with default settings

1. Run the following command:

   ```
   $ ./deploy.sh --deploy \
   --server-image=registry.access.redhat.com/codeready-
   workspaces/server:1.0 \
   --operator-image=registry.access.redhat.com/codeready-
   workspaces/server-operator:1.0
   ```

   NOTE

   Run the **`./deploy.sh --help`** command to get a list of all available arguments. For a description of all the options, see Section 2.2.6, “Configuring environment variables”.

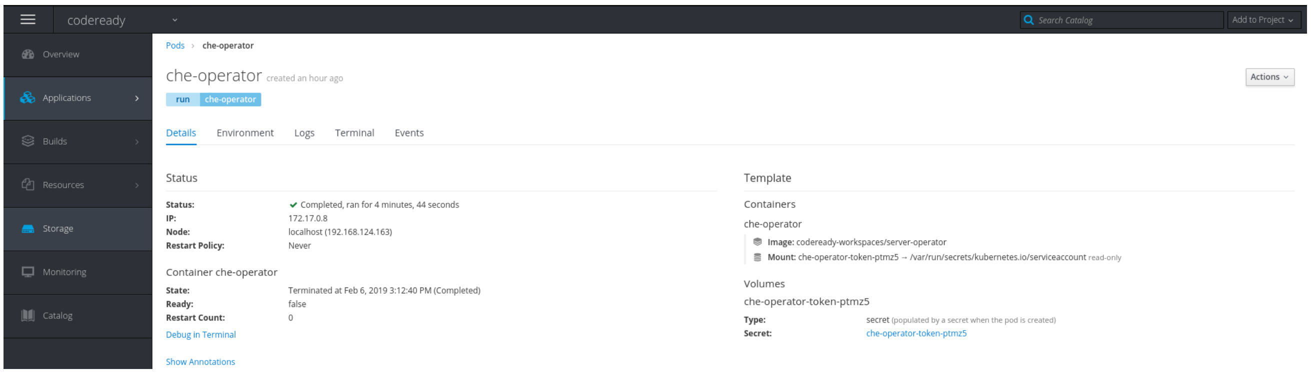
   The following message indicates that CodeReady Workspaces is getting installed:

   ```
   [INFO]: Welcome to CodeReady Workspaces Installer
   [INFO]: Found oc client in PATH
   [INFO]: Checking if you are currently logged in...
   [INFO]: Active session found. Your current context is:
   myproject/192-168-42-106:8443/developer
   [INFO]: Creating namespace "codeready"
   [INFO]: Namespace "codeready" successfully created
   [INFO]: Creating installer service account
   serviceaccount "codeready-apb" created
   ```

   The **CodeReady Workspaces successfully deployed and available at *<URL>*** message confirms that the deployment is successful.

2. Open the OpenShift console in a web browser (**`<OpenShift-IP>/console`**).
3. In the **My Projects** pane, click **codeready**.
4. Click **Applications** > **Pods**.
5. Click **che-operator** to see the pod running.

Figure 2.1. Pod for codeready shown running



che-operator created an hour ago

Status

Status: ✓ Completed, ran for 4 minutes, 44 seconds
IP: 172.17.0.8
Node: localhost (192.168.124.163)
Restart Policy: Never

Container che-operator

State: Terminated at Feb 6, 2019 3:12:40 PM (Completed)
Ready: false
Restart Count: 0

Template

Containers

che-operator

Image: codeready-workspaces/server-operator
Mount: che-operator-token-ptmz5 → /var/run/secrets/kubernetes.io/serviceaccount read-only

Volumes

che-operator-token-ptmz5

Type: secret (populated by a secret when the pod is created)
Secret: che-operator-token-ptmz5

2.2.2. Deploying CodeReady Workspaces with a self-signed certificate and OpenShift oAuth

To deploy CodeReady Workspaces with a self-signed certificate, specify its location for the **`deploy.sh`** script using the **`--cert`** option. Then build a custom stack container image with the certificate to launch workspaces.

1. Run the deployment script with the following options:

```
$ ./deploy.sh --deploy --enable-oauth \
  --cert=<path-to-the-certificate> \
  --server-image=registry.access.redhat.com/codeready-
workspaces/server:1.0 \
  --operator-image=registry.access.redhat.com/codeready-
workspaces/server-operator:1.0
```

   Example:

```
$ ./deploy.sh --deploy --enable-oauth \
  --cert=/var/lib/origin/openshift.local.config/master/ca.crt \
  --server-image=registry.access.redhat.com/codeready-
workspaces/server:1.0 \
  --operator-image=registry.access.redhat.com/codeready-
workspaces/server-operator:1.0
```

2.2.3. Deploying CodeReady Workspaces with a public certificate

When deploying CodeReady Workspaces to a cluster configured with public certificates, the deployment script requires no additional parameters.

Run the deploy script as:

```
$ ./deploy.sh --deploy \
  --server-image=registry.access.redhat.com/codeready-
workspaces/server:1.0 \
  --operator-image=registry.access.redhat.com/codeready-workspaces/server-
operator:1.0
```

2.2.4. Deploying CodeReady Workspaces with external Red Hat SSO and Keycloak

To deploy with an external Red Hat SSO and enable a Keycloak instance, take the following steps:

1. Update the following values in the **`config.yaml`** file:

```
CHE_EXTERNAL_KEYCLOAK:    "true"      (1)
CHE_KEYCLOAK_AUTH__SERVER__URL: "<keycloak_url>"   (2)
CHE_KEYCLOAK_ADMIN_USERNAME:  "<keycloak_admin_username>" (3)
CHE_KEYCLOAK_ADMIN_PASSWORD:  "<keycloak_admin_password>" (4)
CHE_KEYCLOAK_REALM:    "<realm>"     (5)
CHE_KEYCLOAK_CLIENT_ID:   "<client_id>"    (6)
```

(1) Connect to an existing external Keycloak instance and skip deploying a dedicated Keycloak instance. The default value is **`false`**.

(2) Keycloak/Red Hat SSO

(3) Keycloak Administrator username. Not required when connecting to an existing realm.

(4) Keycloak Administrator password. Auto-generated if empty.

(5) Red Hat SSO realm.

(6) Red Hat SSO client ID.

2. Run the deploy script:

```
$ ./deploy.sh --deploy \
  --server-image=registry.access.redhat.com/codeready-
workspaces/server:1.0 \
  --operator-image=registry.access.redhat.com/codeready-
workspaces/server-operator:1.0
```

2.2.5. Deploying CodeReady Workspaces with external Red Hat SSO and PostgreSQL

To deploy with an external Red Hat SSO and PostgreSQL database, take the following steps:

1. Update the following values in the **`config.yaml`** file:

```
CHE_EXTERNAL_KEYCLOAK:    "true"      (1)
CHE_KEYCLOAK_AUTH__SERVER__URL: "<keycloak_url>"   (2)
CHE_KEYCLOAK_ADMIN_USERNAME:  "<keycloak_admin_username>" (3)
CHE_KEYCLOAK_ADMIN_PASSWORD:  "<keycloak_admin_password>" (4)
CHE_KEYCLOAK_REALM:    "<realm>"     (5)
CHE_KEYCLOAK_CLIENT_ID:   "<client_id>"    (6)
```

(1) Connect to an existing external Keycloak instance and skip deploying a dedicated Keycloak instance. The default value is **`false`**.

(2) Keycloak/Red Hat SSO

(3) Keycloak Administrator username. Not required when connecting to an existing realm.

(4) Keycloak Administrator password. Auto-generated if empty.

5. Red Hat SSO realm.
6. Red Hat SSO client ID.

2. Update the following values in the **config.yaml** file:

```
CHE_EXTERNAL_DB:    "true"     (1)
CHE_DB_HOSTNAME:    "<postgres_service_ip>"   (2)
CHE_DB_PORT:    "5432"     (3)
CHE_DB_DATABASE:    "<database_name>"   (4)
CHE_JDBC_USERNAME:    "<user_name>"   (5)
CHE_JDBC_PASSWORD:    "<password>"   (6)
CHE_DB_ADMIN_PASSWORD:    "<password>"   (7)
```

1. Use an existing external Postgres database. The default value is **false** which means that a new instance of Postgres will be started. When set to **true**, provide the connection details and ensure that the database user is a **SUPERUSER**.
2. Database hostname from **Applications** → **Services** → **Cluster IP**.
3. Database port.
4. Database name.
5. Database username.
6. Database password.
7. Your database administrator password.

3. Run the deploy script:

```
$ ./deploy.sh --deploy \
--server-image=registry.access.redhat.com/codeready-
workspaces/server:1.0 \
--operator-image=registry.access.redhat.com/codeready-
workspaces/server-operator:1.0
```

2.2.6. Configuring environment variables

The **config.yaml** file contains default values for the installation parameters. Those parameters that take environment variables as values can be overridden from a command line. Not all installation parameters are available as flags.

Before running the deployment script in a fast mode, review the **config.yaml** file. Run the **./deploy.sh --help** command to get a list of all available arguments.

The following is an annotated example of the **config.yaml** file with all available parameters:

General Che settings:

```
CHE_FLAVOR:      "codeready" (1)
```

```
CHE_IMAGE:      ""  (2)
CHE_TLS_SUPPORT:     ""  (3)
CHE_INFRA_KUBERNETES_PVC_STRATEGY:  ""  (4)
CHE_INFRA_KUBERNETES_PVC_QUANTITY:  ""  (5)
CHE_SELF_SIGNED_CERT:     ""  (6)
CHE_OPENSHIFT_OAUTH:     ""  (7)
CHE_OPENSHIFT_API_URL:     ""  (8)
CHE_WORKSPACE_PLUGIN_REGISTRY_URL:  ""  (9)
CHE_UPDATE_CHE_ADMIN_PASSWORD:    ""  (10)
```

1. Che flavor. Upstream **`che`** or Red Hat **`codeready`**. Defaults to **`che`**.
2. Docker image for Che server. Defaults to **`eclipse/che-server:latest`**. Keep blank unless you need to deploy your custom image.
3. TLS support in Che. Defaults to **`false`**.
4. PVC strategy for Che workspaces. Defaults to 'common' where all workspaces use one shared PVC. A 'unique' strategy implies that each workspace gets own PVC.
5. Workspace PVC claim. Defaults to **`1Gi`**. It is recommended to increase it when using shared PVC.
6. cat root ca.crt | base64 -w 0
7. Enable login with OpenShift in Codeready Workspaces. OpenShift only. Defaults to **`false`**.
8. OpenShift API endpoint URL. Required only when **`OPENSHIFT_OAUTH`** is **`true`**. Auto detected.
9. Plugin registry URL. Defaults to `https://che-plugin-registry.openshift.io`.
10. Asks for password update at first login as Che administrator user. Defaults to **`true`**, that is, you will be asked to update password.

Che Proxy settings:

```
CHE_WORKSPACE_MASTER_PROXY_JAVA_OPTS:   ""  (1)
CHE_WORKSPACE_PROXY_JAVA_OPTS:    ""  (2)
CHE_WORKSPACE_HTTP__PROXY:    ""  (3)
CHE_WORKSPACE_HTTPS__PROXY:    ""  (4)
CHE_WORKSPACE_NO__PROXY:    ""  (5)
```

1. Proxy settings for workspace master, for example: **`Dhttp.proxyHost=host`** -Dhttp.proxyPort=8080 -Dhttps.proxyHost=host -Dhttps.proxyPort=8080 -Dhttp.nonProxyHosts='localhost|127.0.0.1|*.foo.com'
2. Proxy settings for workspace JVM, Maven, and workspace agent: **`Dhttp.proxyHost=host`**, **`Dhttp.proxyPort=8080`**, **`Dhttps.proxyHost=host`**, **`Dhttps.proxyPort=8080`**, **`Dhttp.nonProxyHosts='localhost|127.0.0.1|*.foo.com'`**
3. http proxy for workspaces. Example: `http://myproxy:8051`
4. https proxy for workspaces. Example: `http://myproxy:8051`

5 no_proxy for workspaces. Example: **`localhost,10.2.34.54`**

Database settings:

```
CHE_EXTERNAL_DB:  ""  (1)
CHE_DB_HOSTNAME:  ""  (2)
CHE_DB_PORT:   "5432"  (3)
CHE_DB_DATABASE:  ""  (4)
CHE_JDBC_USERNAME:  ""  (5)
CHE_JDBC_PASSWORD:  ""  (6)
CHE_DB_ADMIN_PASSWORD:  ""  (7)
```

1 Use external existing Postgres database. Defaults to **`false`** which means a new instance of Postgres will be started. When set to **`true`**, provide connection details and ensure that the database user is a **`SUPERUSER`**

2 Database hostname

3 Database port

4 DB database

5 Database username

6 Database password

7 Your databse administrator password

Keycloak settings:

```
CHE_EXTERNAL_KEYCLOAK:   ""  (1)
CHE_KEYCLOAK_AUTH__SERVER__URL: ""  (2)
CHE_KEYCLOAK_ADMIN_USERNAME:  ""  (3)
CHE_KEYCLOAK_ADMIN_PASSWORD:  "admin"  (4)
CHE_KEYCLOAK_REALM:   ""  (5)
CHE_KEYCLOAK_CLIENT__ID:  ""  (6)
```

1 Connect to an existing external Keycloak and skip deploying a dedicated Keycloak instance. Defaults to **`false`**.

2 Keycloak or Red Hat SSO.

3 Keyloak administrator username. Not required when connecting to an existing realm.

4 Keycloak administrator password. Auto-generated if empty.

5 Red Hat SSO realm.

6 Red Hat SSO client ID

Operator settings:

-

```
WAIT_DEPLOYMENT_TIMEOUT:  "" (1)
```

(1) wait timeout for an Operator to watch deployments. Defaults to 420 seconds.

2.3. VIEWING CODEREADY WORKSPACES INSTALLATION LOGS

You can view the installation logs in the terminal or from the OpenShift console.

2.3.1. Viewing CodeReady Workspaces installation logs in the terminal

To view the installation logs on the terminal, take the following steps:

1. To obtain the names of the pods, run:

   ```
   $ oc get pods
   ```

2. To view the logs for the pod, run:

   ```
   $ oc logs --follow=false <log-name>
   ```

   The following is an example output:

   ```
   time="2019-02-06T09:38:52Z" level=info msg="Provisioning resources
   in pod postgres-7dd4d9cd8f-dwk77"
   time="2019-02-06T09:38:52Z" level=info msg="Provisioning completed"
   time="2019-02-06T09:38:53Z" level=info msg="Waiting for deployment
   keycloak. Default timeout: 420 seconds"
   time="2019-02-06T09:41:02Z" level=info msg="keycloak successfully
   deployed"
   time="2019-02-06T09:41:02Z" level=info msg="Provisioning resources
   in pod keycloak-66f64ddd49-xmsk7"
   time="2019-02-06T09:41:11Z" level=info msg="Provisioning completed"
   time="2019-02-06T09:41:11Z" level=info msg="Waiting for deployment
   che. Default timeout: 420 seconds"
   time="2019-02-06T09:42:39Z" level=info msg="che successfully
   deployed"
   time="2019-02-06T09:42:39Z" level=info msg="Che is available at:
   http://codeready-codeready.192.168.42.192.nip.io"
   time="2019-02-06T09:42:39Z" level=info msg="Deployment took
   4m43.240319357s"
   ```

2.3.2. Viewing CodeReady Workspaces installation logs in the OpenShift console

To view installation logs in OpenShift console, take the following steps:

1. Navigate to ***<OpenShift-IP>*/console**.
2. In the **My Projects** pane, click **codeready**.
3. Click **Applications** > **Pods**. Click **che-operator**.
4. Click **Logs** and click **Follow**.

   ■

```
time="2019-02-06T09:38:52Z" level=info msg="Provisioning completed"
time="2019-02-06T09:38:53Z" level=info msg="Waiting for deployment
keycloak. Default timeout: 420 seconds"
time="2019-02-06T09:41:02Z" level=info msg="keycloak successfully
deployed"
time="2019-02-06T09:41:02Z" level=info msg="Provisioning resources
in pod keycloak-66f64ddd49-xmsk7"
time="2019-02-06T09:41:11Z" level=info msg="Provisioning completed"
time="2019-02-06T09:41:11Z" level=info msg="Waiting for deployment
che. Default timeout: 420 seconds"
time="2019-02-06T09:42:39Z" level=info msg="che successfully
deployed"
time="2019-02-06T09:42:39Z" level=info msg="Che is available at:
http://codeready-codeready.192.168.42.192.nip.io "
time="2019-02-06T09:42:39Z" level=info msg="Deployment took
4m43.240319357s"
```

2.4. VIEWING CODEREADY WORKSPACES OPERATION LOGS

After the CodeReady Workspaces pods are created, you can view the operation logs of the application in the terminal or through the OpenShift console.

2.4.1. Viewing CodeReady Workspaces operation logs in the terminal

To view the operation logs on the terminal, run the following commands:

1. To view the names of the pods, run:

```
$ oc get pods
```

This command shows the pods that have been created:

```
NAME                          READY     STATUS      RESTARTS   AGE
che-58d8456f55-89pvw          1/1       Running     0          1h
che-operator                  0/1       Completed   0          1h
keycloak-66f64ddd49-xmsk7     1/1       Running     0          1h
postgres-7dd4d9cd8f-dwk77     1/1       Running     0          1h
```

2. To view the operation log for a specific pod, run:

```
$ oc logs --follow=false <log-name>
```

The output of this command for the **che-58d8456f55-89pvw** pod (as an example) is as follows:

```
2019-02-06 09:42:36,818[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 957]        - Deploying web application
archive [/home/jboss/codeready/tomcat/webapps/dashboard.war]
2019-02-06 09:42:37,976[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 1020]       - Deployment of web
application archive
[/home/jboss/codeready/tomcat/webapps/dashboard.war] has finished in
[1,158] ms
2019-02-06 09:42:37,976[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 957]        - Deploying web application
```

```
archive [/home/jboss/codeready/tomcat/webapps/swagger.war]
2019-02-06 09:42:38,490[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 1020]       - Deployment of web
application archive
[/home/jboss/codeready/tomcat/webapps/swagger.war] has finished in
[513] ms
2019-02-06 09:42:38,490[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 957]        - Deploying web application
archive [/home/jboss/codeready/tomcat/webapps/workspace-loader.war]
2019-02-06 09:42:38,620[ost-startStop-1]  [INFO ]
[o.a.c.startup.HostConfig 1020]       - Deployment of web
application archive [/home/jboss/codeready/tomcat/webapps/workspace-
loader.war] has finished in [130] ms
2019-02-06 09:42:38,623[main]             [INFO ]
[o.a.c.http11.Http11NioProtocol 588]  - Starting ProtocolHandler
["http-nio-8080"]
2019-02-06 09:42:38,646[main]             [INFO ]
[o.a.catalina.startup.Catalina 700]   - Server startup in 34592 ms
```

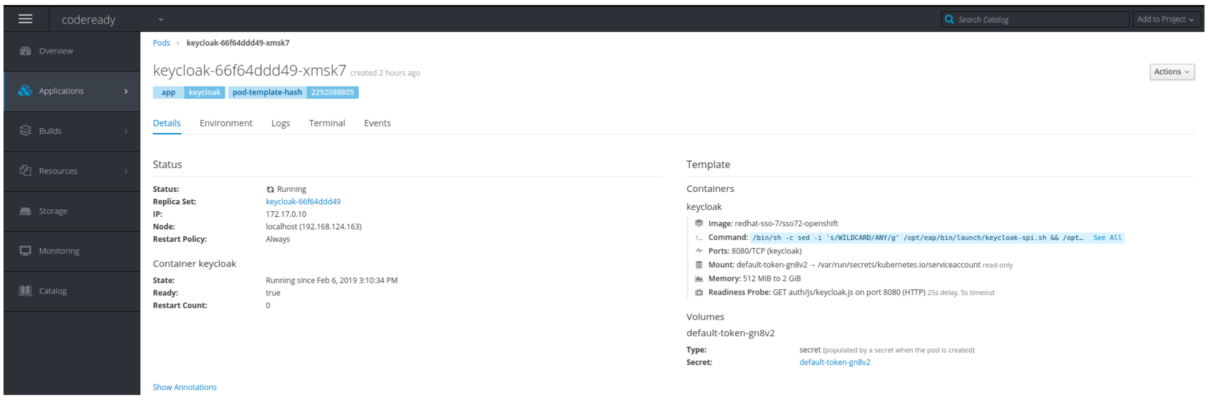
3. For operation logs of the other pods, run:

   - For the **keycloak-66f64ddd49-xmsk7** pod: **`oc logs --follow=false keycloak-66f64ddd49-xmsk7`**
   - For the **postgres-7dd4d9cd8f-dwk77** pod: **`oc logs --follow=false postgres-7dd4d9cd8f-dwk77`**
   - For the **che-operator** pod: **`oc logs --follow=false che-operator`**

2.4.2. Viewing CodeReady Workspaces operation logs in the OpenShift console

To view the operation logs in the OpenShift console, take the following steps:

1. Navigate to **`<OpenShift-IP>/console`**.
2. In the **My Projects** pane, click **codeready**.
3. In the **Overview** tab, click the application that you want to view the logs for.
4. In the **Deployments > *<application-name>*** window, click the name of the pod.
5. Scroll down to the **Pods** section and click the *<pod-name>*.
6. Click **Logs**



keycloak-66f64ddd49-xmsk7 created 2 hours ago

Status

Status: Running
Replica Set: keycloak-66f64ddd49
IP: 172.17.0.10
Node: localhost (192.168.124.163)
Restart Policy: Always

Container keycloak

State: Running since Feb 6, 2019 3:10:34 PM
Ready: true
Restart Count: 0

Template

Containers

keycloak

Image: redhat-sso-7/sso72-openshift
Command: `/bin/sh -c sed -i 's/WILDCARD/ANY/g' /opt/eap/bin/launch/keycloak-spi.sh && /opt…`
Ports: 8080/TCP (keycloak)
Mount: default-token-gn8v2 → /var/run/secrets/kubernetes.io/serviceaccount read-only
Memory: 512 MiB to 2 GiB
Readiness Probe: GET auth/js/keycloak.js on port 8080 (HTTP) 25s delay, 5s timeout

Volumes

default-token-gn8v2

Type: secret (populated by a secret when the pod is created)
Secret: default-token-gn8v2

Figure 2.2. Clicking View Log

7. Click **Follow** to follow the log.

```
09:41:00,260 INFO  [org.jboss.resteasy.resteasy_jaxrs.i18n]
(ServerService Thread Pool -- 62) RESTEASY002220: Adding singleton
resource org.keycloak.services.resources.RealmsResource from
Application class
org.keycloak.services.resources.KeycloakApplication
09:41:00,260 INFO  [org.jboss.resteasy.resteasy_jaxrs.i18n]
(ServerService Thread Pool -- 62) RESTEASY002220: Adding singleton
resource org.keycloak.services.resources.admin.AdminRoot from
Application class
org.keycloak.services.resources.KeycloakApplication
09:41:00,260 INFO  [org.jboss.resteasy.resteasy_jaxrs.i18n]
(ServerService Thread Pool -- 62) RESTEASY002210: Adding provider
singleton org.keycloak.services.util.ObjectMapperResolver from
Application class
org.keycloak.services.resources.KeycloakApplication
09:41:00,260 INFO  [org.jboss.resteasy.resteasy_jaxrs.i18n]
(ServerService Thread Pool -- 62) RESTEASY002220: Adding singleton
resource org.keycloak.services.resources.RobotsResource from
Application class
org.keycloak.services.resources.KeycloakApplication
09:41:00,260 INFO  [org.jboss.resteasy.resteasy_jaxrs.i18n]
(ServerService Thread Pool -- 62) RESTEASY002220: Adding singleton
resource org.keycloak.services.resources.ServerVersionResource from
Application class
org.keycloak.services.resources.KeycloakApplication
```

2.5. CONFIGURING CODEREADY WORKSPACES TO WORK BEHIND THE HTTPS PROXY SERVER

This procedure describes how to configure CodeReady Workspaces for use in a deployment behind a proxy. To access external resources (for example, to download Maven artifacts to build Java projects), change the workspace configuration.

Prerequisites

- Installed CodeReady Workspaces
- Deployment parameters in **`config.yaml`** reflecting proxy set up

Procedure

For example, to build a Java project, delete HTTPS-proxy related parameters from the `JAVA_OPTS` value. Edit the HTTPS-proxy related parameters in the **che configuration map**:

1. In the OpenShift web console, click **Resources** > **Config Maps** > **che**.
2. Click **Actions** > **Edit YAML**.
3. In the editor, navigate to the `JAVA_OPTS` section.
4. Delete the following HTTPS-proxy related parameters from the `JAVA_OPTS` values:

   ```
   -Dhttps.proxyHost
   -Dhttps.proxyPort
   -Dhttps.nonProxyHosts
   ```

5. To re-deploy CodeReady Workspaces from OpenShift web console, click **Applications** > **Deployments** > **codeready** > **Deploy**.

CHAPTER 3. USING THE CHE 7 IDE IN CODEREADY WORKSPACES

IMPORTANT

CodeReady Workspaces 1.0.2 is based on upstream Che 6. The next version of Che, Che 7, is being worked on. To try this upcoming version of Che 7 in CodeReady Workspaces, follow the instructions in this section.

Che 7-based stacks are not included by default in CodeReady Workspaces. However, you can use a Che 7-based workspace configuration in CodeReady Workspaces. To use the Che 7-based stacks in CodeReady Workspaces, take the following steps:

1. In the **Dashboard**, click **Workspaces** and then click **Add Workspace**.
2. Select any stack from the list and click the dropdown icon next to **CREATE & OPEN**.
3. Click **Create & Proceed Editing**.
4. Click the **Config** tab and replace the default configuration content with the following content:

```
{
  "environments": {},
  "projects": [],
  "name": "che7",
  "attributes": {
    "editor": "org.eclipse.che.editor.theia:1.0.0",
    "plugins": "che-machine-exec-plugin:0.0.1"
  },
  "commands": [],
  "links": []
}
```

5. To add additional Che plugins (for example to add support for a particular language), add them in the workspace configuration attributes#plugins list. For a list of available plugins, see https://github.com/eclipse/che-plugin-registry/.
6. To add a runtime environment (for example Java runtime environment), add the following content in the **Config** tab:

```
"environments": {
     "default": {
      "machines": {
        "ws/dev": {
          "attributes": {
            "memoryLimitBytes": "536870912"
          },
          "servers": {},
          "volumes": {
            "projects": {
              "path": "/projects"
            }
          },
          "installers": [],
          "env": {}
        }
      },
      "recipe": {
        "type": "kubernetes",
        "content": "kind: List\nitems:\n - \n  apiVersion: v1\n
kind: Pod\n  metadata:\n   name: ws\n  spec:\n   containers:\n    -
\n     image: 'eclipse/che-dev:nightly'\n     name: dev\n
resources:\n      limits:\n       memory: 512Mi\n",
        "contentType": "application/x-yaml"
      }
    }
  }
```

7. To define it as the default environment, add the following content:

```
"defaultEnv": "default"
"environments": {
     "default": {...} }
```

8. Click the **Open** button and run the Che 7-based workspace that you just created.

CHAPTER 4. USING THE ANALYTICS PLUG-IN IN CODEREADY WORKSPACES

The Analytics plug-in provides insights about application dependencies: security vulnerabilities, license compatibility, and AI-based guidance for additional, alternative dependencies.

The Analytics plug-in is enabled by default in the Java and NodeJS stacks in CodeReady Workspaces. When a user opens the **`pom.xml`** or the **`package.json`** files, the dependencies are analyzed. The editor shows warnings for available CVEs or issues with any dependency.

CHAPTER 5. USING VERSION CONTROL

CodeReady Workspaces natively supports the Git version control system (VCS), which is provided by the JGit library. Versioning functionality is available in the IDE and in the terminal.

The following sections describe how to connect and authenticate users to a remote Git repository. Any operations that involve authentication on the remote repository need to be done via the IDE interface unless authentication is configured separately for the workspace machine (terminal, commands).

Private repositories require a secure SSH connection. In order to connect to Git repositories over SSH, an SSH key pair needs to be generated. SSH keys are saved in user preferences, so the same key can be used in all workspaces.

NOTE

HTTPS Git URLs can only be used to push changes when OAuth authentication is enabled. See Enabling authentication with social accounts and brokering.

5.1. GENERATING AND UPLOADING SSH KEYS

SSH keys can be generated in the CodeReady Workspaces user interface.

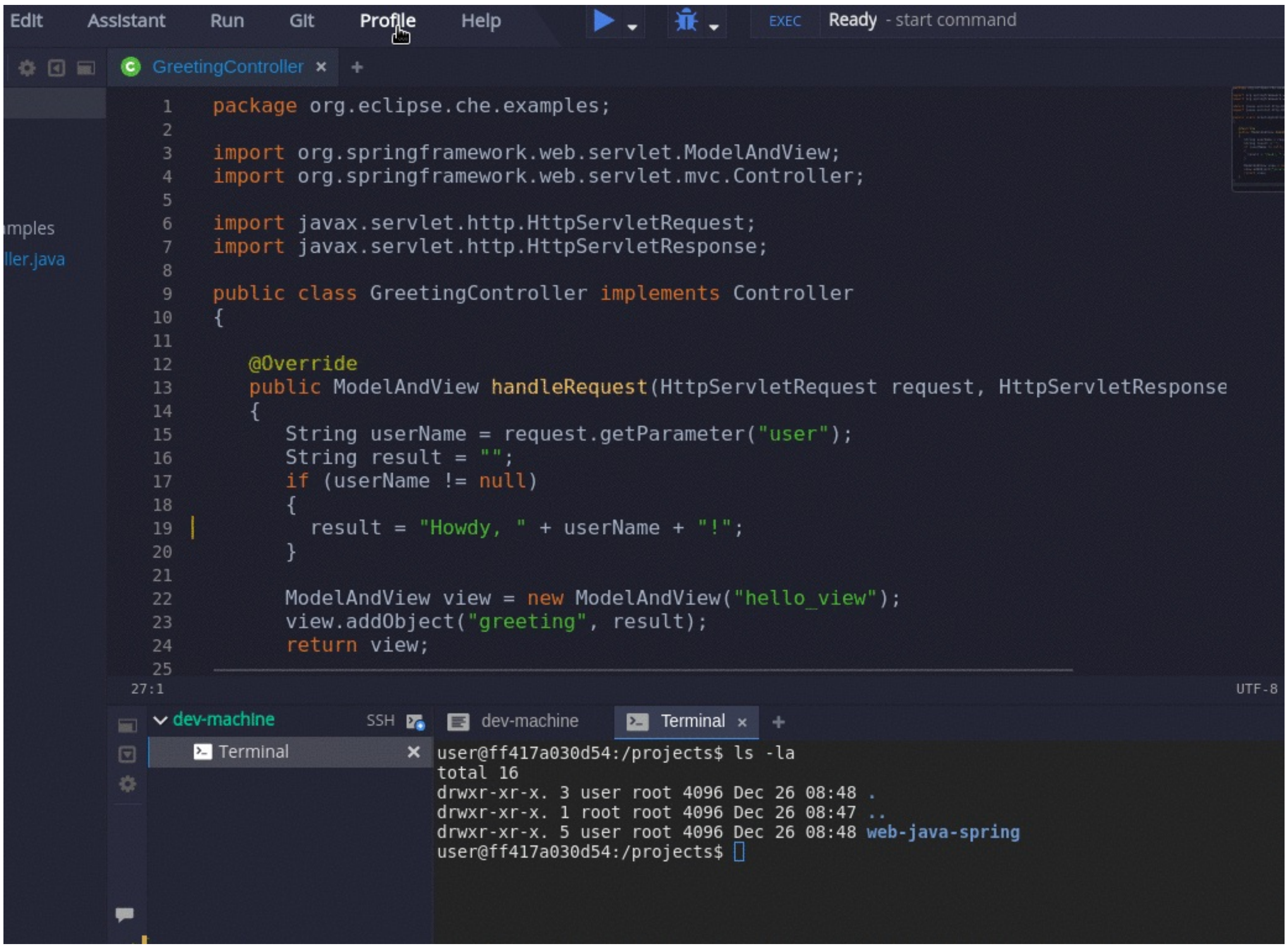
1. Go to **Profile** > **Preferences** > **SSH** > **VCS**, and click the **`Generate Key`** button.
2. When prompted to provide the host name for your repository, use the bare host name (do not include **`www`** or **`https`**) as in the example below.
3. Save the resulting key to your Git-hosting provider account.

IMPORTANT

The host name is an actual host name of a VCS provider. Examples: **`github.com`**, **`bitbucket.org`**.

5.1.1. Using existing SSH keys

You can upload an existing public key instead of creating a new SSH key. When uploading a key, add the host name (using no **`www`** or **`https`** - as in the example below). Note that the **Public key** > **View** button is not be available when using this option because the public file should be generated already.



```
package org.eclipse.che.examples;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GreetingController implements Controller
{

   @Override
   public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
   {
      String userName = request.getParameter("user");
      String result = "";
      if (userName != null)
      {
        result = "Howdy, " + userName + "!";
      }

      ModelAndView view = new ModelAndView("hello_view");
      view.addObject("greeting", result);
      return view;
```

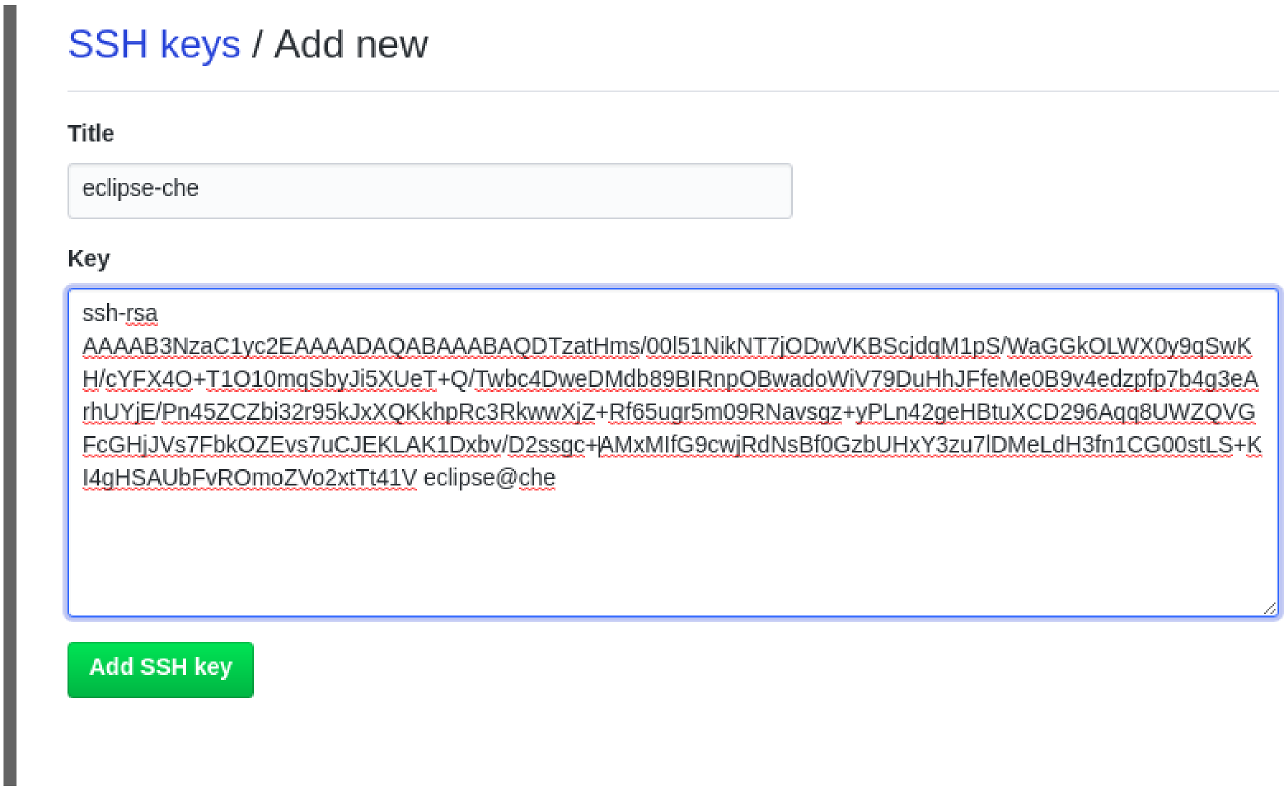
```
user@ff417a030d54:/projects$ ls -la
total 16
drwxr-xr-x. 3 user root 4096 Dec 26 08:48 .
drwxr-xr-x. 1 root root 4096 Dec 26 08:47 ..
drwxr-xr-x. 5 user root 4096 Dec 26 08:48 web-java-spring
user@ff417a030d54:/projects$
```

The following examples are specific to GitHub and GitLab, but a similar procedure can be used with all Git or SVN repository providers that use SSH authentication. See documentation provided by other providers for additional assistance.

Example 5.1. GitHub example

When not using GitHub OAuth, you can manually upload a key. To add the associated public key to a repository or account on **github.com**:

1. Click your user icon (top right).
2. Go to **Settings > SSH and GPG keys** and click the **New SSH key** button.
3. Enter a title and paste the public key copied from CodeReady Workspaces to the **Key** text field.



Title

eclipse-che

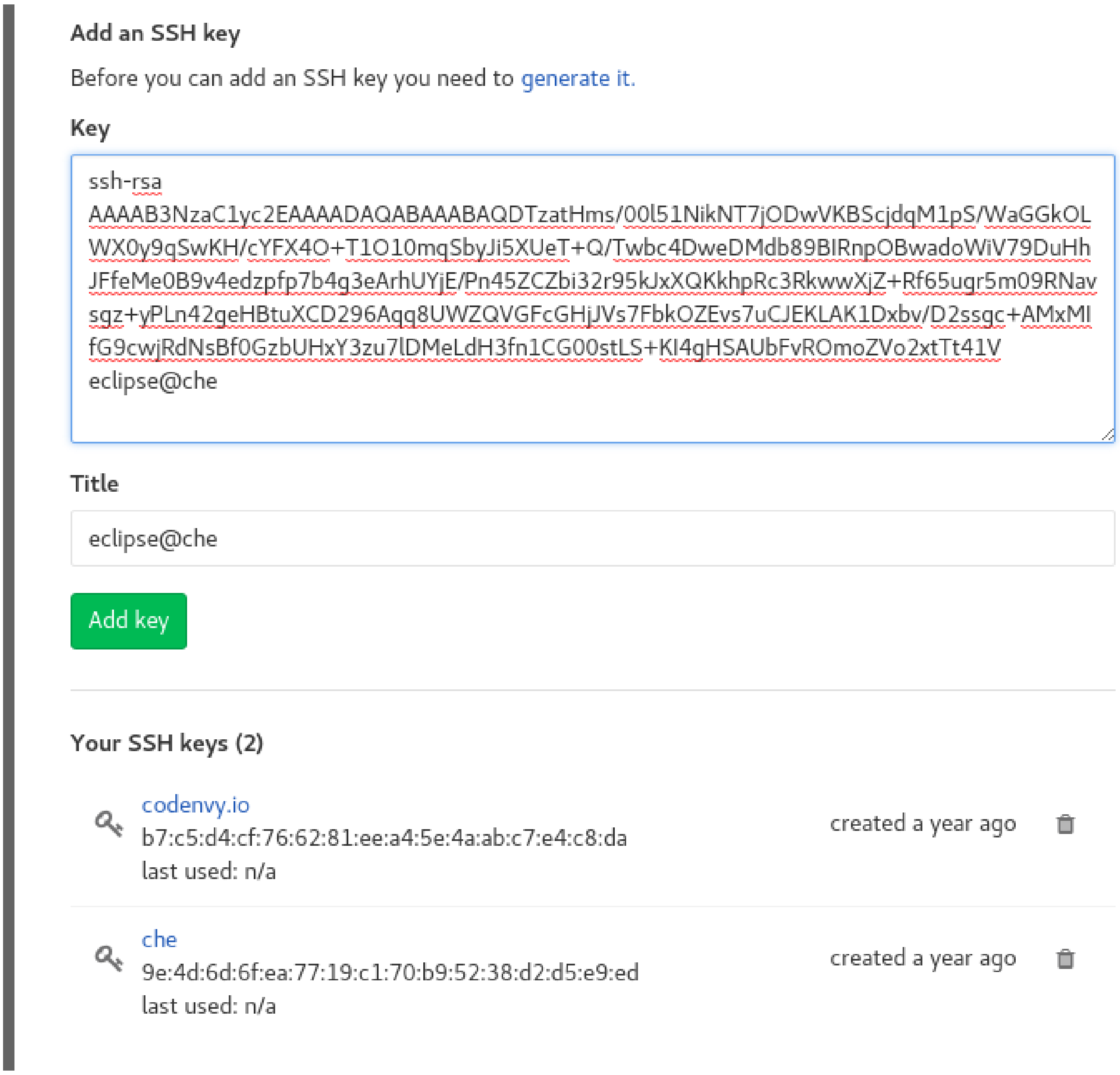
Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDTzatHms/00l51NikNT7jODwVKBScjdqM1pS/WaGGkOLWX0y9qSwK
H/cYFX4O+T1O10mqSbyJi5XUeT+Q/Twbc4DweDMdb89BIRnpOBwadoWiV79DuHhJFfeMe0B9v4edzpfp7b4g3eA
rhUYjE/Pn45ZCZbi32r95kJxXQKkhpRc3RkwwXjZ+Rf65ugr5m09RNavsgz+yPLn42geHBtuXCD296Aqq8UWZQVG
FcGHjJVs7FbkOZEvs7uCJEKLAK1Dxbv/D2ssgc+AMxMIfG9cwjRdNsBf0GzbUHxY3zu7lDMeLdH3fn1CG00stLS+K
I4gHSAUbFvROmoZVo2xtTt41V eclipse@che
```

Example 5.2. GitLab example

To add the associated public key to a Git repository or account on **gitlab.com**:

1. Click your user icon (top right).
2. Go to **Settings > SSH Keys**.
3. Enter a title and paste the public key copied from CodeReady Workspaces to the **Key** text field.


Add an SSH key

Before you can add an SSH key you need to generate it.

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDTzatHms/00l51NikNT7jODwVKBScjdqM1pS/WaGGkOL
WX0y9qSwKH/cYFX4O+T1O10mqSbyJi5XUeT+Q/Twbc4DweDMdb89BIRnpOBwadoWiV79DuHh
JFfeMe0B9v4edzpfp7b4q3eArhUYjE/Pn45ZCZbi32r95kJxXQKkhpRc3RkwwXjZ+Rf65uqr5m09RNav
sgz+yPLn42geHBtuXCD296Aqq8UWZQVGFcGHjJVs7FbkOZEvs7uCJEKLAK1Dxbv/D2ssgc+AMxMI
fG9cwjRdNsBf0GzbUHxY3zu7lDMeLdH3fn1CG00stLS+KI4qHSAUbFvROmoZVo2xtTt41V
eclipse@che
```

Title

eclipse@che

Your SSH keys (2)

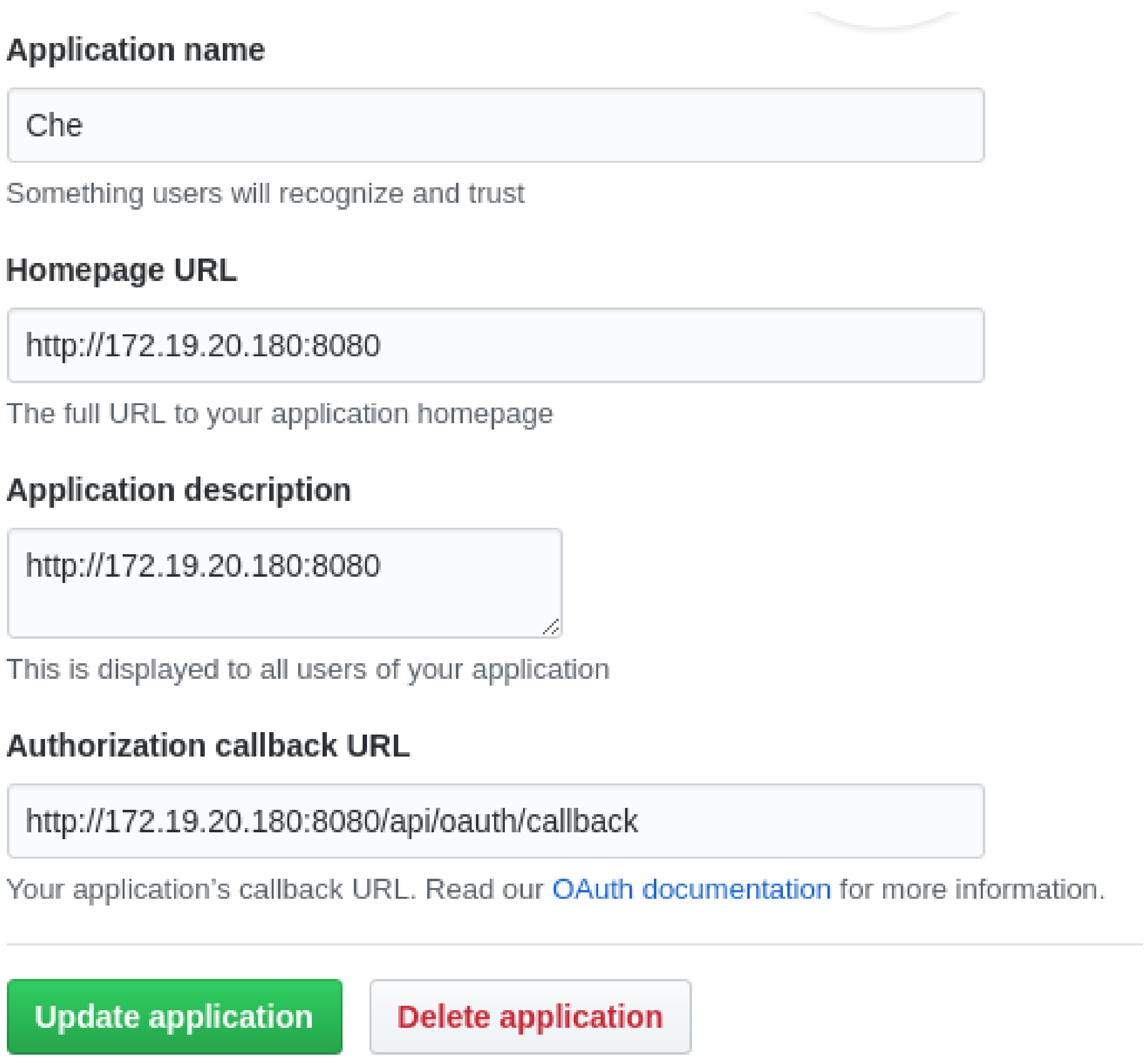
codenvy.io
b7:c5:d4:cf:76:62:81:ee:a4:5e:4a:ab:c7:e4:c8:da
last used: n/a
created a year ago

che
9e:4d:6d:6f:ea:77:19:c1:70:b9:52:38:d2:d5:e9:ed
last used: n/a
created a year ago

5.2. CONFIGURING GITHUB OAUTH

OAuth for Github allows users to import projects using SSH addresses (**`git@`**), push to repositories, and use the pull request panel. To enable automatic SSH key upload to GitHub for users:

1. On **github.com**, click your user icon (top right).
2. Go to **Settings > Developer settings > OAuth Apps**.
3. Click the **`Register a new application`** button.
4. In the **Application name** field, enter, for example, **`CodeReady Workspaces`**.
5. In the **Homepage URL** field, enter **`http://${CHE_HOST}:${CHE_PORT}`**.
6. In the **Authorization callback URL** field, enter **`http://${CHE_HOST}:${CHE_PORT}/api/oauth/callback`**.


Application name

Che

Something users will recognize and trust

Homepage URL

http://172.19.20.180:8080

The full URL to your application homepage

Application description

http://172.19.20.180:8080

This is displayed to all users of your application

Authorization callback URL

http://172.19.20.180:8080/api/oauth/callback

Your application's callback URL. Read our OAuth documentation for more information.

7. On OpenShift, update the deployment configuration.

```
CHE_OAUTH_GITHUB_CLIENTID=<your-github-client-id>
CHE_OAUTH_GITHUB_CLIENTSECRET=<your-github-secret>
CHE_OAUTH_GITHUB_AUTHURI=https://github.com/login/oauth/authorize
CHE_OAUTH_GITHUB_TOKENURI=https://github.com/login/oauth/access_toke
n
CHE_OAUTH_GITHUB_REDIRECTURIS=http://${CHE_HOST}:${CHE_PORT}/api/oau
th/callback
```

NOTE

- Substitute all occurrences of ***`${CHE_HOST}`*** and ***`${CHE_PORT}`*** with the URL and port of your CodeReady Workspaces installation.
- Substitute ***`<your-github-client-id>`*** and ***`<your-github-secret>`*** with your GitHub client ID and secret.

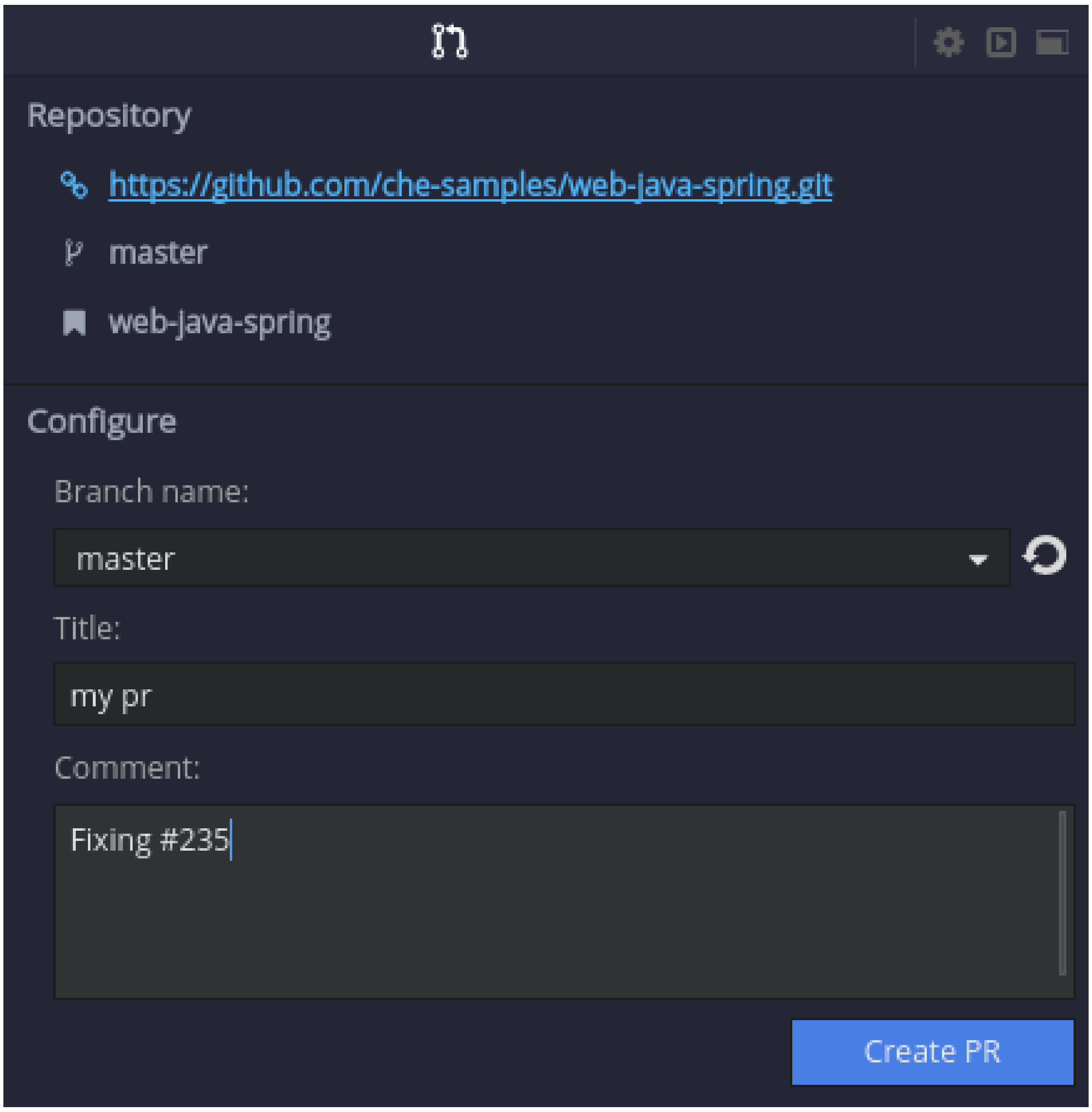
Once OAuth is configured, SSH keys are generated and uploaded automatically to GitHub by a user in the IDE in **Profile** > **Preferences** > **SSH** > **VCS** by clicking the *Octocat* icon. You can connect to your GitHub account and choose repositories to clone, rather than having to manually type (or paste) GitHub project URLs.

5.3. CONFIGURING GITLAB OAUTH

OAuth integration with GitLab is not supported. Although GitLab supports OAuth for clone operations, pushes are not supported. A feature request to add support exists in the GitLab issue management system: Allow runners to push via their CI token.

5.4. SUBMITTING PULL REQUESTS USING THE BUILT-IN PULL REQUEST PANEL

CodeReady Workspaces provides a **Pull Request** panel to simplify the creation of pull requests for GitHub, BitBucket, and Microsoft VSTS (with Git) repositories.



5.5. SAVING COMMITTER NAME AND EMAIL

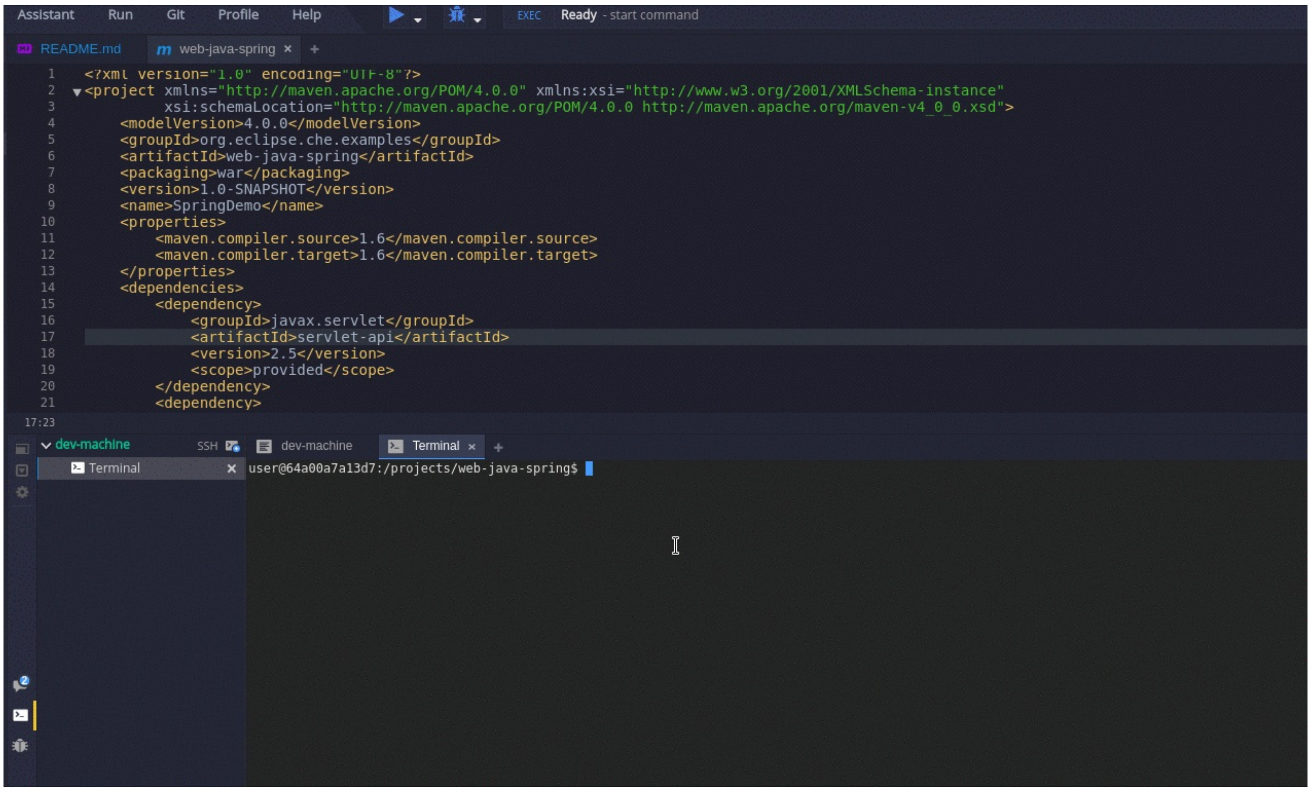
Committer name and email are set in **Profile** > **Preferences** > **Git** > **Committer**. Once set, every commit will include this information.

5.6. INTERACTING WITH GIT FROM A WORKSPACE

After importing a repository, you can perform the most common Git operations using interactive menus or as terminal commands.

NOTE

Terminal Git commands require their own authentication setup. This means that keys generated in the IDE work only when Git is used through the IDE menus. Git installed in a terminal is a different Git system. You can generate keys in **`~/.ssh`** there as well.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.eclipse.che.examples</groupId>
    <artifactId>web-java-spring</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>SpringDemo</name>
    <properties>
        <maven.compiler.source>1.6</maven.compiler.source>
        <maven.compiler.target>1.6</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
```

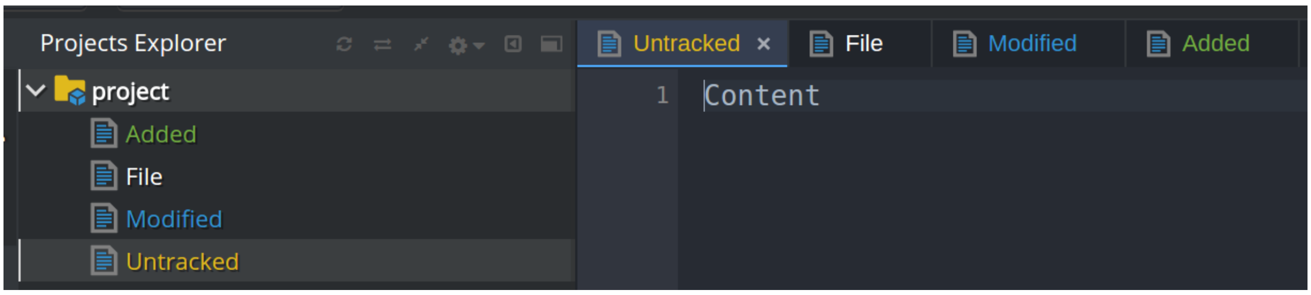
```
user@64a00a7a13d7:/projects/web-java-spring$
```

Use keyboard shortcuts to access the most frequently used Git functionality faster:

Commit	**`Alt`**+**`C`**
Push to remote	**`Alt`**+**`Shift`**+**`C`**
Pull from remote	**`Alt`**+**`P`**
Work with branches	**`Ctrl`**+**`B`**
Compare current changes with the latest repository version	**`Ctrl`**+**`Alt`**+**`D`**

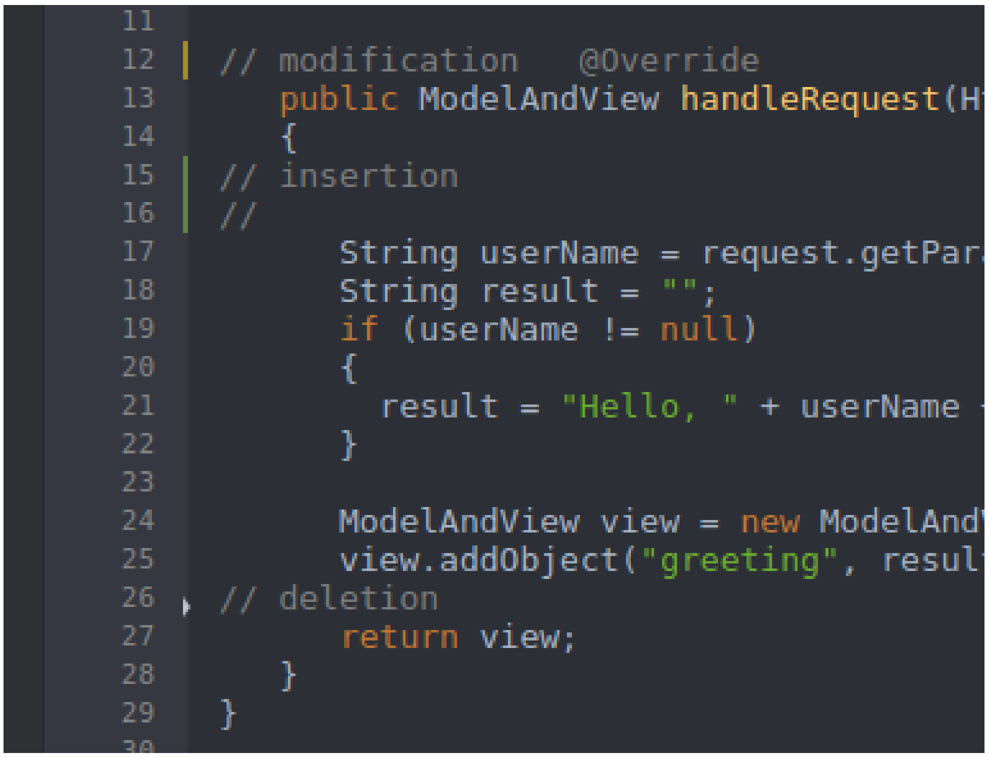
5.7. GIT STATUS HIGHLIGHTING IN THE PROJECT TREE AND EDITOR

Files in project explorer and editor tabs can be colored according to their Git status:



- Green: new files that are staged in index
- Blue: files that contain changes
- Yellow: files that are not staged in index

The editor displays change markers according to file edits:



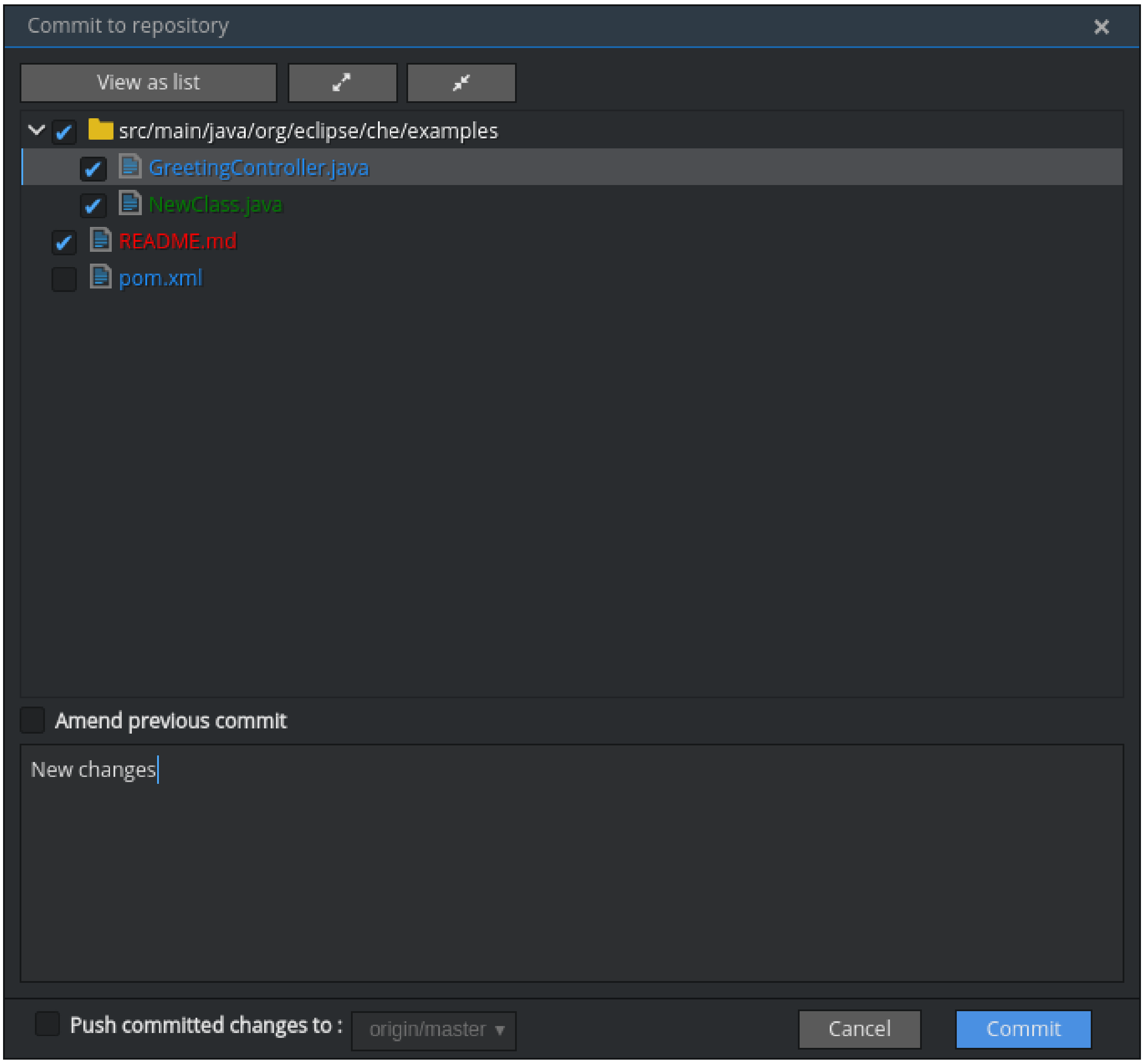
```
// modification   @Override
    public ModelAndView handleRequest(H
    {
// insertion
//
        String userName = request.getPar
        String result = "";
        if (userName != null)
        {
          result = "Hello, " + userName
        }

        ModelAndView view = new ModelAnd
        view.addObject("greeting", resul
// deletion
        return view;
    }
}
```

- Yellow marker: modified line(s)
- Green marker: new line(s)
- White triangle: removed line(s)

5.8. PERFORMING GIT OPERATIONS

5.8.1. Commiting

Commit your changes by navigating to **Git > Commit…** in the main menu, or use the **`Alt`+`C`** shortcut.



New changes

1. Select files that will be added to index and committed. All files in the selected package or folder in the project explorer are checked by default.

2. Type your commit message. Optionally, you can select **Amend previous commit** to modify the previous commit (for more details, see Git commit documentation).

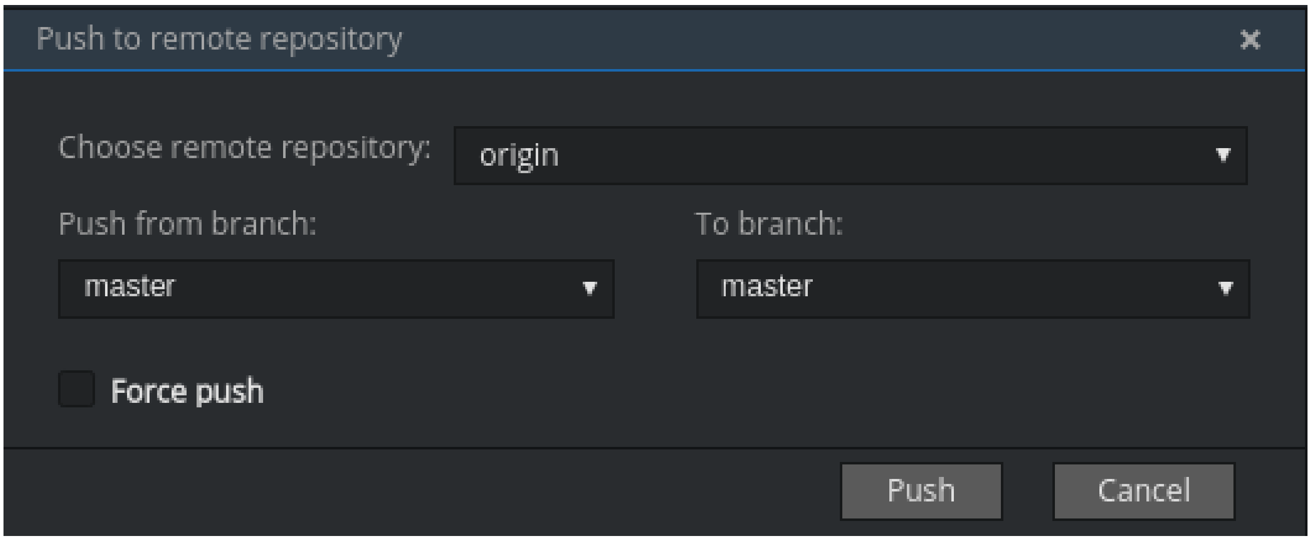
3. To push your commit to a remote repository by checking the **Push committed changes to** check-box and select a remote branch.

4. Click **`Commit`** to proceed (the **`Commit`** button is active when at least one file is selected and a commit message is present, or **Amend previous commit** is checked).

Behavior for files in the list view is the same as in the **Compare** window (see Reviewing changed files section). Double-clicking a file opens diff window with it.

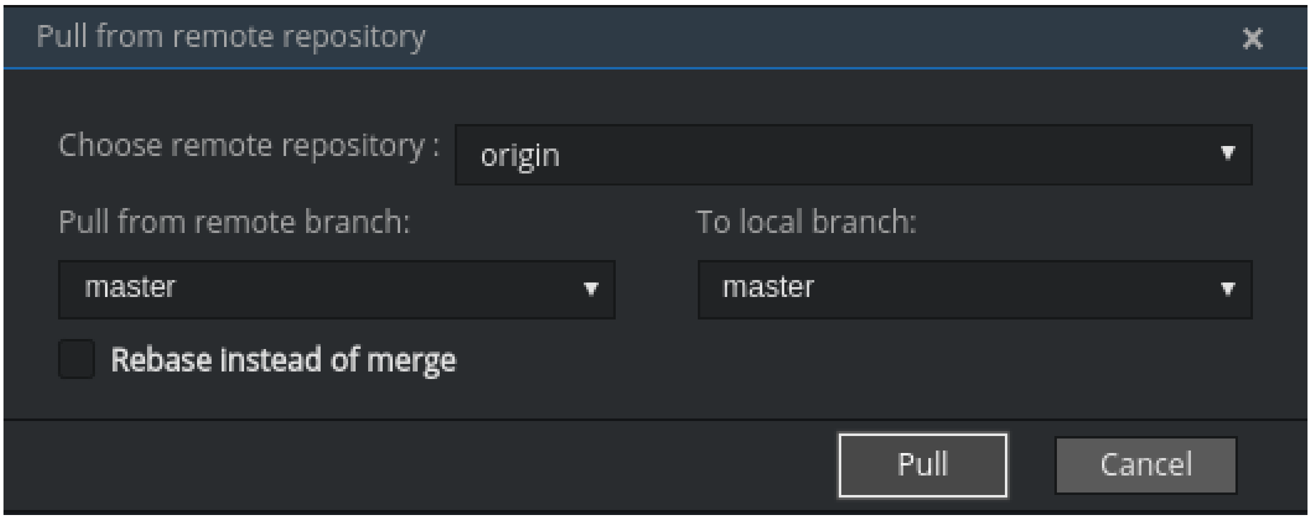
5.8.2. Pushing and pulling

Push your commits by navigating to **Git** > **Remotes…** > **Push** in the main menu, or use the **`Alt`**+**`Shift`**+**`C`** shortcut.



1. Choose the remote repository.
2. Choose the local and remote branch.
3. Optionally, you can force select **Force push**.

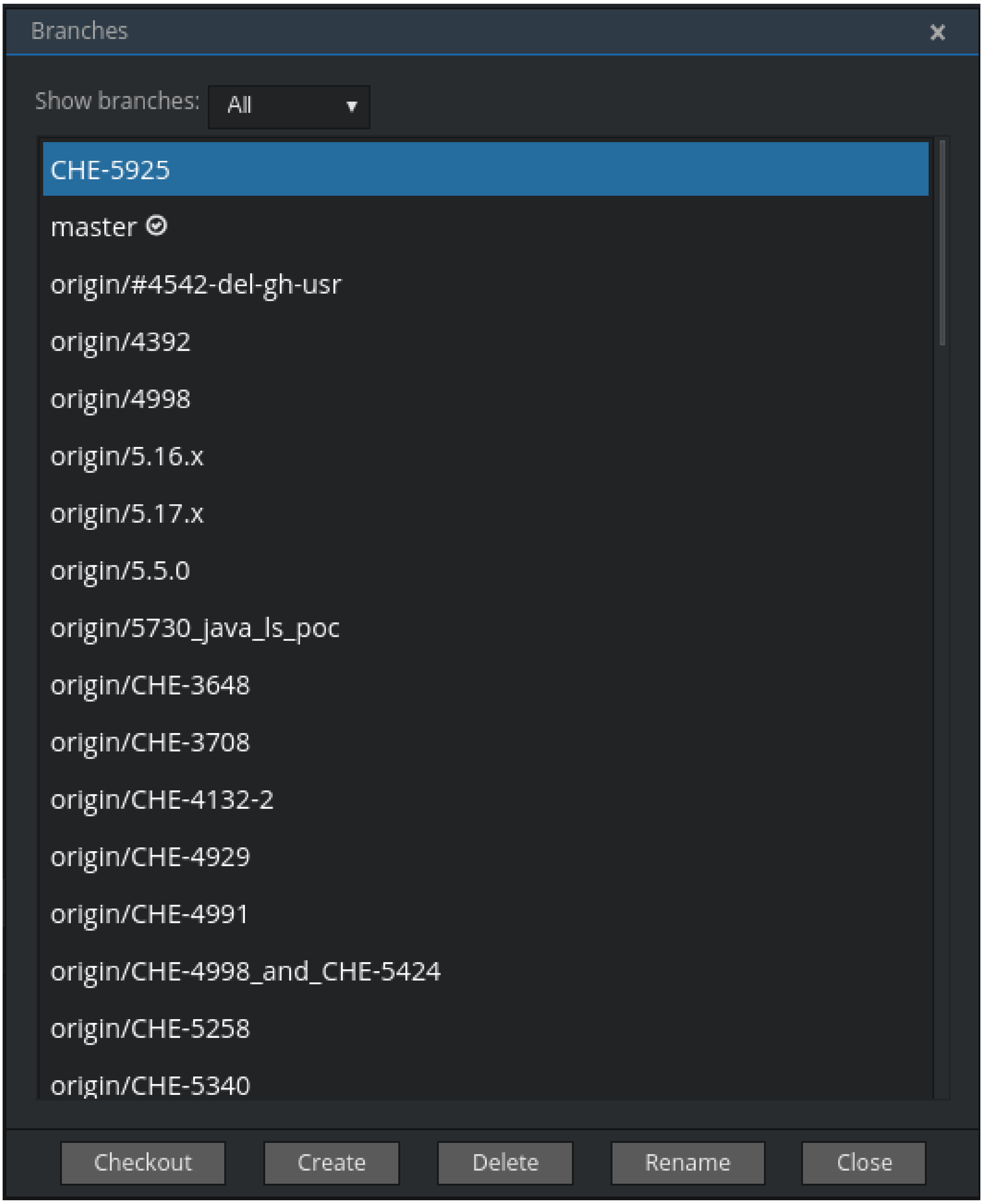
Get changes from a remote repository by navigating to **Git > Remotes… > Pull** in the main menu, or use the **`Alt`**+**`P`** shortcut.



You can use **Rebase instead of merge** to keep your local commits on top (for more information, see Git pull documentation).

5.8.3. Managing branches

Manage your git branches by navigating to **Git > Branches…** in the main menu, or use the **`Ctrl`**+**`B`** shortcut.



CHE-5925
master
origin/#4542-del-gh-usr
origin/4392
origin/4998
origin/5.16.x
origin/5.17.x
origin/5.5.0
origin/5730_java_ls_poc
origin/CHE-3648
origin/CHE-3708
origin/CHE-4132-2
origin/CHE-4929
origin/CHE-4991
origin/CHE-4998_and_CHE-5424
origin/CHE-5258
origin/CHE-5340

You can filter the branches view by choosing to see only local or remote branches.

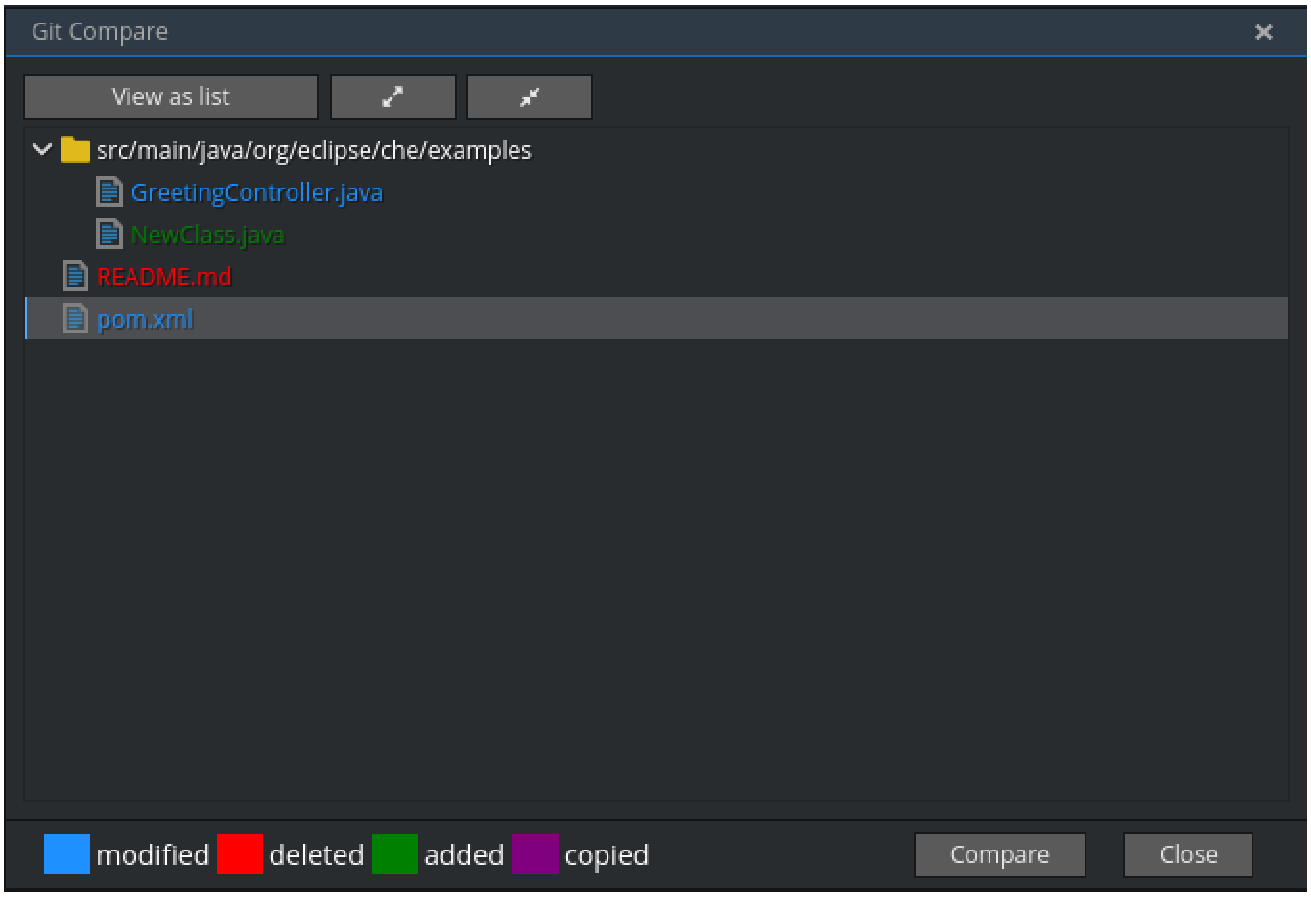
5.9. REVIEWING CHANGED FILES

The **Git Compare** window is used to show files that have changed.

To compare the current state of code to the latest local commit, navigate to **Git** > **Compare** > **Select-to-what** in the main menu, or use the **`Ctrl`**+**`Alt`**+**`D`** shortcut. Another way is to select an object in the project tree and choose **Git** > **Select-to-what** from the context menu of an item.

Listing changed files

The **Git Compare** window shows changed files in the selected object in the project explorer. To see all changes, select a project folder. If only one file has changed, a diff window is shown instead of the compare window.

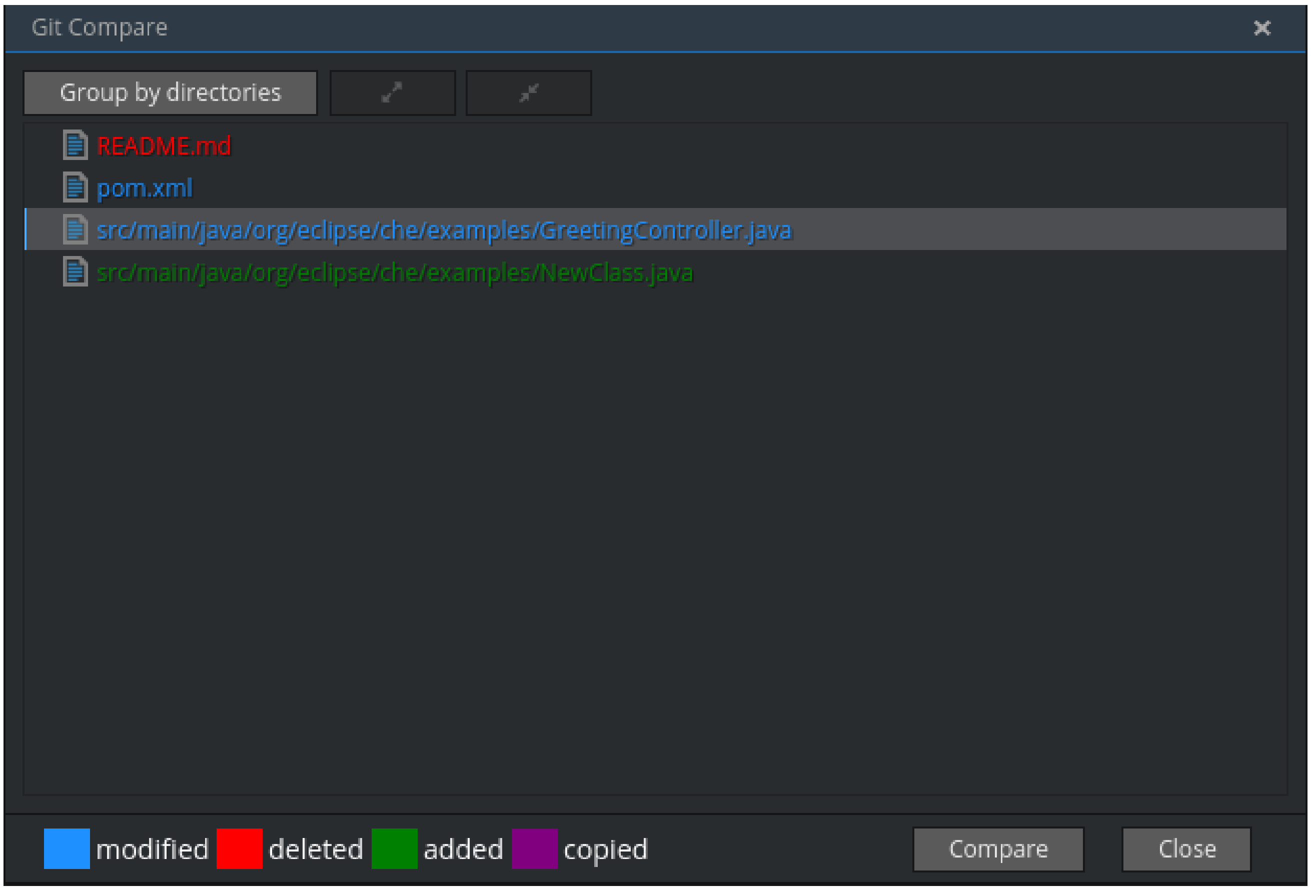


```
src/main/java/org/eclipse/che/examples
    GreetingController.java
    NewClass.java
README.md
pom.xml
```

🟦 modified 🟥 deleted 🟩 added 🟪 copied

By default, affected files are listed as a tree.

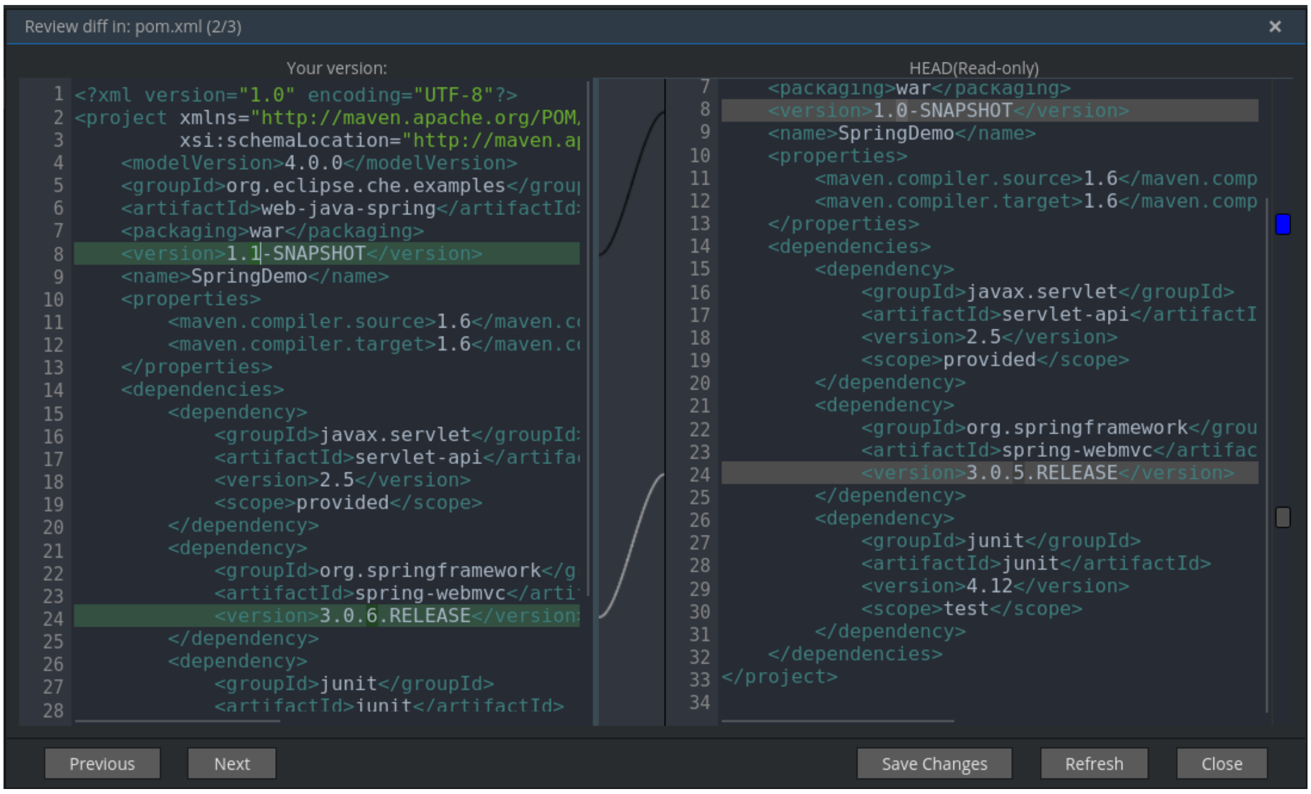
The **Expand all directories** and **Collapse all directories** options help to get a better view. The **`View as list`** button switches the view of changed files to a list, where each file is shown with its full path. To return to the tree view, click **`Group by directories`**.


Git Compare
Group by directories
README.md
pom.xml
src/main/java/org/eclipse/che/examples/GreetingController.java
src/main/java/org/eclipse/che/examples/NewClass.java
modified
deleted
added
copied
Compare
Close

Viewing diffs

To view a diff for a file, select the file and click **Compare**, or double-click the file name.

You can review changes between two states of code. To view the diff, go to **Git > Compare > Select-to-what** in main menu. If more than one file has changed, a list of the changed files is opened first. To select a file to compare, double-click it, or select a file, and then click **Compare**. Another way to open a diff is to select a file in the **Projects Explorer** and choose **Git > Select-to-what** from its context menu or directly from the context menu in the editor.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM,
         xsi:schemaLocation="http://maven.a
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.eclipse.che.examples</grou
    <artifactId>web-java-spring</artifactId
    <packaging>war</packaging>
    <version>1.1-SNAPSHOT</version>
    <name>SpringDemo</name>
    <properties>
        <maven.compiler.source>1.6</maven.c
        <maven.compiler.target>1.6</maven.c
    </properties>
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId
            <artifactId>servlet-api</artifa
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</g
            <artifactId>spring-webmvc</arti
            <version>3.0.6.RELEASE</version
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
```

```
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>SpringDemo</name>
    <properties>
        <maven.compiler.source>1.6</maven.comp
        <maven.compiler.target>1.6</maven.comp
    </properties>
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactI
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</grou
            <artifactId>spring-webmvc</artifac
            <version>3.0.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Your changes are shown on the left, and the file being compared to is on the right. The left pane can be used for editing and fixing your changes.

To review multiple files, you can navigate between them using the **`Previous`** (or **`Alt+.`**) and **`Next`** (or **`Alt+,`**) buttons. The number of files for review is displayed in the title of the diff window.

The **`Refresh`** button updates the difference links between the two panes.

CHAPTER 6. CODEREADY WORKSPACES ADMINISTRATION GUIDE

6.1. RAM PREREQUISITES

You must have at least 5 GB of RAM to run CodeReady Workspaces. The Keycloak authorization server and PostgreSQL database require the extra RAM. CodeReady Workspaces uses RAM in this distribution:

- CodeReady Workspaces server: approximately 750 MB
- Keycloak: approximately 1 GB
- PostgreSQL: approximately 515 MB
- Workspaces: 2 GB of RAM per workspace. The total workspace RAM depends on the size of the workspace runtime(s) and the number of concurrent workspace pods.

6.1.1. Setting default workspace RAM limits

The default workspace RAM limit and the RAM allocation request can be configured by passing the **`CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB`** and **`CHE_WORKSPACE_DEFAULT__MEMORY__REQUEST__MB`** parameters to a CodeReady Workspaces deployment.

For example, use the following configuration to limit the amount of RAM used by workspaces to 2048 MB and to request the allocation of 1024 MB of RAM:

```
$ oc set env dc/che CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB=2048 \
                         CHE_WORKSPACE_DEFAULT__MEMORY__REQUEST__MB=1024
```

NOTE

- The user can override the default values when creating a workspace.
- A RAM request greater than the RAM limit is ignored.

6.2. REQUIREMENTS FOR RESOURCE ALLOCATION AND QUOTAS

Workspace pods are created in the account of the user who deploys CodeReady Workspaces. The user needs enough quota for RAM, CPU, and storage to create the pods.

6.3. SETTING UP THE PROJECT WORKSPACE

Workspace objects are created differently depending on the configuration. CodeReady Workspaces currently supports two different configurations:

- Single OpenShift project
- Multi OpenShift project

6.3.1. Setting up a single OpenShift project

To setup a single OpenShift project:

1. Define the service account used to create workspace objects with the **`CHE_OPENSHIFT_SERVICEACCOUNTNAME`** variable.
2. To ensure this service account is visible to the CodeReady Workspaces server, put the service and the CodeReady Workspaces server in the same namespace.
3. Give the service account permissions to create and edit OpenShift resources.
4. If the developer needs to create an object outside of the service accounts bound namespace, give the service account cluster-admin rights by running this command:

```
$ oc adm policy add-cluster-role-to-user self-provisioner
system:serviceaccount:eclipse-che:che
```

In the command above, **`eclipse-che`** is the CodeReady Workspaces namespace.

6.3.2. Setting up a multi OpenShift project

1. To create workspace objects in different namespaces for each user, set the **`NULL_CHE_INFRA_OPENSHIFT_PROJECT`** variable to **`NULL`**.
2. To create resources on behalf of the currently logged-in user, use the user's OpenShift tokens.

6.4. HOW THE CODEREADY WORKSPACES SERVER USES PVCS AND PVS FOR STORAGE

CodeReady Workspaces server, Keycloak and PostgreSQL pods, and workspace pods use Persistent Volume Claims (PVCs), which are bound to the physical Persistent Volumes (PVs) with ReadWriteOnce access mode. When the deployment YAML files run, they define the CodeReady Workspaces PVCs. You can configure workspace PVC access mode and claim size with CodeReady Workspaces deployment environment variables.

6.4.1. Storage requirements for CodeReady Workspaces infrastructure

- CodeReady Workspaces server: 1 GB to store logs and initial workspace stacks.
- Keycloak: 2 PVCs, 1 GB each to store logs and Keycloak data.
- PostgreSQL: 1 GB PVC to store database.

6.4.2. Storage strategies for workspaces

The workspace PVC strategy is configurable:

strategy	details	pros	cons
unique (default)	One PVC per workspace volume	Storage isolation	An undefined number of PVs is required

strategy	details	pros	cons
common	One PVC for all workspaces in one OpenShift Project Sub-paths pre-created	Easy to manage and control storage	Workspaces must be in a separate OpenShift Project if PV does not support ReadWriteMany (RWX) access mode
per-workspace	One PVC for one workspace Sub-paths pre-created	Easy to manage and control storage	Workspace containers must all be in one pod if PV does not support ReadWriteMany (RWX) access mode

6.4.3. Unique PVC strategy

To define the unique strategy, set **`CHE_INFRA_KUBERNETES_PVC_STRATEGY`** to **`unique`**.

Every workspace gets its own PVC, which means a workspace PVC is created when a workspace starts for the first time. Workspace PVC is deleted when a corresponding workspace is deleted.

6.4.4. Common PVC Strategy

6.4.4.1. How the common PVC strategy works

All workspaces use the same PVC to store data declared in their volumes (projects and workspace logs by default and whatever additional volumes that a user can define.)

A PV that is bound to PVC **`che-claim-workspace`** will have the following structure:

```
pv0001
  workspaceid1
  workspaceid2
  workspaceidn
    che-logs projects <volume1> <volume2>
```

Volumes can be anything that a user defines as volumes for workspace machines. The volume name is equal to the directory name in **`${PV}/${ws-id}`**.

When a workspace is deleted, a corresponding subdirectory (**`${ws-id}`**) is deleted in the PV directory.

6.4.4.2. Enabling a common strategy

If you have already deployed CodeReady Workspaces with unique strategy, set the **`CHE_INFRA_KUBERNETES_PVC_STRATEGY`** variable to **`common`** in **`dc/che`**.

If applying the **`che-server-template.yaml`** configuration, pass **`-p CHE_INFRA_KUBERNETES_PVC_STRATEGY=common`** to the **`oc new-app`** command.

6.4.4.3. Restrictions on using common PVC strategy

When a common strategy is used and a workspace PVC access mode is ReadWriteOnce (RWO), only one OpenShift node can simultaneously use PVC. If there are several nodes, you can use a common strategy, but the workspace PVC access mode is ReadWriteMany (RWM). Multiple nodes can use this PVC simultaneously.

To change the access mode for workspace PVCs, pass the **`CHE_INFRA_KUBERNETES_PVC_ACCESS_MODE=ReadWriteMany`** environment variable to CodeReady Workspaces deployment either when initially deploying CodeReady Workspaces or through the CodeReady Workspaces deployment update.

Another restriction is that only pods in the same namespace can use the same PVC. The **`CHE_INFRA_KUBERNETES_PROJECT`** environment variable should not be empty. It should be either the CodeReady Workspaces server namespace where objects can be created with the CodeReady Workspaces service account (SA) or a dedicated namespace where a token or a username and password need to be used.

6.4.5. Per workspace PVC strategy

To define the unique strategy, set **`CHE_INFRA_KUBERNETES_PVC_STRATEGY`** to **`per-workspace`**.

6.4.5.1. How the per workspace PVC strategy works

It works similarly to the common PVC strategy. The only difference is that all workspace volumes (but not all workspaces) use the same PVC to store data (projects and workspace logs by default and any additional volumes that a user can define).

6.5. UPDATING YOUR CODEREADY WORKSPACES DEPLOYMENT

To update CodeReady Workspaces deployment:

1. Change the image tag:
   You can change the image tag in one of the following ways:

   - On the command line, edit the image tag by running:

     ```
     $ oc edit dc/che
     ```

   - In the OpenShift web console, edit the **`image:tag`** line in the YAML file in **Deployments**

   - Using the Docker service:

     ```
     $ oc set image dc/che che=eclipse/che-server:${VERSION} --
     source=docker
     ```

2. Update Keycloak and PostgreSQL deployments (optional):

   - Run the **`eclipse/che-keycloak`** command.

   - Run the **`eclipse/che-postgres`** command.
     You can get the list of available versions at CodeReady Workspaces GitHub page.

3. Change the pull policy (optional):
   To change the pull policy, do one of the following:

- Add **`--set cheImagePullPolicy=IfNotPresent`** to the CodeReady Workspaces deployment.
- Manually edit **`dc/che`** after deployment.

The default pull policy is **`Always`**. The default tag is **`nightly`**. This tag sets the image pull policy to **`Always`** and triggers a new deployment with a newer image, if available.

6.6. SCALABILITY

To run more workspaces, add more nodes to your OpenShift cluster. An error message is returned when the system is out of resources.

6.7. GDPR

To delete data or request the administrator to delete data, run this command with the user or administrator token:

```
$ curl -X DELETE http://che-server/api/user/{id}
```

6.8. DEBUG MODE

To run CodeReady Workspaces Server in debug mode, set the following environment variable in the CodeReady Workspaces deployment to **`true`** (default is **`false`**):

`CHE_DEBUG_SERVER=true`

6.9. PRIVATE DOCKER REGISTRIES

See OpenShift documentation.

6.10. CODEREADY WORKSPACES SERVER LOGS

Logs are persisted in a PV .The PVC **`che-data-volume`** is created and bound to a PV after CodeReady Workspaces deploys to OpenShift.

To retrieve logs, do one of the following:

- Run the **`oc get log dc/che`** command.
- Run the **`oc describe pvc che-data-claim`** command to find the PV. Next, run the **`oc describe pv $pvName`** command with the PV to get a local path with the logs directory. Be careful with permissions for that directory, since once changed, CodeReady Workspaces server will not be able to write to a file.
- In the OpenShift web console, select **Pods** > **che-pod** > **Logs**.

It is also possible to configure CodeReady Workspaces master not to store logs, but produce JSON encoded logs to output instead. It may be used to collect logs by systems such as Logstash. To configure JSON logging instead of plain text environment variable **`CHE_LOGS_APPENDERS_IMPL`** should have value **`json`**.

6.11. WORKSPACE LOGS

Workspace logs are stored in an PV bound to **`che-claim-workspace`** PVC. Workspace logs include logs from workspace agent, bootstrapper and other agents if applicable.

6.12. CODEREADY WORKSPACES MASTER STATES

The CodeReady Workspaces master has three possible states:

- **`RUNNING`**
- **`PREPARING_TO_SHUTDOWN`**
- **`READY_TO_SHUTDOWN`**

The **`PREPARING_TO_SHUTDOWN`** state means that no new workspace startups are allowed. This situation can cause two different results:

- If your infrastructure does not support workspace recovery, all running workspaces are forcibly stopped.
- If your infrastructure does support workspace recovery, any workspaces that are currently starting or stopping is allowed to finish that process. Running workspaces do not stop.

For those that did not stop, automatic fallback to the shutdown with full workspaces stopping will be performed.

If you want a full shutdown with workspaces stopped, you can request this by using the **`shutdown=true`** parameter. When preparation process is finished, the **`READY_TO_SHUTDOWN`** state is set which allows to stop current CodeReady Workspaces master instance.

6.13. WORKSPACE TERMINATION GRACE PERIOD

The default grace termination period of OpenShift workspace pods is **`0`**. This setting terminates pods almost instantly and significantly decreases the time required for stopping a workspace.

To increase the grace termination period, use the following environment variable: **`CHE_INFRA_KUBERNETES_POD_TERMINATION__GRACE__PERIOD__SEC`**.

IMPORTANT

If the **`terminationGracePeriodSeconds`** variable is explicitly set in the OpenShift recipe, the **`CHE_INFRA_KUBERNETES_POD_TERMINATION__GRACE__PERIOD__SEC`** environment variable does not override the recipe.

6.14. AUTO-STOPPING A WORKSPACE WHEN ITS PODS ARE REMOVED

CodeReady Workspaces Server includes a job that automatically stops workspace runtimes if their pods have been terminated. Pods are terminated when, for example, users remove them from the OpenShift console, administrators terminate them to prevent misuse, or an infrastructure node crashes.

The job is disabled by default to avoid problems in configurations where CodeReady Workspaces Server cannot interact with the Kubernetes API without user intervention.

The job cannot function with the following CodeReady Workspaces Server configuration:

- CodeReady Workspaces Server communicates with the Kubernetes API using a token from the OAuth provider.

The job can function with the following CodeReady Workspaces Server configurations:

- Workspaces objects are created in the same namespace where CodeReady Workspaces Server is located.
- The **cluster-admin** service account token is mounted to the CodeReady Workspaces Server pod.

To enable the job, set the **`CHE_INFRA_KUBERNETES_RUNTIMES__CONSISTENCY__CHECK__PERIOD__MIN`** environment variable to contain a value greater than **`0`**. The value is the time period in minutes between checks for runtimes without pods.

6.15. UPDATING CODEREADY WORKSPACES WITHOUT STOPPING ACTIVE WORKSPACES

The differences between a Recreate update and a Rolling update:

Recreate update	Rolling update
CodeReady Workspaces downtime	No CodeReady Workspaces downtime
-	New deployment starts in parallel and traffic is hot-switched

6.15.1. Performing a recreate update

To perform a recreate update:

- Ensure that the new master version is fully API compatible with the old workspace agent version.
- Set the deployment update strategy to Recreate
- Make POST request to the **`/api/system/stop`** api to start WS master suspend. This means that all new attempts to start workspaces will be refused, and all current starts and stops will be finished. Note that this method requires system admin credentials.
- Make periodical **`GET`** requests to the **`/api/system/state`** API, until it returns the **`READY_TO_SHUTDOWN`** state. Also, you can check for "System is ready to shutdown" in the server logs.
- Perform new deploy.

6.15.2. Performing a rolling update

To perform a rolling update:

- Ensure that the new master is fully API compatible with the old ws agent versions, as well as database compatibility. It is impossible to use database migrations on this update mode.
- Set the deployment update strategy set to Rolling.
- Ensure **`terminationGracePeriodSeconds`** deployment parameter has enough value (see details below).
- Press **Deploy** button or execute **`oc rollout latest che`** from cli client.

6.15.2.1. Known issues

- Workspaces may fallback to the stopped state when they are started five to thirty seconds before the network traffic are switched to the new pod. This happens when the bootstrappers use the CodeReady Workspaces server route URL for notifying the CodeReady Workspaces Server that bootstrapping is done. Since traffic is already switched to the new CodeReady Workspaces server, the old CodeReady Workspaces server cannot get the bootstrapper's report and fails to start after the waiting timeout is reached. If the old CodeReady Workspaces server is killed before this timeout, the workspaces can be stuck in the **`STARTING`** state. The **`terminationGracePeriodSeconds`** parameter must define enough time to cover the workspace start timeout, which is eight minutes plus some additional time. Typically, setting **`terminationGracePeriodSeconds`** to 540 sec is enough to cover all timeouts.
- Users may experience problems with websocket reconnections or missed events published by WebSocket connection when a workspace is **`STARTED`** but dashboard displays that it is **`STARTING`**. In this case, you need to reload the page to restore connections and the actual workspace states.

6.15.3. Updating with database migrations or API incompatibility

If new version of CodeReady Workspaces server contains some DB migrations, but there is still API compatibility between old and new version, recreate update type may be used, without stopping running workspaces.

API incompatible versions should be updated with full workspaces stop. It means that **`/api/system/stop?shutdown=true`** must be called prior to update.

6.16. DELETING DEPLOYMENTS

The fastest way to completely delete CodeReady Workspaces and its infrastructure components is to delete the project and namespace.

To delete CodeReady Workspaces and components:

```
$ oc delete namespace che
```

You can use selectors to delete particular deployments and associated objects.

To remove all CodeReady Workspaces server related objects:

```
$ oc delete all -l=app=che
```

To remove all Keycloak related objects:

```
$ oc delete all -l=app=keycloak
```

To remove all PostgreSQL-related objects:

```
$ oc delete all -l=app=postgres
```

PVCs, service accounts and role bindings should be deleted separately because **`oc delete all`** does not delete them.

To delete CodeReady Workspaces server PVC, ServiceAccount and RoleBinding:

```
$ oc delete sa -l=app=che
$ oc delete rolebinding -l=app=che
```

To delete Keycloak and PostgreSQL PVCs:

```
$ oc delete pvc -l=app=keycloak
$ oc delete pvc -l=app=postgres
```

6.17. MONITORING CODEREADY WORKSPACES MASTER SERVER

Master server emits metrics in Prometheus format by default on port **`8087`** of the CodeReady Workspaces server host (this can be customized by the **`che.metrics.port`**configuration property).

You can configure your own Prometheus deployment to scrape the metrics (as per convention, the metrics are published on the **`<CHE_HOST>:8087/metrics`** endpoint).

The CodeReady Workspaces's Helm chart can optionally install Prometheus and Grafana servers preconfigured to collect the metrics of the CodeReady Workspaces server. When you set the **`global.metricsEnabled`** value to **`true`** when installing CodeReady Workspaces's Helm chart, Prometheus and Grafana servers are automatically deployed. The servers are accessible on **`prometheus-<CHE_NAMESPACE>.domain`** or **`grafana-<CHE_NAMESPACE>.domain`** domains respectively. The Grafana server is preconfigured with a sample dashboard showing the memory usage of the CodeReady Workspaces server. You can log in to the Grafana server using the predefined username **`admin`** with the default password **`admin`**.

6.18. CREATING WORKSPACE OBJECTS IN PERSONAL NAMESPACES

You can register the OpenShift server as an identity provider when CodeReady Workspaces is installed in multi-user mode. This allows you to create workspace objects in the OpenShift namespace of the user that is logged in CodeReady Workspaces through Keycloak.

To create a workspace object in the namespace of the user that is logged into CodeReady Workspaces:

- Register, inside Keycloak, an OpenShift identity provider that points to the OpenShift console of the cluster.
- Configure CodeReady Workspaces to use the Keycloak identity provider to retrieve the OpenShift tokens of the CodeReady Workspaces users.

Every workspace action such as start or stop creates an OpenShift resource in the OpenShift user account. A notification message displays which allows you to link the Keycloak account to your OpenShift user account.

But for non-interactive workspace actions, such as workspace stop on idling or CodeReady Workspaces server shutdown, the dedicated OpenShift account configured for the Kubernetes infrastructure is used. See Setting up the project workspace for more information.

To easily install CodeReady Workspaces on OpenShift with this feature enabled, see this section for Minishift and this one for OCP

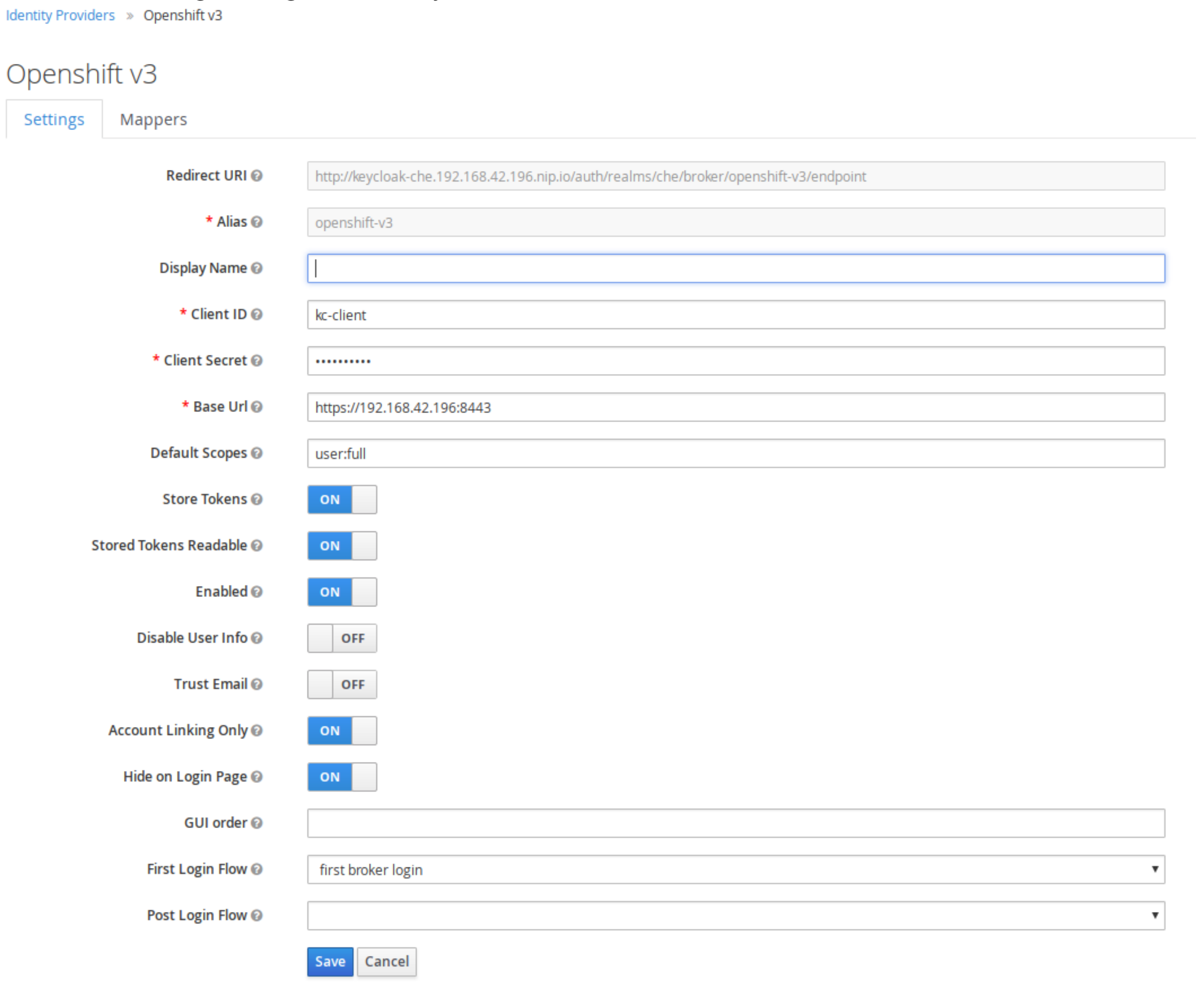
6.19. OPENSHIFT IDENTITY PROVIDER REGISTRATION

NOTE

Cluster-wide administrator rights are required to add an OAuth client.

To add the OpenShift identity provider:

1. Use the following settings in the Keycloak administration console:



Identity Providers » Openshift v3

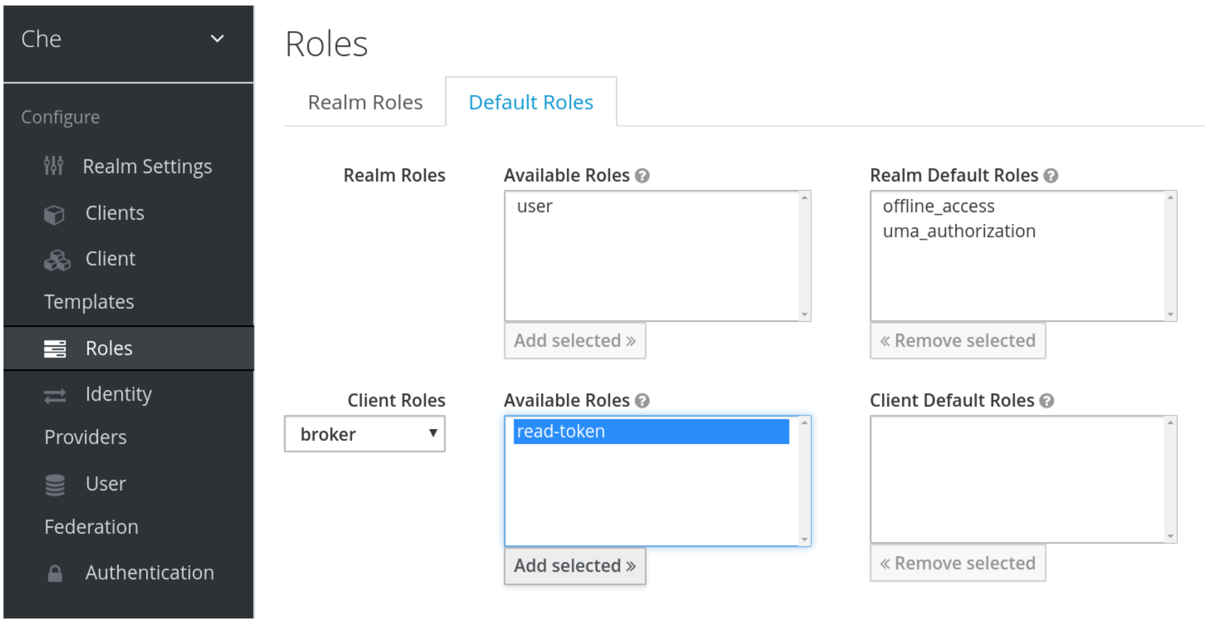
Openshift v3

Settings | Mappers

Redirect URI	http://keycloak-che.192.168.42.196.nip.io/auth/realms/che/broker/openshift-v3/endpoint
* Alias	openshift-v3
Display Name	
* Client ID	kc-client
* Client Secret	••••••••••
* Base Url	https://192.168.42.196:8443
Default Scopes	user:full
Store Tokens	ON
Stored Tokens Readable	ON
Enabled	ON
Disable User Info	OFF
Trust Email	OFF
Account Linking Only	ON
Hide on Login Page	ON
GUI order	
First Login Flow	first broker login
Post Login Flow	

The **`Base URL`** is the URL of the OpenShift console.

2. Add a default read-token role.



3. Declare the identity provider as an OAuth client inside OpenShift with the following commands:

```
$ oc create -f <(echo '
apiVersion: v1
kind: OAuthClient
metadata:
  name: kc-client
secret: "<value set for the 'Client Secret' field in step 1>"
redirectURIs:
  - "<value provided in the 'Redirect URI' field in step 1>"
grantMethod: prompt
')
```

See Keycloak documentation for more information on the Keycloak OpenShift identity provider.

6.20. CONFIGURING CODEREADY WORKSPACES

To configure CodeReady Workspaces deployment:

- Set the **CHE_INFRA_OPENSHIFT_PROJECT** variable to **NULL** to ensure a new distinct OpenShift namespace is created for every workspace that is started.
- Set the **CHE_INFRA_OPENSHIFT_OAUTH__IDENTITY__PROVIDER** variable to the alias of the OpenShift identity provider specified in step 1 of its registration in Keycloak. The default value is **openshift-v3**.

6.21. PROVIDING THE OPENSHIFT CERTIFICATE TO KEYCLOAK

If the certificate used by the OpenShift console is self-signed or is not trusted, then by default the Keycloak will not be able to contact the OpenShift console to retrieve linked tokens.

Keycloak cannot contact the OpenShift console to retrieve linked tokens when the certificate used by the OpenShift console is self-signed or is not trusted.

When the certificate is self-signed or is not trusted, use the **OPENSHIFT_IDENTITY_PROVIDER_CERTIFICATE** variable to pass the OpenShift console certificate to the Keycloak deployment. This will enable the Keycloak server to add the certificate to the list of trusted certificates. The environment variable refers to a secret that contains the certificate.

CHAPTER 7. MANAGING USERS

7.1. AUTHORIZATION AND USER MANAGEMENT

CodeReady Workspaces uses Keycloak to create, import, manage, delete, and authenticate users. Keycloak uses built-in authentication mechanisms and user storage. It can use third-party identity management systems to create and authenticate users. CodeReady Workspaces requires a Keycloak token when you request access to CodeReady Workspaces resources.

Local users and imported federation users must have an email address in their profile.

The default Keycloak credentials are `admin:admin`. You can use the `admin:admin` credentials when logging into CodeReady Workspaces for the first time. It has system privileges.

To find your Keycloak URL:

If CodeReady Workspaces is deployed on OpenShift:

- Go to the OpenShift web console and navigate to the **Keycloak** namespace.

7.2. CONFIGURING CODEREADY WORKSPACES TO WORK WITH KEYCLOAK

The deployment script ensures that Keycloak is properly configured when CodeReady Workspaces is deployed on OpenShift or installed on Docker. When the `che-public` client is created, the following fields are populated:

- **Valid Redirect URIs**: Use this URL to access CodeReady Workspaces.
- **Web Origins**

The following are common errors when configuring CodeReady Workspaces to work with Keycloak:

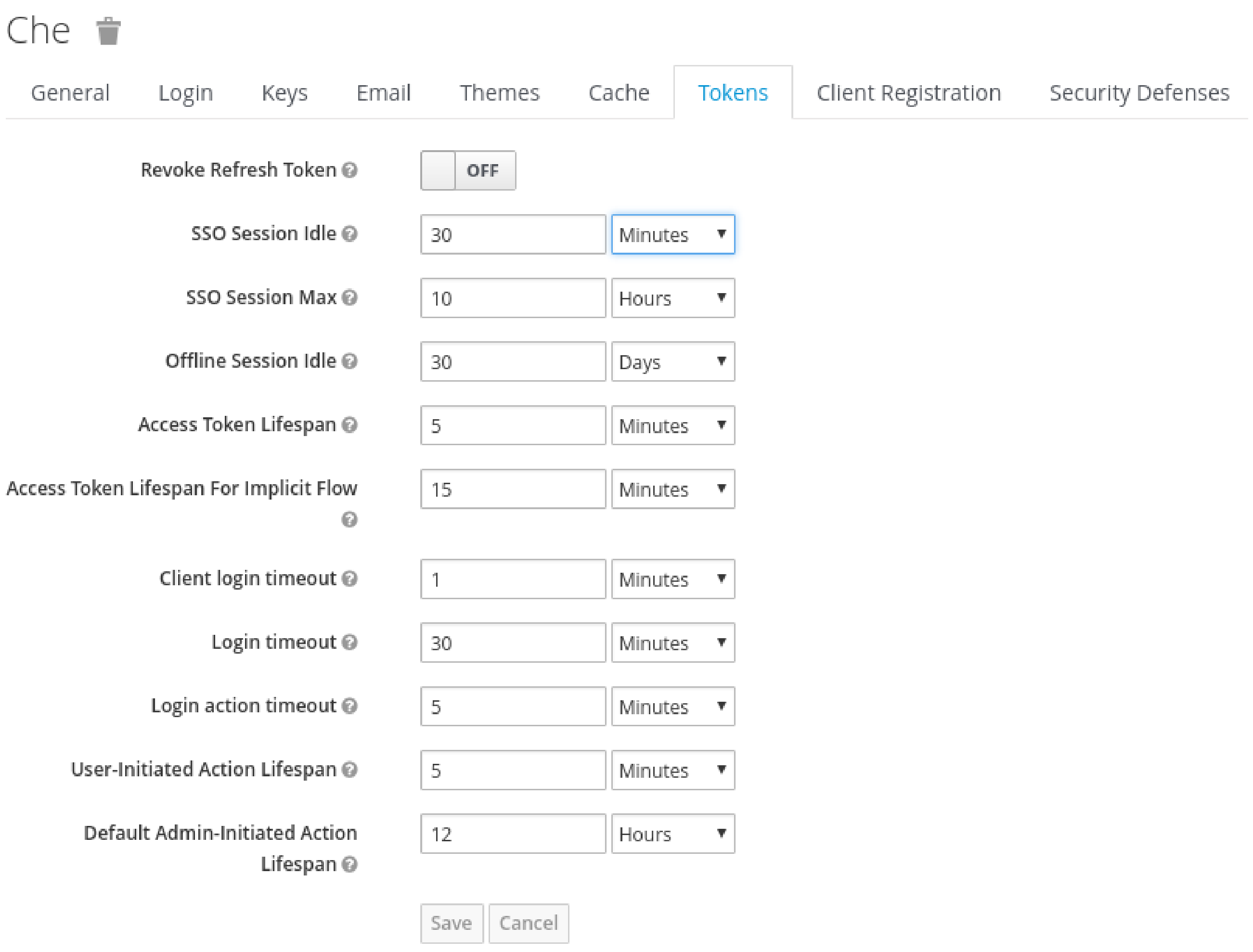
Invalid `redirectURI` error: occurs when you access CodeReady Workspaces at `myhost`, which is an alias, and your original `CHE_HOST` is `1.1.1.1`. If this error occurs, go to the Keycloak administration console and ensure that the valid redirect URIs are configured.

CORS error: occurs when you have an invalid web origin

7.3. CONFIGURING KEYCLOAK TOKENS

A user token expires after 30 minutes by default.

You can change the following Keycloak token settings:



Revoke Refresh Token	OFF	
SSO Session Idle	30	Minutes
SSO Session Max	10	Hours
Offline Session Idle	30	Days
Access Token Lifespan	5	Minutes
Access Token Lifespan For Implicit Flow	15	Minutes
Client login timeout	1	Minutes
Login timeout	30	Minutes
Login action timeout	5	Minutes
User-Initiated Action Lifespan	5	Minutes
Default Admin-Initiated Action Lifespan	12	Hours

7.4. SETTING UP USER FEDERATION

Keycloak federates external user databases and supports LDAP and Active Directory. You can test the connection and authenticate users before choosing a storage provider.

See the User storage federation page in Keycloak documentation to learn how to add a provider.

See the LDAP and Active Directory page in Keycloak documentation to specify multiple LDAP servers.

7.5. ENABLING AUTHENTICATION WITH SOCIAL ACCOUNTS AND BROKERING

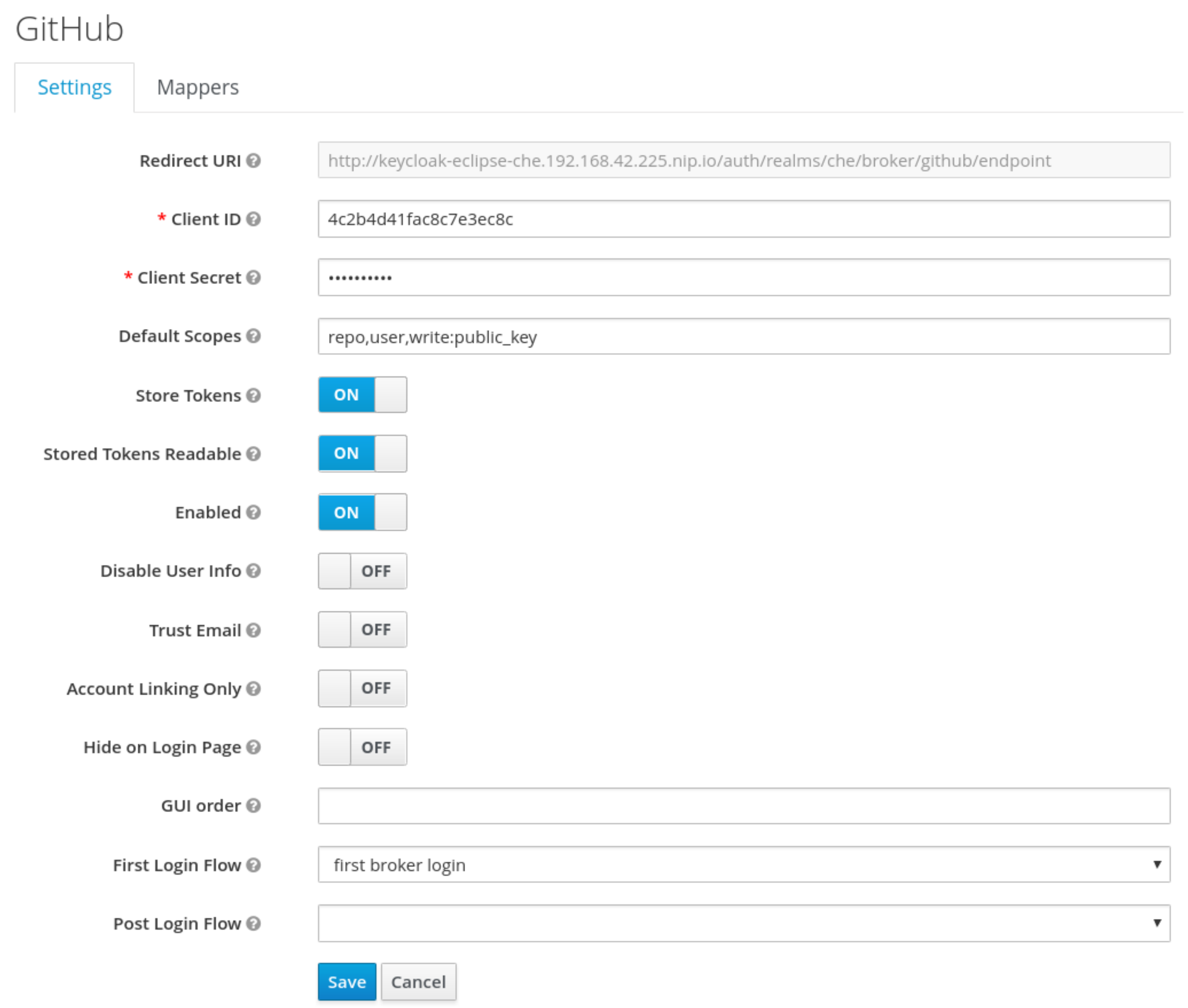
Keycloak provides built-in support for GitHub, OpenShift, and most common social networks such as Facebook and Twitter. See Instructions to enable Login with GitHub.

You can also enable the SSH key and upload it to the CodeReady Workspaces users’ GitHub accounts.

To enable this feature when you register a GitHub identity provider:

1. Set scope to **`repo,user,write:public_key`**.

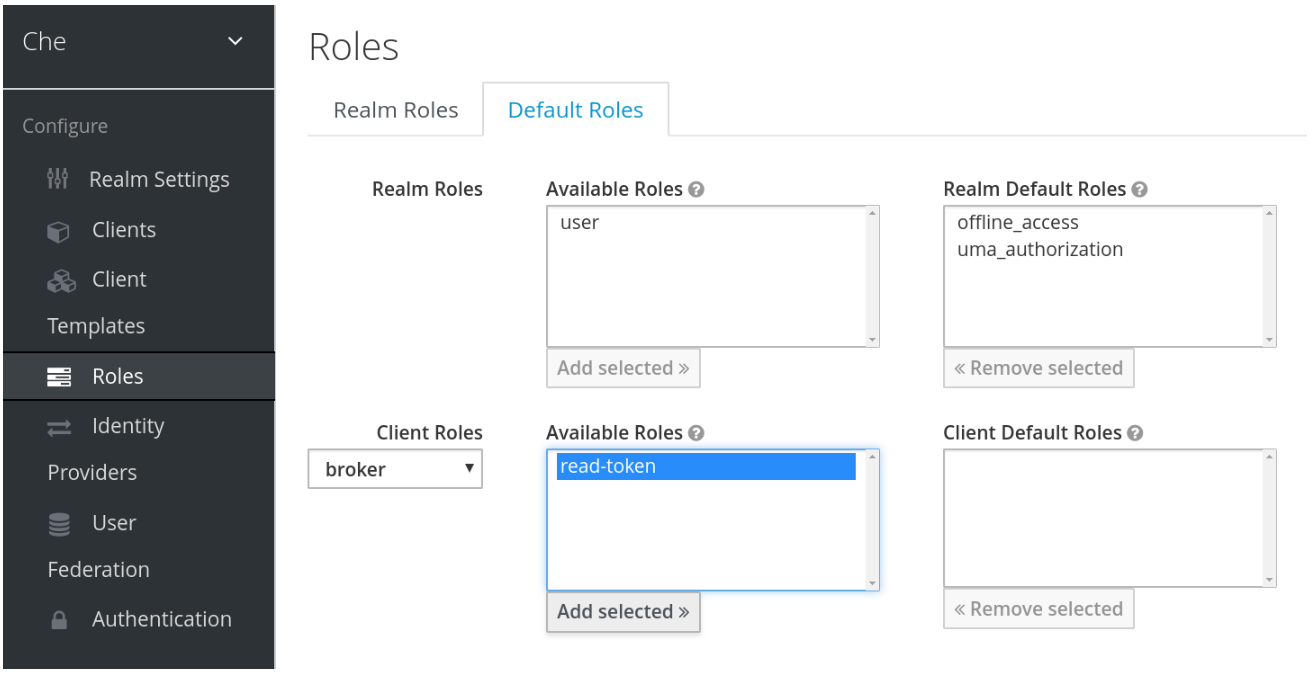
2. Set store tokens and stored tokens readable to **ON**.


GitHub

Settings | Mappers

Redirect URI	http://keycloak-eclipse-che.192.168.42.225.nip.io/auth/realms/che/broker/github/endpoint
* Client ID	4c2b4d41fac8c7e3ec8c
* Client Secret	••••••••••
Default Scopes	repo,user,write:public_key
Store Tokens	ON
Stored Tokens Readable	ON
Enabled	ON
Disable User Info	OFF
Trust Email	OFF
Account Linking Only	OFF
Hide on Login Page	OFF
GUI order	
First Login Flow	first broker login
Post Login Flow	

1. Add a default read-token role.



Roles

Realm Roles | Default Roles

Realm Roles	Available Roles: user	Realm Default Roles: offline_access, uma_authorization
Client Roles: broker	Available Roles: read-token	Client Default Roles:

This is the default **`delegated`** OAuth service mode for CodeReady Workspaces. You can configure the OAuth service mode with the property **`che.oauth.service_mode`**.

To use CodeReady Workspaces's OAuth Authenticator, set **`che.oauth.service_mode`** to **`embedded`** and use Instructions for single-user mode.

See SSH key management for more information.

7.6. USING PROTOCOL-BASED PROVIDERS

Keycloak supports SAML v2.0 and OpenID Connect v1.0 protocols. You can connect your identity provider systems if they support these protocols.

7.7. MANAGING USERS

You can add, delete, and edit users in the user interface. See: Keycloak User Management for more information.

7.8. CONFIGURING SMTP AND EMAIL NOTIFICATIONS

CodeReady Workspaces does not provide any pre-configured MTP servers.

To enable SMTP servers in Keycloak:

1. Go to **`che realm settings > Email`**.

2. Specify the host, port, username, and password.

CodeReady Workspaces uses the default theme for email templates for registration, email confirmation, password recovery, and failed login.

7.9. CODEREADY WORKSPACES AUTHENTICATION

7.9.1. Authentication on CodeReady Workspaces Master

7.9.1.1. OpenId

OpenId authentication on CodeReady Workspaces master, implies presence of an external OpenId provider and has 2 main steps:

1. Authenticate the user through the JWT token he brought or redirect him to login; (Authentication tokens should be send in a **`Authorization`** header. Also, in limited cases when it's not possible to use **`Authorization`** header, token can be send in **`token`** query parameter. An example of such exceptional case can be: OAuth authentification initialization, IDE shows javadoc in iframe where authentication must be initialized.)

2. Compose internal "subject" object which represents the current user inside of the CodeReady Workspaces master code.

At the time of writing the only supported/tested OpenId provider is Keycloak, so all examples/links will refer to this implementation.

The flow starts from the settings service where clients can find all the necessary URLs and properties of the OpenId provider such as **`jwks.endpoint`**, **`token.endpoint`**, **`logout.endpoint`**, **`realm.name`**, **`client_id`** etc returned. in JSON format.

Service class is **org.eclipse.che.multiuser.keycloak.server.KeycloakSettings**, and it is bound only in multi-user version of CodeReady Workspaces, so by its presence it is possible to detect if authentication enabled in current deployment or not.

Example output:

```
{
    "che.keycloak.token.endpoint":
"http://172.19.20.9:5050/auth/realms/che/protocol/openid-connect/token",
    "che.keycloak.profile.endpoint":
"http://172.19.20.9:5050/auth/realms/che/account",
    "che.keycloak.client_id": "che-public",
    "che.keycloak.auth_server_url": "http://172.19.20.9:5050/auth",
    "che.keycloak.password.endpoint":
"http://172.19.20.9:5050/auth/realms/che/account/password",
    "che.keycloak.logout.endpoint":
"http://172.19.20.9:5050/auth/realms/che/protocol/openid-connect/logout",
    "che.keycloak.realm": "che"
}
```

Also, this service allows to download JS client library to interact with provider. Service URL is **`<che.host>:<che.port>/api/keycloak/settings`** for retrieving settings JSON and **`<che.host>:<che.port>/api/keycloak/OIDCKeycloak.js`** for JS adapter library.

Next step is redirection of user to the appropriate provider's login page with all the necessary params like client_id, return redirection path etc. This can be basically done with any client library (JS or Java etc).

After user logged in on provider's side and client side code obtained and passed the JWT token, validation of it and creation of subject begins.

Verification of tokens signature occurs in the two main filters chain:

- **org.eclipse.che.multiuser.keycloak.server.KeycloakAuthenticationFilter** class. Token is extracted from **`Authorization`** header or **`token`** query param and tried to being parsed using public key retrieved from provider. In case of expired/invalid/malformed token, 403 error is sent to user. As noted above, usage of query parameter should be minimised as much as possible, since support of it may be limited/dropped at some point.

If validation was successful, token is passed to the

- **org.eclipse.che.multiuser.keycloak.server.KeycloakEnvironmentInitalizationFilter** filter in the parsed form. This filter simply extracts data from JWT token claims, creates user in the local DB if it is not yet present, and constructs subject object and sets it into per-request **`EnvironmentContext`** object which is statically accessible everywhere.

If the request was made using machine token only (e.g. from ws agent) then it is only one auth filter in the chain:

- **org.eclipse.che.multiuser.machine.authentication.server.MachineLoginFilter** - finds userId given token belongs to, than retrieves user instance and sets principal to the session.

Master-to-master requests are performed using **org.eclipse.che.multiuser.keycloak.server.KeycloakHttpJsonRequestFactory** which signs every request with the current subject token obtained from EnvironmentContext.

7.9.1.1.1. User Profile

Since keycloak may store user specific information (first/last name, phone number, job title etc), there is special implementation of the ProfileDao which can provide this data to consumers inside CodeReady Workspaces. Implementation is read-only, so no create/update operations are possible. Class is **org.eclipse.che.multiuser.keycloak.server.dao.KeycloakProfileDao**.

7.9.1.1.2. Obtaining Token From Keycloak

For the clients which cannot run JS or other type clients (like CLI or selenium tests), auth token may be requested directly from Keycloak. The simplest way to obtain Keycloak auth token, is to perform request to the token endpoint with username and password credentials. This request can be schematically described as following cURL request:

```
curl
    --data "grant_type=password&client_id=<client_name>&username=
<username>&password=<password>"

http://<keyckloak_host>:5050/auth/realms/<realm_name>/protocol/openid-
connect/token
```

Since the two main CodeReady Workspaces clients (IDE and Dashboard) utilizes native Keycloak js library, they’re using a customized Keycloak login page and somewhat more complicated authentication mechanism using **`grant_type=authorization_code`**. It’s a two step authentication: first step is login and obtaining authorization code, and second step is obtaining token using this code.

Example of correct token response:

```
{
   "access_token":"eyJhb...<rest of JWT token here>",
   "expires_in":300,
   "refresh_expires_in":1800,
   "refresh_token":"Nj0C...<rest of refresh token here>",
   "token_type":"bearer",
   "not-before-policy":0,
   "session_state":"14de1b98-8065-43e1-9536-43e7472250c9"
}
```

7.9.1.2. Other authentication implementations

If you want to adapt authentication implementation other than Keycloak, the following steps must be done:

- Write own or refactor existing info service which will provide list of configured provider endpoints to the clients;
- Write single or chain of filters to validate tokens, create user in CodeReady Workspaces DB and compose the Subject object;
- If the new auth provider supports OpenId protocol, OIDC JS client available at settings endpoint can be used as well since it is maximally decoupled of specific implementations.
- If the selected provider stores some additional data about user (first/last name, job title etc), it is a good idea to write an provider-specific ProfileDao implementation which will provide such kind of information.

7.9.1.3. OAuth

OAuth authentication part has 2 main flows - internal and external based on Keycloak brokering mechanism. So, there are 2 main OAuth API implementations - **org.eclipse.che.security.oauth.EmbeddedOAuthAPI** and **org.eclipse.che.multiuser.keycloak.server.oauth2.DelegatedOAuthAPI**.

They can be switched using **`che.oauth.service_mode=<embedded|delegated>`** configuration property.

Also, there is support of OAuth1 protocol can be found at **org.eclipse.che.security.oauth1** package.

The main REST endpoint in tha OAuth API is **org.eclipse.che.security.oauth.OAuthAuthenticationService**, which contains **`authenticate`** method to start OAuth authentication flow, **`callback`** method to process callbacks from provider, **`token`** to retrieve current user's oauth token, etc.

Those methods refer to the currently activated embedded/delegated OAuthAPI which is doing all the undercover stuff (finds appropriate authenticator, initializes the login process, user forwarding etc).

7.9.2. Authentication on CodeReady Workspaces Agents

Machines may contain services that must be protected with authentication, e.g. agents like workspace agent and terminal. For this purpose, machine authentication mechanism should be used. Machine tokens were introduced to avoid passing the Keycloak tokens to the machine side (which can be potentially insecure). Another reason is that Keycloak tokens may have relatively small lifetime and require periodical renewal/refresh which is hard to manage and keep in sync with same user session tokens on clients etc.

As agents cannot be queried using Keycloak token, there is only Machine Token option. Machine token can be also passed in header or query parameter.

7.9.2.1. Machine JWT Token

Machine token is JWT that contains the following information in its claim:

- **uid** - id of user who owns this token
- **uname** - name of user who owns this token
- **wsid** - id of a workspace which can be queried with this token

Each user is provided with unique personal token for each workspace.

The structure of token and the signature are different to Keycloak and have the following view:

```
# Header
{
  "alg": "RS512",
  "kind": "machine_token"
}
# Payload
{
  "wsid": "workspacekrh99xjenek3h571",
  "uid": "b07e3a58-ed50-4a6e-be17-fcf49ff8b242",
  "uname": "john",
  "jti": "06c73349-2242-45f8-a94c-722e081bb6fd"
}
# Signature
{
  "value": "RSASHA512(base64UrlEncode(header) + . +
base64UrlEncode(payload))"
}
```

The algorithm that is used for signing machine tokens is **`SHA-512`** and it’s not configurable for now. Also, there is no public service that distributes the public part of the key pair with which the token was signed. But in each machine, there must be environment variables that contains key value. So, agents can verify machine JWT token using the following environment variables:

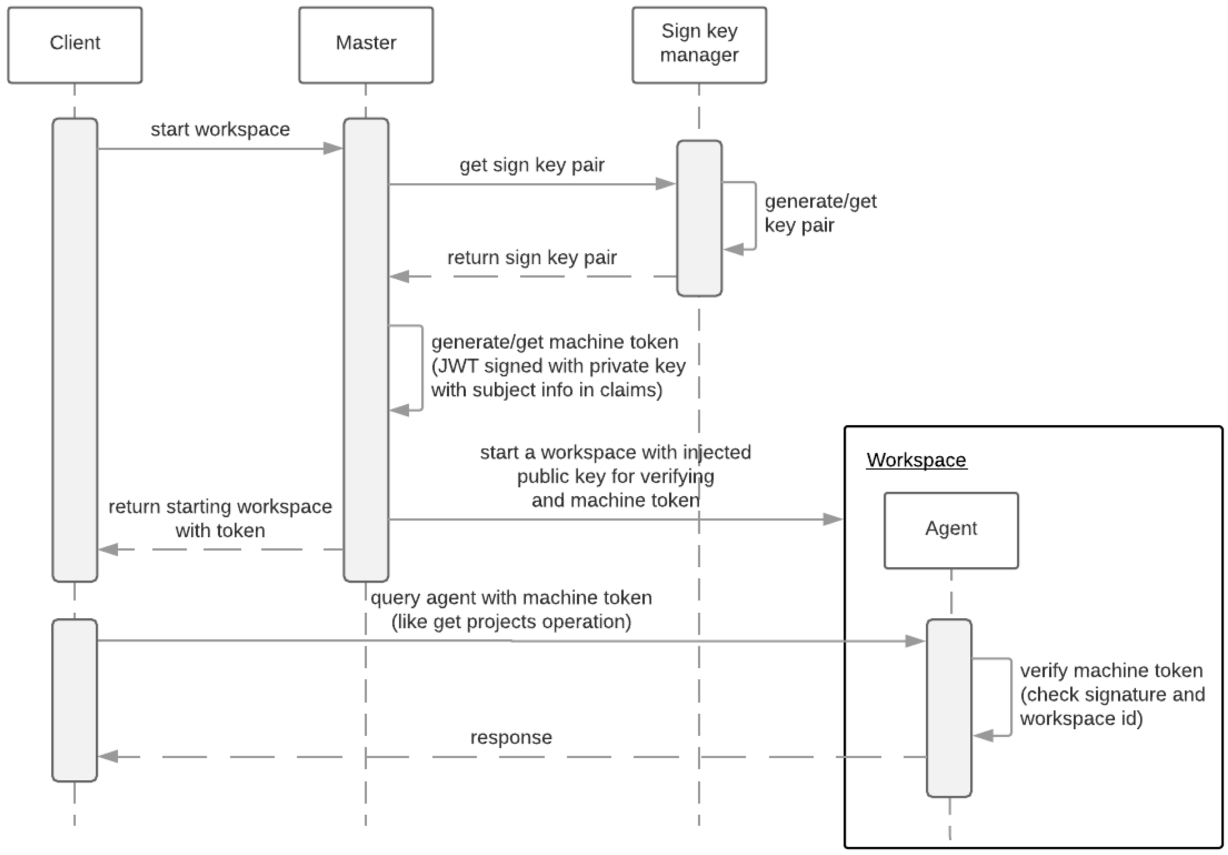
- **`CHE_MACHINE_AUTH_SIGNATURE__ALGORITHM`** - contains information about the algorithm which the token was signed
- **`CHE_MACHINE_AUTH_SIGNATURE__PUBLIC__KEY`** - contains public key value encoded in Base64

It’s all that is needed for verifying machine token inside of machine. To make sure that specified token is related to current workspace, it is needed to fetch **`wsid`** from JWT token claims and compare it with **`CHE_WORKSPACE_ID`** environment variable.

Also, if agents need to query CodeReady Workspaces Master they can use machine token provided in **`CHE_MACHINE_TOKEN`** environment, actually it is token of user who starts a workspace.

7.9.2.2. Authentication schema

The way how CodeReady Workspaces master interacts with agents with enabled authentication mechanism is the following:


Client
Master
Sign key manager
start workspace
get sign key pair
generate/get key pair
return sign key pair
generate/get machine token (JWT signed with private key with subject info in claims)
start a workspace with injected public key for verifying and machine token
Workspace
Agent
return starting workspace with token
query agent with machine token (like get projects operation)
verify machine token (check signature and workspace id)
response

Machine token verification on agents is done by the following components:

- **org.eclipse.che.multiuser.machine.authentication.agent.MachineLoginFilter** - doing basically the same as the appropriate filter on a master, the only thing that is different it's a way how agent obtains the public signature part. The public key for the signature check is placed in a machine environment, with algorithm description.

- **auth.auth.go** - the entry point for all request that is proceeding on go agents side, the logic of token verification is similar with MachineLoginFilter.

7.9.2.3. Obtaining Machine Token

A machine token is provided for users in runtime object. It can be fetched by using get workspace by key (id or namespace/name) method which path equals to `/api/workspace/<workspace_key>`. The machine token will be placed in `runtime.machineToken` field.

7.9.3. Using Swagger or REST Clients

User's Keycloak token is used to execute queries to secured API on his behalf through REST clients. A valid token must be attached as request header or query parameter - `?token=$token`. CodeReady Workspaces Swagger can be accessed from `http://che_host:8080/swagger`. A user must be signed-in through Keycloak so that access token is included in request headers.

By default, swagger loads `swagger.json` from CodeReady Workspaces master.

To work with WS Agent, a URL to its `swagger.json` should be provided. It can be retrieved from Workspace Runtime, by getting URL to WS Agent server endpoint and adding `api/docs/swagger.json` to it. Also, to authenticate on WS Agent API, user must include Machine

Token, which can be found in Workspace Runtime as well.

To use Swagger for a workspace agent, user must do following steps:

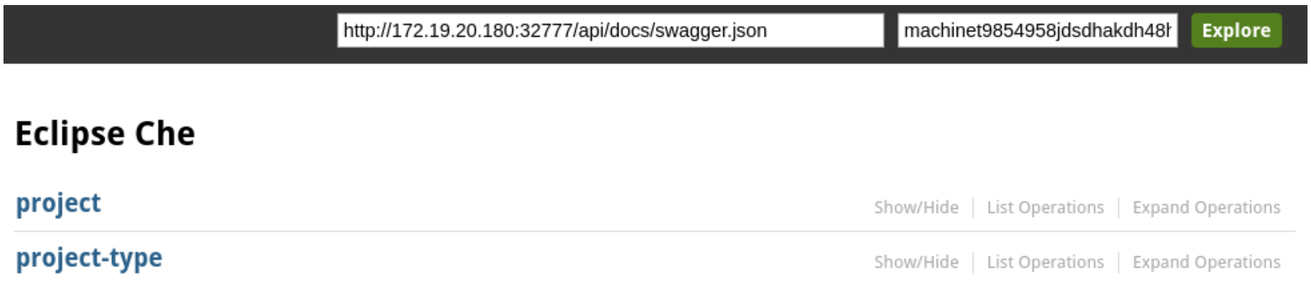
- get workspace object with runtimes fetched (using **`/api/workspace/<workspace_key>`** service)
- get WS agent API endpoint URL, and add a path to its **`swagger.json`** (e.g. `http://<che_host>:<machine_port>/api/docs/swagger.json` for Docker or `http://<ws-agent-route>/api/docs/swagger.json` for OpenShift). Put it in the upper bar **`URL`** field:

```
"wsagent/http": {
  "url": "http://172.19.20.180:32777/api",
  "attributes": {},
  "status": "RUNNING"
}
```

- get machine token from **`runtime.machineToken`** field, and put it in the upper bar **`token`** field

```
"machineToken":
"eyJhbGciOiJSUzUxMiIsImtpbmQiOiJtYWNoaW5lX3Rva2VuIn0.eyJ3c2lkIjoid29ya3NwY
WNlMzEiLCJ1aWQiOiJ1c2VyMTMiLCJ1bmFtZSI6InRlc3RVc2VyIiwianRpIjoiOTAwYTUwNWY
tYWY4ZS00MWQxLWFhYzktMTFkOGI5OTA5Y2QxIn0.UwU7NDzqnHxTr4vu8UqjZ7-
cjIfQBY4gP70Nqxkwfx8EsPfZMpoHGPt8bfqLWVWkpp3OacQVaswAOMOG9Uc9FtLnQWnup_6vv
yMo6gchZ1lTZFJMVHIw9RnSJAGFl98adWe3NqE_DdM02PyHb23MoHqE_xd8z3eFhngyaMImhc4
",
```

- click Explore to load Swagger for WS Agent



7.10. PERMISSIONS

7.10.1. Overview

Permissions are used to control the actions of users and establish a security model. You can control resources managed by CodeReady Workspaces and allow certain actions by assigning permissions to users.

Permissions can be applied to the following:

- Workspace
- Organization
- Stack

- System

7.10.2. Workspace permissions

The user who creates a workspace is the workspace owner. The workspace owner has the following permissions by default: **`read`**, **`use`**, **`run`**, **`configure`**, **`setPermissions`**, and **`delete`**. Workspace owners invite users into the workspace and control workspace permissions for each user.

The following permissions are associated with workspaces:

Permission	Description
read	Allows reading the workspace configuration.
use	Allows using a workspace and interacting with it.
run	Allows starting and stopping a workspace.
configure	Allows defining and changing the workspace configuration.
setPermissions	Allows updating the workspace permissions for other users.
delete	Allows deleting the workspace.

7.10.3. Organization permissions

An organization is a named set of users.

The following permissions are applicable to organizations:

Permission	Description
update	Allows editing of the organization settings and information.
delete	Allows deleting an organization.
manageSuborganizations	Allows creating and managing sub-organizations.
manageResources	Allows redistribution of an organization’s resources and defining the resource limits.
manageWorkspaces	Allows creating and managing all the organization’s workspaces.
setPermissions	Allows adding and removing users and updating their permissions.

7.10.4. System permissions

System permissions control aspects of the whole CodeReady Workspaces installation.

The following permissions are applicable to the organization:

Permission	Description
manageSystem	Allows control of the system, workspaces, and organizations.
setPermissions	Allows updating the permissions for users on the system.
manageUsers	Allows creating and managing users.
monitorSystem	Allows for accessing endpoints used for monitoring the state of the server.

All system permissions will be granted to the administration user configured with the \`CHE_SYSTEM_ADMIN__NAME\` property (the default is **`admin`**). This happens at CodeReady Workspaces Server start. If the user is not present in the CodeReady Workspaces user database, it happens after the user's login.

7.10.5. manageSystem permission

Users with the **`manageSystem`** permission have access to the following services:

Path	HTTP Method	Description
`/resource/free/`	GET	Get free resource limits
`/resource/free/{accountId}`	GET	Get free resource limits for given account
`/resource/free/{accountId}`	POST	Edit free resource limit for given account
`/resource/free/{accountId}`	DELETE	Remove free resource limit for given account
`/installer/`	POST	Add installer to the registry
`/installer/{key}`	PUT	Update installer in the registry
`/installer/{key}`	DELETE	Remove installer from the registry
`/logger/`	GET	Get logging configurations in CodeReady Workspaces Server

Path	HTTP Method	Description
`/logger/{name}`	GET	Get configurations of logger by its name in CodeReady Workspaces Server
`/logger/{name}`	PUT	Create logging in CodeReady Workspaces Server
`/logger/{name}`	POST	Edit logging in CodeReady Workspaces Server
`/resource/{accountId}/details`	GET	Get detailed information about resources for given account
`/system/stop`	POST	Shutdown all system services, prepare CodeReady Workspaces to stop
`/stacks`	All methods	All Stack service methods

7.10.6. monitorSystem permission

Users with the **`monitorSystem`** permission have access to the following services:

Path	HTTP Method	Description
`/activity`	GET	Get workspaces in certain state for a certain amount of time

7.10.7. Super-privileged mode

The **`manageSystem`** permission can be extended to provide a super-privileged mode. This allows the user to perform advanced actions on any resources managed by the system. You can read and stop any workspaces with the **`manageSystem`** permission and assign permissions to users as needed.

The super-privileged mode is disabled by default. You can change to super-privileged mode by configuring the `CHE_SYSTEM_SUPER__PRIVILEGED__MODE` variable to **`true`** in the **`che.env`** file.

List of services that are enabled for users with **`manageSystems`** permissions and with super-privileged mode on:

Path	HTTP Method	Description
`/workspace/namespace/{namespace:.*}`	GET	Get all workspaces for given namespace.
`/workspace/{id}`	DELETE	Stop workspace

Path	HTTP Method	Description
`/workspace/{key:.*}`	GET	Get workspace by key
`/organization/resource/{suborganizationId}/cap`	GET	Get resource cap for given organization
`/organization/resource/{suborganizationId}/cap`	POST	Set resource cap for given organization
`/organization/{parent}/organizations`	GET	Get child organizations
`/organization`	GET	Get user's organizations

7.10.8. Stack permissions

A stack is a runtime configuration for a workspace. See stack definition for more information on stacks.

The following permissions are applicable to stacks:

Permission	Description
search	Allows searching of the stacks.
read	Allows reading of the stack configuration.
update	Allows updating of the stack configuration.
delete	Allows deleting of the stack.
setPermissions	Allows managing permissions for the stack.

7.10.9. Permissions API

All permissions can be managed using the provided REST API. The APIs are documented using Swagger at **`[{host}/swagger/#!/permissions]`**.

7.10.10. Listing permissions

To list the permissions that apply to a specific resources, run this command:

```
$ GET /permissions
```

The **`domain`** values are:

Domain
`system`
`organization`
`workspace`
`stack`

> **NOTE**
>
> **`domain`** is optional. In this case, the API returns all possible permissions for all domains.

7.10.11. Listing permissions for a user

To list the permissions that apply to a user, run this command:

```
$ GET /permissions/{domain}
```

The **`domain`** values are:

Domain
`system`
`organization`
`workspace`
`stack`

7.10.12. Listing permissions for all users

> **NOTE**
>
> You must have sufficient permissions to see this information.

To list the permissions that apply to all users, run this command:

```
GET /permissions/{domain}/all
```

The **`domain`** values are:

Domain
system
organization
workspace
stack

7.10.13. Assigning permissions

To assign permissions to a resource, run this command:

`POST /permissions`

The **`domain`** values are:

Domain
system
organization
workspace
stack

The following is a message **body** that requests permissions for a user with a **`userID`** to a workspace with a **`workspaceID`**:

```
{
  "actions": [
    "read",
    "use",
    "run",
    "configure",
    "setPermissions"
  ],
  "userId": "userID",
  "domainId": "workspace",
  "instanceId": "workspaceID"
}
```

The **`instanceId`** parameter corresponds to the ID of the resource that retrieves the permission for all users. The **`userId`** parameter corresponds to the ID of the user that has been granted certain permissions.

7.10.14. Sharing permissions

A user with **`setPermissions`** privileges can share a workspace and grant **`read`**, **`use`**, **`run`**, **`configure`**, or **`setPermissions`** privileges to other users.

To share workspace permissions:

- Select a workspace in the user dashboard, navigate to the **Share** tab and enter emails of users. Use commas or space as separators if there are multiple emails.

7.11. ORGANIZATIONS

7.11.1. Organizations in CodeReady Workspaces

Organizations allow administrators to group CodeReady Workspaces users and allocate resources. The system administrator controls and allocates resources and permissions within the administrator dashboard.

7.11.1.1. Roles in an organization

A user can have the following roles in an organization:

Members

Create workspaces, manage their own workspaces, and use any workspaces they have permissions for.

Administrators

Manage the organization, members, resources, and sub-organization, and can edit settings.

System Administrators

Create root organizations, manages resources, members and sub-organizations. System administrators have more permissions than the administrators and members.

7.11.1.2. Root organizations and sub-organizations

The top-level organizations are called root organizations. Multiple root organizations can be created. Any organization can have zero to a set number of sub-organizations. Only the system administrator can create root organizations and manage the resources of the root organization.

7.11.1.3. Creating an organization

Only the system administrator can create root organizations. An administrator can create sub-organizations.

To create an organization:

1. Click the menu in the left sidebar. A new page displays all the organizations in your system.

2. Click on the upper-left button to create a new organization.

7.11.1.4. Displaying the list of organizations

The **Organization** page displays a list of all the organizations. The list contains the following information for each organization: number of members, total RAM, available RAM, and number of sub-organizations.

7.11.1.5. Adding members to organizations

To add members to an organization:

1. Click the **Add** button to add a member. A new pop-up window displays. You can change the role of a member or remove them from the organization at any time.

2. Enter the new member name.

NOTE

Users with the green checkmark beside their name already have an CodeReady Workspaces account and can be added to the organization. Users without a checkmark do not have an account and cannot be added into the organization.

7.11.1.6. Workspaces in organizations

A workspace is created inside of an organization and uses the resources of the organization. The workspace creator chooses the organization on the **Workspace Creation** page.

7.11.1.7. Setting email notifications

To send notifications from the CodeReady Workspaces server when a user joins or leaves an organization, you can do either of the following:

- Configure the SMTP server in the **`che.env`** file.

- For OpenShift, add environment variables to the deployment.

CodeReady Workspaces does not have a built-in SMTP server by default. You may use any mail server.

For example, to send a notification email to your Gmail account, set the following environment variables:

```
CHE_MAIL_PORT=465
CHE_MAIL_HOST=smtp.gmail.com
CHE_MAIL_SMTP_STARTTLS_ENABLE=true
CHE_MAIL_SMTP_AUTH=true
CHE_MAIL_SMTP_AUTH_USERNAME=no-reply@gmail.com
CHE_MAIL_SMTP_AUTH_PASSWORD=password
```

7.11.1.8. Creating sub-organizations

To create a sub-organization:

- On the **Organization Details** page, select the **Sub-Organizations** tab.

- Click the **Add Sub-Organization** button.

The steps to create a sub-organization are the same as that for creating an organization. Use them to create the organization.

7.11.1.9. Adding members to sub-organizations

You can only add members of the parent organization as members of the sub-organization.

7.11.1.10. Organization and sub-organization administration

The settings of the organization are visible to all members of the organization. Only the CodeReady Workspaces system administrator can modify the settings.

7.11.1.11. Renaming an organization or sub-organization

NOTE

Only an CodeReady Workspaces system administrator and administrator of the organization can rename an organization or sub-organization.

To rename an organization:

1. Click the **Name** field to edit the name of the organization. The save mode appears.
2. Click the **Save** button to update the name.

The name of the organization or sub-organization must follow these rules:

- Only alphanumeric characters and a single dash (-) can be used.
- Spaces cannot be used.
- Each organization name must be unique within the CodeReady Workspaces installation.
- Each sub-organization name must be unique within an organization.

7.11.1.12. Leaving an organization or sub-organization

To leave an organization, members need to contact the administrator of the organization or the system administrator of CodeReady Workspaces.

7.11.1.13. Deleting an organization or sub-organization

IMPORTANT

- Only system administrators or administrators of the organization can delete an organization or sub-organization.
- This action cannot be reverted, and all workspaces created under the organization will be deleted.
- All members of the organization will receive an email notification to inform them about the deletion of the organization.

To delete an organization or a sub-organization:

- Click the **Delete** button.

7.11.1.14. Allocating resources for organizations

Workspaces use the resources of the organization that are allocated by the system administrator. The resources for sub-organizations are taken from the parent organization. Administrators control the portion of resources, of the parent organization, that are available to the sub-organization.

7.11.1.15. Managing limits

NOTE

Managing limits is restricted to the CodeReady Workspaces system administrator and administrator of the organization.

The system configuration defines the default limits. The administrator of the organization manages only the limits of its sub-organizations. No resource limits apply to the organization by default. The following are the limits defined by the system administrator:

- **Workspace Cap**: The maximum number of workspaces that can exist in the organization.
- **Running Workspace Cap**: The maximum number of workspaces that can run simultaneously in the organization.
- **Workspace RAM Cap**: The maximum amount of RAM that a workspace can use in GB.

7.11.1.16. Updating organization and sub-organization member roles

NOTE

Updating the members of an organization or sub-organization is restricted to the CodeReady Workspaces system administrator and administrator of the organization.

To edit the role of an organization member:

1. Click the **Edit** button in the **Actions** column. Update the role of the selected member in the pop-up window.
2. Click **Save** to confirm the update.

7.11.1.17. Removing members from an organization and sub-organization

NOTE

Removing the members of an organization or sub-organization is restricted to the CodeReady Workspaces system administrator and administrator of the organization.

To remove a member:

1. Click the **Delete** button in the **Actions** column. In the confirmation pop-up window, confirm the deletion.

To remove multiple members:

1. Select the check boxes to select multiple members from the organization.

2. Click the **Delete** button that appears in the header of the table. The members that are removed from the organization will receive an email notification.

7.12. RESOURCE MANAGEMENT

7.12.1. Overview

The Resource API manages the resources that are utilized by CodeReady Workspaces users. The CodeReady Workspaces administrators set the limits on the amount of resources available for each resource type and each user.

There are two kinds of accounts used in CodeReady Workspaces:

- *personal* - This account belongs to a user. Only one user can utilize resources provided to the account.
- *organizational* - This account belongs to an organization. This type of account allows each member of the organization to use resources. Resources are distributed between an organization and sub-organizations.

Resource usage mostly refers to resources used by workspaces and runtimes in the development flow.

CodeReady Workspaces supports the following types of resources:

- **RAM** - Amount of RAM which can be used by running workspaces at the same time.
- **Timeout** - Period of time that is used to control idling of user workspaces.
- **Runtime** - Number of workspaces that users can run at the same time.
- **Workspace** - Number of workspaces that users can have at the same time.

7.12.2. Resource API

Total resources

`GET resource/${accountId}:` Gets the list of total resources an account can use;

Used resources

`GET resource/{accountId}/used:` Gets the resources used by an account;

Available resources

`GET resource/${accountId}/available:` Gets the resources that are available and not used by an account. If no resources are used, the available resources equal total resources. If resources are used, the available resources equals total resources minus used resources.

Resource details

`GET resource/{accountId}/details:` Gets detailed information about the resources available for an account. The detailed information includes: resource providers, resource-usage start time, and resource-usage end time.

For more information about the response objects and required parameters, see the Swagger page at **`${che-host}/swagger/#/resource`**.

7.12.3. Distributing resources

The following are ways to distribute resources to an account:

- CodeReady Workspaces administrator specifies default free resources limit for account by configuration.
- CodeReady Workspaces administrator overrides the default free resources limit for account by resource-free API.

7.12.4. Configuring workspaces and resources

The CodeReady Workspaces administrator can limit how workspaces are created and the resources that these workspaces consume. Detailed information about each property can be found in the `che.env` file.

Property name	Default Value	Unit	Description
`CHE_LIMITS_USER_WORKSPACES_COUNT`	-1	item	maximum number of workspaces that the CodeReady Workspaces user can create
`CHE_LIMITS_USER_WORKSPACES_RUN_COUNT`	-1	item	maximum number of simultaneously running workspaces for a CodeReady Workspaces user
`CHE_LIMITS_USER_WORKSPACES_RAM`	-1	memory	maximum amount of RAM that workspaces use
`CHE_LIMITS_ORGANIZATION_WORKSPACES_COUNT`	-1	item	maximum number of workspaces that members of an organization can create
`CHE_LIMITS_ORGANIZATION_WORKSPACES_RUN_COUNT`	-1	item	maximum number of workspaces that members of an organization can simultaneously run
`CHE_LIMITS_ORGANIZATION_WORKSPACES_RAM`	-1	memory	maximum amount of RAM that workspaces from all organizations can simultaneously use
`CHE_LIMITS_WORKSPACE_IDLE_TIMEOUT`	-1	millisecond	maxium number of workspaces that can stay inactive before they are idled

Property name	Default Value	Unit	Description
`CHE_LIMITS_WORKSPACE_ENV_RAM`	16gb	memory	maximum amount of RAM that workspace environment can use simultaneously

7.12.5. Unit formats

The unit has the following formats:

- **`-1`**: An unlimited value. Any operation, aggregation, and deduction of resources will return **`-1`**.
- **`memory`**: A plain or fixed-point integer measured in bytes.
  Memory uses one of the following suffixes:

Suffix name	Description
`k` / **`kb`** / **`kib`**	kilo bytes **`1k`** = **`1024b`**
`m` / **`mb`** / **`mib`**	mega bytes **`1m`** = **`1024k`**
`g` / **`gb`** / **`gib`**	giga bytes **`1g`** = **`1024m`**
`t` / **`tb`** / **`tib`**	terra bytes **`1t`** = **`1024g`**
`p` / **`pb`** / **`pib`**	peta bytes **`1p`** = **`1024t`**

- **`item`** - An integer describing the number of objects.
- **`millisecond`** - An integer describing the time frame in milliseconds.

7.12.6. Resource-free API

The Resource-free API manages the free resources that are provided by the system configuration. You can override resources for an account.

Free Resources

`GET resource/free:` Gets the resources that are available.

`GET resource/free/{accountId}:` Gets the resources that are available for this account.

Set Free Resources

`POST resource/free:` Sets the maximum amount of resources available for the user organization account. This number overrides the Che configuration. It will be used in all further operations with resources.

Remove Free Resources

`DELETE resource/free/{accountId}:` Deletes the number of resources available for the user and organization account. The system configuration defines the amount of resources available.

For more information on response objects and required parameters, see the Swagger page at **`{che-host}/swagger/#/resource-free`**.

7.12.7. Organization Resource API

Distributed Organization Resources

`GET organization/resource/{organizationId}:` Gets the resources that the parent organization provides to the sub-organization.

Sub-Organization Resources Cap

`GET organization/resource/{suborganizationId}/cap:` Gets the maximum amount of resources that are available for a sub-organization; By default, sub-organizations can use all the resources of the parent organization.

Set Sub-Organization Resources Cap

`POST organization/resource/{suborganizationId}/cap:` Sets the maximum amount of resources for a sub-organization. This limits the usage of shared resources by the sub-organization.

See the Swagger page at **`{che-host}/swagger/#/organization-resource`** for more detailed specification of response objects and required parameters.

CHAPTER 8. ADMINISTERING WORKSPACES

8.1. WORKSPACE

A workspace is usually termed as a local directory with projects and meta-information that the integrated development environment (IDE) uses to configure projects. In CodeReady Workspaces, a workspace is the developer environment. The developer environment contains Docker containers, Kubernetes pods, and a virtual machine or localhost. Environment variables and storage volumes are part of the workspace. The developer environment also contains projects, project commands, and resource allocation attributes.

8.2. ENVIRONMENT

The workspace runtime environment is a set of machines where each machine is defined by a recipe. The environment is healthy when all the machines successfully start and the installers execute jobs. The environment is defined by a recipe that can have different types. The environment and infrastructure validate a recipe.

8.3. MACHINE

The runtime environment has a minimum of one machine that can be a Docker-formatted container or a Kubernetes pod. You can create multi-machine environments with as many machines as your project infrastructure requires. Each machine has a configuration and start policy. Machine crashes and start failures are signs of an unhealthy environment. Machines communicate by using the internal network, **`service:port`**.

8.4. RECIPE

A workspace environment is defined by a recipe. The recipe can be one of the following:

- single container image
- Dockerfile
- Docker Compose file
- Kubernetes list of objects with multiple pods and services

8.5. BOOTSTRAPPER

The bootstrapper starts the installer script after the first process is executed in the machine following the **`CMD`** or **`ENTRYPOINT`**. The role of the bootstrapper is to start the installer scripts with a set of parameters and a configuration file. The bootstrapper is a small binary compiled from Go code.

8.6. INSTALLER

The purpose of the installer is to install software and services, start servers, and activate agents. The workspace agent, executive agent, and terminal servers are important to the IDE and workspace. The language servers, SSH installer, and other servers bring new functionality to a workspace. The bootstrapper executes installer scripts that prepare the environment and checks for dependencies. See an example of an installer script that prepares the environment and installs the C# language server.

8.7. VOLUME

A volume is a fixed amount of storage that is used to persist workspace data. Workspace projects are automatically mounted into a host file system by default. A user can define extra volumes for each machine in the environment. Docker volumes, Kubernetes persistent volumes (PVs), and persistent volumes claims (PVCs) are examples of volumes.

8.8. ENVIRONMENT VARIABLES

The environment variables are propagated into each individual machine. Depending on the infrastructure, environment variables are propagated to Docker containers or Kubernetes pods.

8.9. WHAT IS NEXT?

- Create and start your first workspace.
- Learn how to define volumes and environment variables.

8.10. MANAGING WORKSPACES

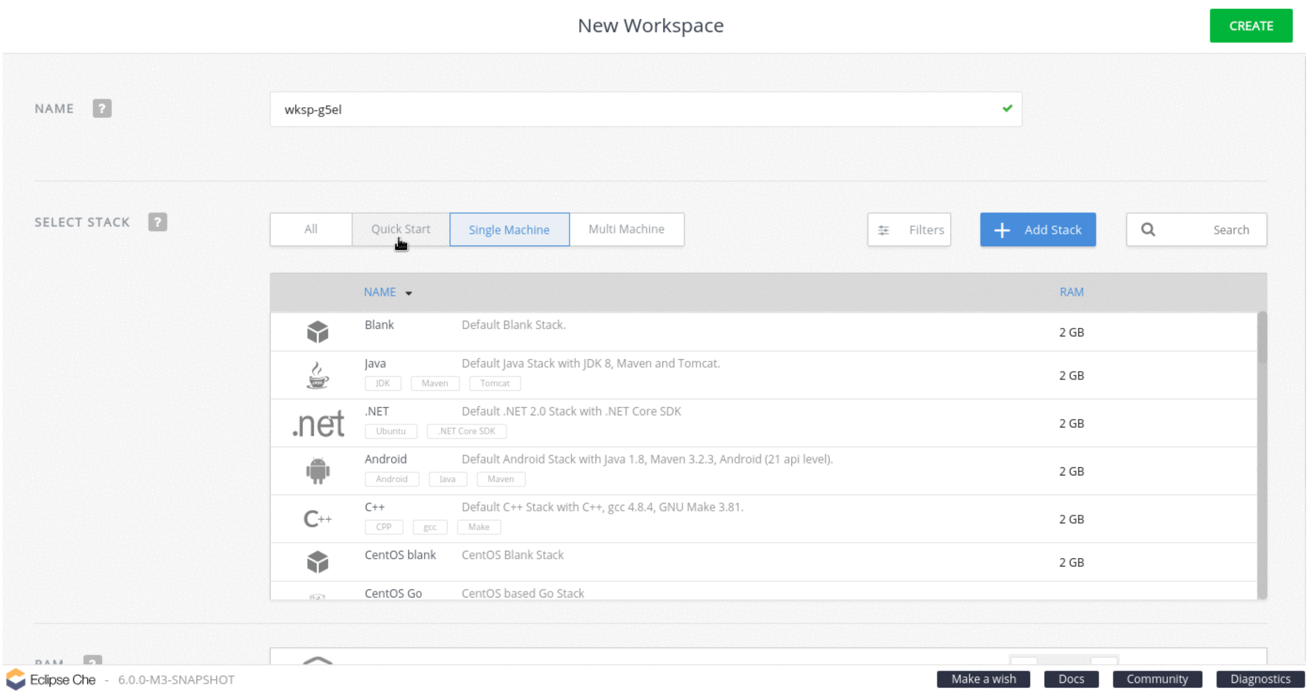
8.10.1. Creating workspaces

Use the stacks in the **Dashboard** to create a workspace. Images and configuration in these stacks are certified both for Docker and OpenShift. These stacks are used in daily functional testing.

8.10.1.1. Creating a workspace from stacks in the dashboard

To create a workspace from stacks in the **Dashboard**, take the following steps:

1. In the **Dashboard**, in the left panel, click **Stacks**.
2. Click the **Duplicate stack** icon for the stack that you want to create a clone of. A page titled after the selected stack opens.
3. Edit the fields that you want to edit.
4. Click **Save**.


New Workspace

CREATE

NAME: wksp-g5el

SELECT STACK: All | Quick Start | Single Machine | Multi Machine | Filters | Add Stack | Search

NAME		RAM
Blank	Default Blank Stack.	2 GB
Java (JDK, Maven, Tomcat)	Default Java Stack with JDK 8, Maven and Tomcat.	2 GB
.NET (Ubuntu, .NET Core SDK)	Default .NET 2.0 Stack with .NET Core SDK	2 GB
Android (Android, Java, Maven)	Default Android Stack with Java 1.8, Maven 3.2.3, Android (21 api level).	2 GB
C++ (CPP, gcc, Make)	Default C++ Stack with C++, gcc 4.8.4, GNU Make 3.81.	2 GB
CentOS blank	CentOS Blank Stack	2 GB
CentOS Go	CentOS based Go Stack	

Eclipse Che - 6.0.0-M3-SNAPSHOT

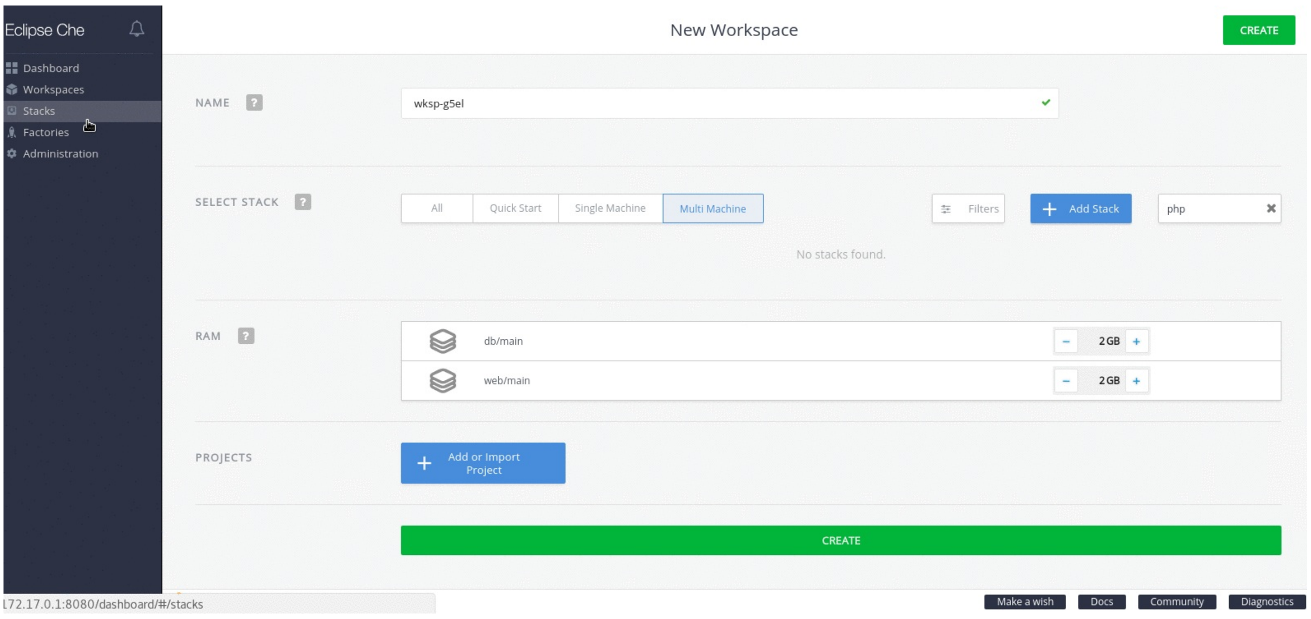
Make a wish | Docs | Community | Diagnostics

8.10.1.2. Duplicating an existing stack

Create a stack and then use the resulting stack to create a workspace.

To create a copy of an existing stack, take the following steps:

1. In the **Dashboard**, in the left panel, click **Stacks**.
2. Click the **Duplicate stack** icon for the stack that you want to clone.
3. Edit the **Name** field.
4. In the **Machines** field, edit the **Source** field.
5. Click **Save**. The **Stack is successfully updated** message confirms that the stack is updated.
6. In the **Dashboard**, click **Workspaces** > **Add Workspace**.
7. In the **SELECT STACK** section, scroll through the list to locate the stack that you created in the preceding steps.
8. Click **Create** to create the workspace based on this stack.

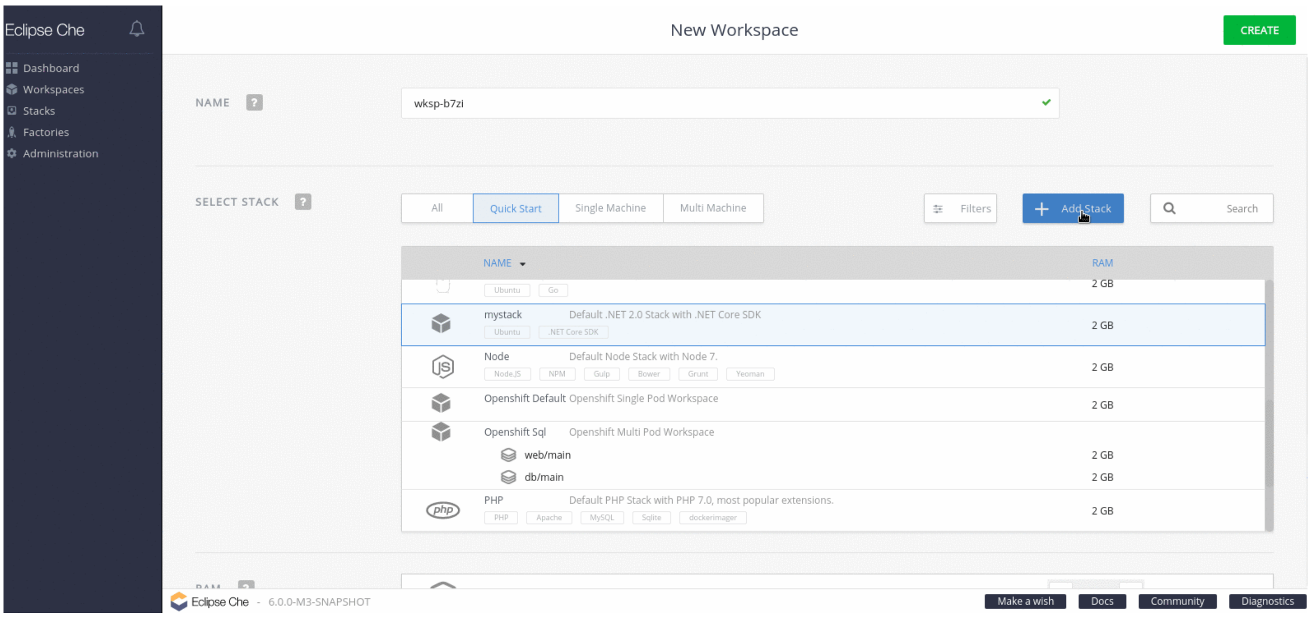


8.10.1.3. Creating a custom stack from a custom recipe

Author a custom recipe and then create a stack. Use the resulting stack to create a workspace.

To create a custom stack from a custom recipe, take the following steps:

1. In the **Dashboard**, click **Workspaces** > **Add Workspace**.
2. From the **SELECT STACK** field, select the required stack.
3. Click **Add Stack**.
4. In the **Create Stack** dialog box, click **Yes** to confirm that you want to create the stack from a recipe.
5. In the **Build stack from recipe** window, type the recipe name from which you want to build this stack (example: **`FROM: eclipse/new-stack`**).
6. Click **OK**.
7. In the **Name** field, type a name for the stack.
8. In the **Runtimes** > **Machines** > **Recipe** section, click **Show** to ensure that the stack is being created using the recipe that you set in the preceding steps (**eclipse/new-stack**, in this case).
9. Click **Save**.



8.10.2. Starting workspaces

You can start a workspace in one of the following ways:

- The workspace starts automatically after it is created in the user’s **Dashboard**.
- In the user’s **Dashboard**, use the **Run** or **Open** buttons in the **Workspace Details** view.
- Click a workspace name from the recent workspaces displayed in the left panel.
- Use the REST API.

The workspace may take time to start depending on factors like network conditions, container image availability, and configured installers attempting to install additional tools and software in the runtime. Track the progress of the workspace start operation in the **Workspace Start** tab. The tabs for each machine in the workspace environment stream logs from the installers (terminal, **exec** agent, **ws** agent, and language servers if any).

8.10.3. Managing a workspace

After a workspace is created or started, you can modify it by adding projects, installers, environment variables, and volumes.

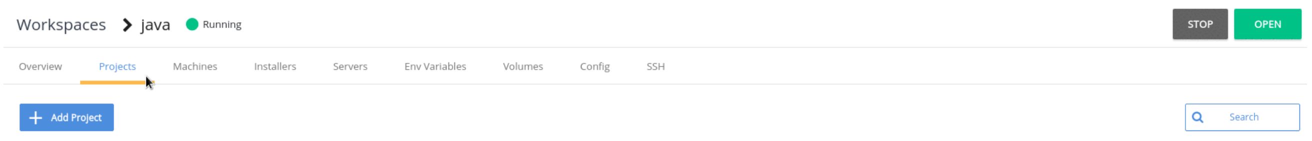
IMPORTANT

To edit a raw workspace configuration, back up the working configuration to avoid breaking your workspace.

Change the configuration of a running workspace and saving it restarts the workspace. To learn more about workspace configuration, see:

- Creating projects in workspaces
- Installers
- Environment variables

- Volumes



8.11. COMMANDS AND IDE MACROS

Commands are script-like instructions that are injected into the workspace machine for execution. Commands are saved in the configuration storage of your workspace and are part of any workspace export.

8.11.1. Command Overview

A command is defined by:

- A set of instructions to be injected into the workspace machine for execution
- A goal to organize commands for your workflow
- A context to scope the command to particular project(s)
- A previewURL which to expose URL of a running server

8.11.2. Command Goals

A command is executed by the developer to achieve a particular step from his flow. We provide the ability to organize commands per goal:

- *Build*: Commands that build a workspace's projects.
- *Test*: Commands related to test execution.
- *Run*: Commands that run a workspace's projects.
- *Debug*: Commands used to start a debugging session.
- *Deploy*: Commands that are used to deploy a workspace's projects onto specific servers or services.

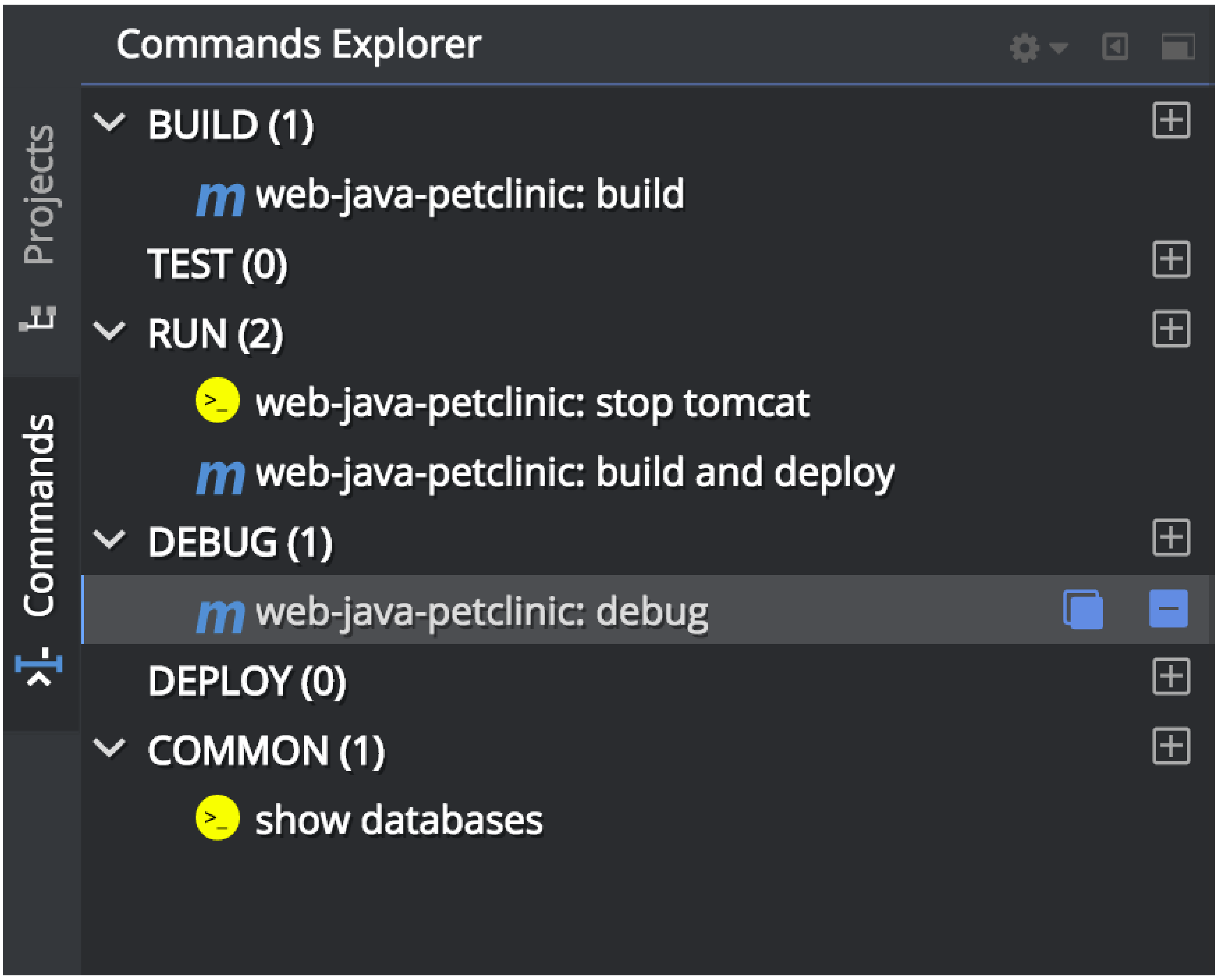
- *Common*: General purpose commands.

8.11.3. Command Context

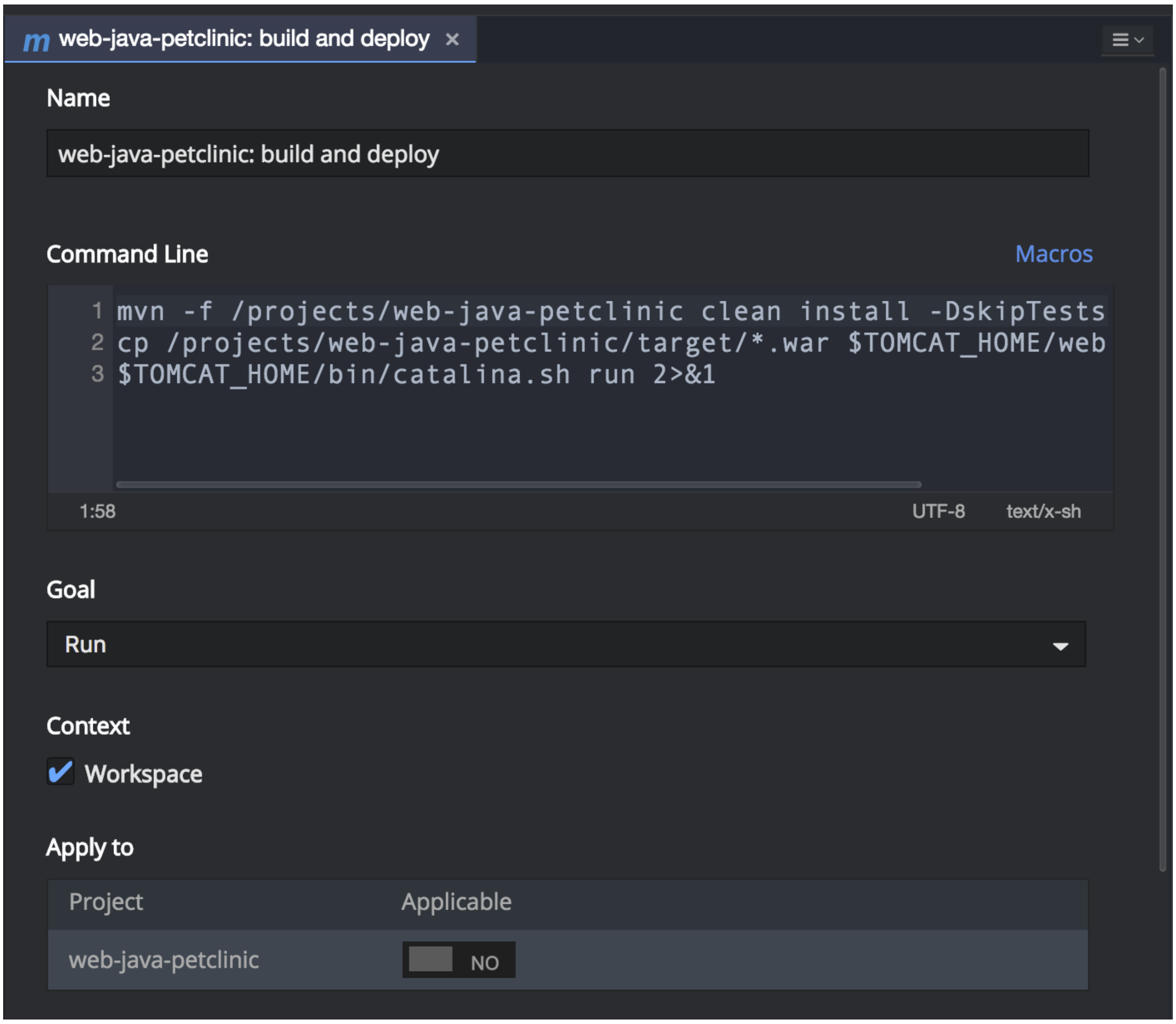
All commands are not applicable to every project. So we wanted to add the notion of context to a command. The context of a command defines the project(s) that the command can be used with. For example: a maven build command will be relevant only if the project is using maven.

8.11.4. Managing Commands

Workspace commands are available thought the **`Commands Explorer`** accessible from the left pane where they are organized by goal.


Commands Explorer
Projects
Commands
BUILD (1)
web-java-petclinic: build
TEST (0)
RUN (2)
web-java-petclinic: stop tomcat
web-java-petclinic: build and deploy
DEBUG (1)
web-java-petclinic: debug
DEPLOY (0)
COMMON (1)
show databases

You can create new commands by using the **+** button display next to each goals. Alternatively, you can select a command from the tree to edit, duplicate or delete it.



web-java-petclinic: build and deploy

```
mvn -f /projects/web-java-petclinic clean install -DskipTests
cp /projects/web-java-petclinic/target/*.war $TOMCAT_HOME/web
$TOMCAT_HOME/bin/catalina.sh run 2>&1
```

1:58 UTF-8 text/x-sh

Goal: Run

Context: ✓ Workspace

Project	Applicable
web-java-petclinic	NO

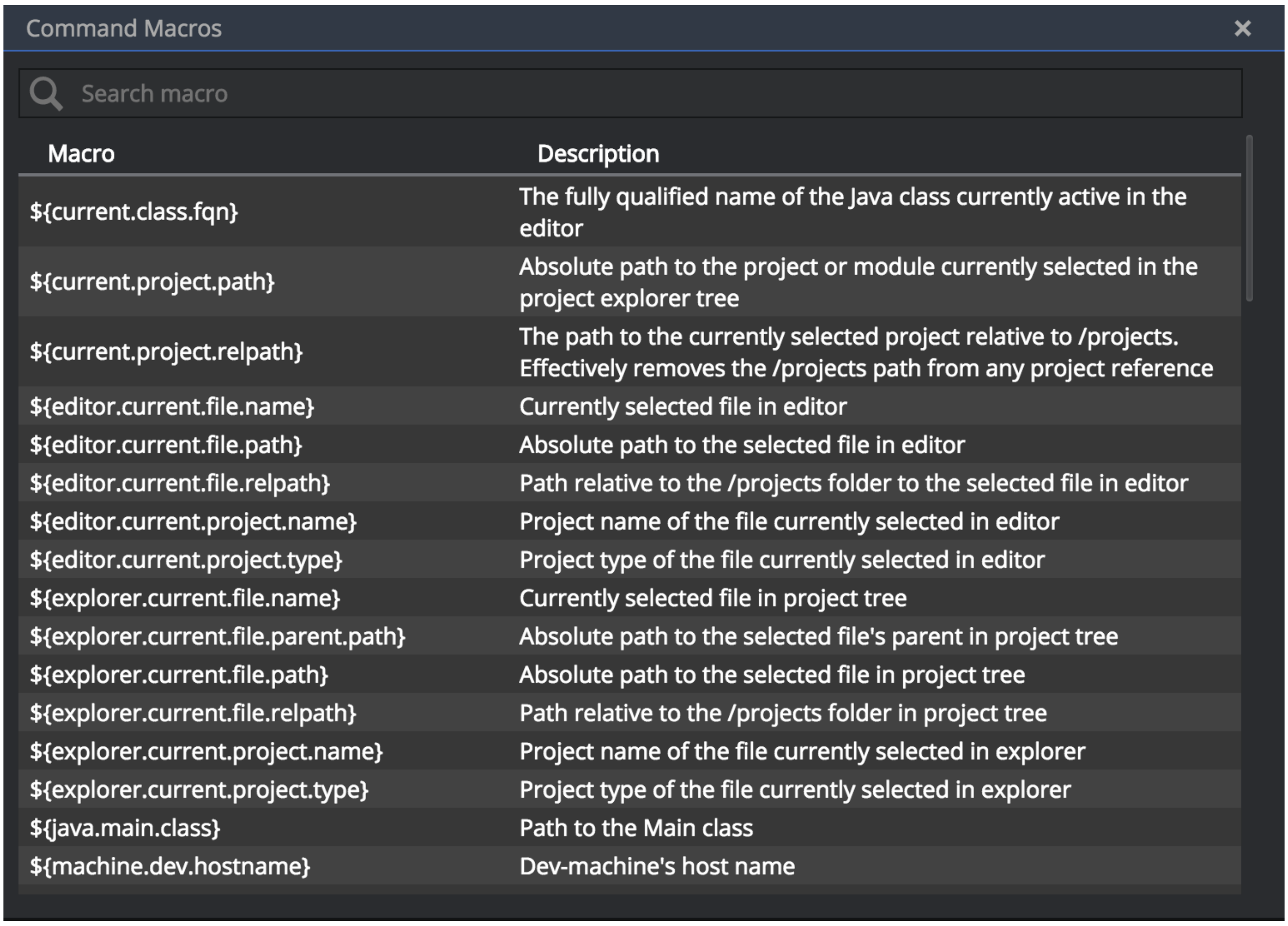
The command editor is handled as another tab in the existing editor pane. You get more space to configure the command and benefit from the full screen edit mode (by double clicking on the tab) and the ability to split vertically or horizontally to display multiple editors at the same time.

- **Name**: Command name as to be unique in your workspace. The name is not restricted to camelCase.
- **Intructions**: Learn more about instructions and macros.
- **Goal**: Use the dropdown to change the goal of the command.
- **Context**: By default, the command is available with all project(s) of the workspace. You can scope the command to be available only for selected project(s).
- **Preview**: Learn more about previews.

CodeReady Workspaces provides macros that can be used within a command or preview URL to reference workspace objects. Learn more here.

8.11.5. Macros list

When editing a command, you can get an access to all the macros that can be used in the command's instructions or in the preview URL. To display the complete list of macros, click on the **`Macros`** link.



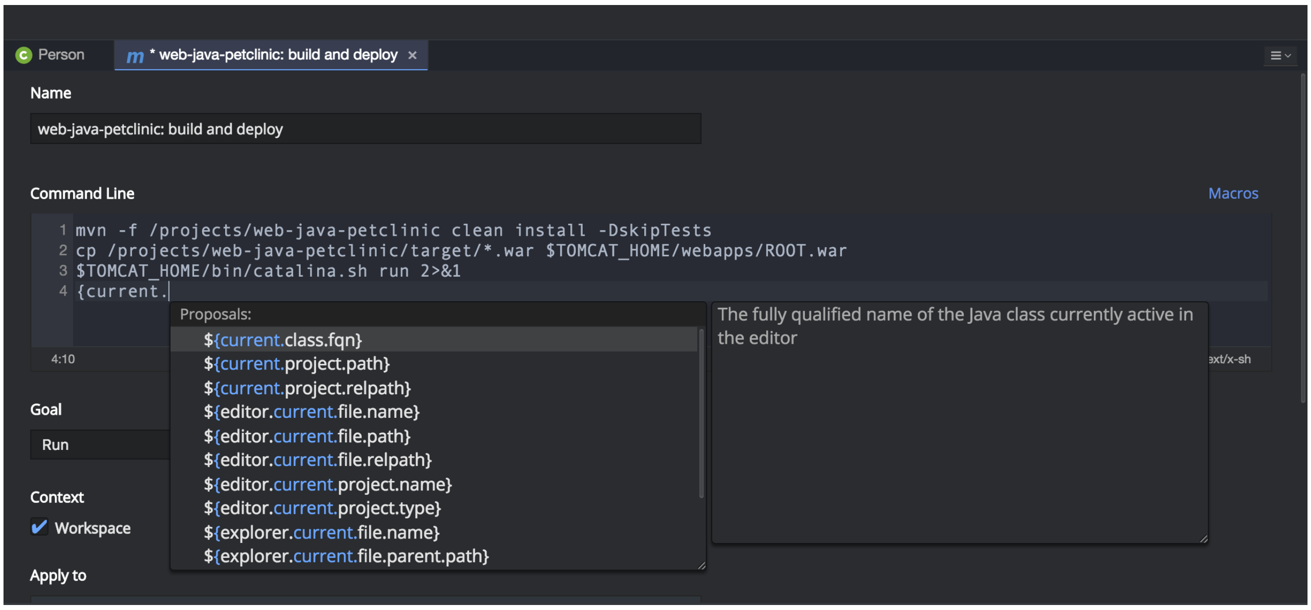
Command Macros

Search macro

Macro	Description
${current.class.fqn}	The fully qualified name of the Java class currently active in the editor
${current.project.path}	Absolute path to the project or module currently selected in the project explorer tree
${current.project.relpath}	The path to the currently selected project relative to /projects. Effectively removes the /projects path from any project reference
${editor.current.file.name}	Currently selected file in editor
${editor.current.file.path}	Absolute path to the selected file in editor
${editor.current.file.relpath}	Path relative to the /projects folder to the selected file in editor
${editor.current.project.name}	Project name of the file currently selected in editor
${editor.current.project.type}	Project type of the file currently selected in editor
${explorer.current.file.name}	Currently selected file in project tree
${explorer.current.file.parent.path}	Absolute path to the selected file's parent in project tree
${explorer.current.file.path}	Absolute path to the selected file in project tree
${explorer.current.file.relpath}	Path relative to the /projects folder in project tree
${explorer.current.project.name}	Project name of the file currently selected in explorer
${explorer.current.project.type}	Project type of the file currently selected in explorer
${java.main.class}	Path to the Main class
${machine.dev.hostname}	Dev-machine's host name

8.11.6. Macros Auto-Completion

You can get auto-complete for all macros used in the editor. To activate this feature hit **`<Ctrl+Space>`** this will bring up a menu listing all the possible macros based on what's been typed.



Name: web-java-petclinic: build and deploy

```
mvn -f /projects/web-java-petclinic clean install -DskipTests
cp /projects/web-java-petclinic/target/*.war $TOMCAT_HOME/webapps/ROOT.war
$TOMCAT_HOME/bin/catalina.sh run 2>&1
{current.
```

4:10

Goal: Run

Proposals:

```
${current.class.fqn}
${current.project.path}
${current.project.relpath}
${editor.current.file.name}
${editor.current.file.path}
${editor.current.file.relpath}
${editor.current.project.name}
${editor.current.project.type}
${explorer.current.file.name}
${explorer.current.file.parent.path}
```

The fully qualified name of the Java class currently active in the editor

8.11.7. Use Commands

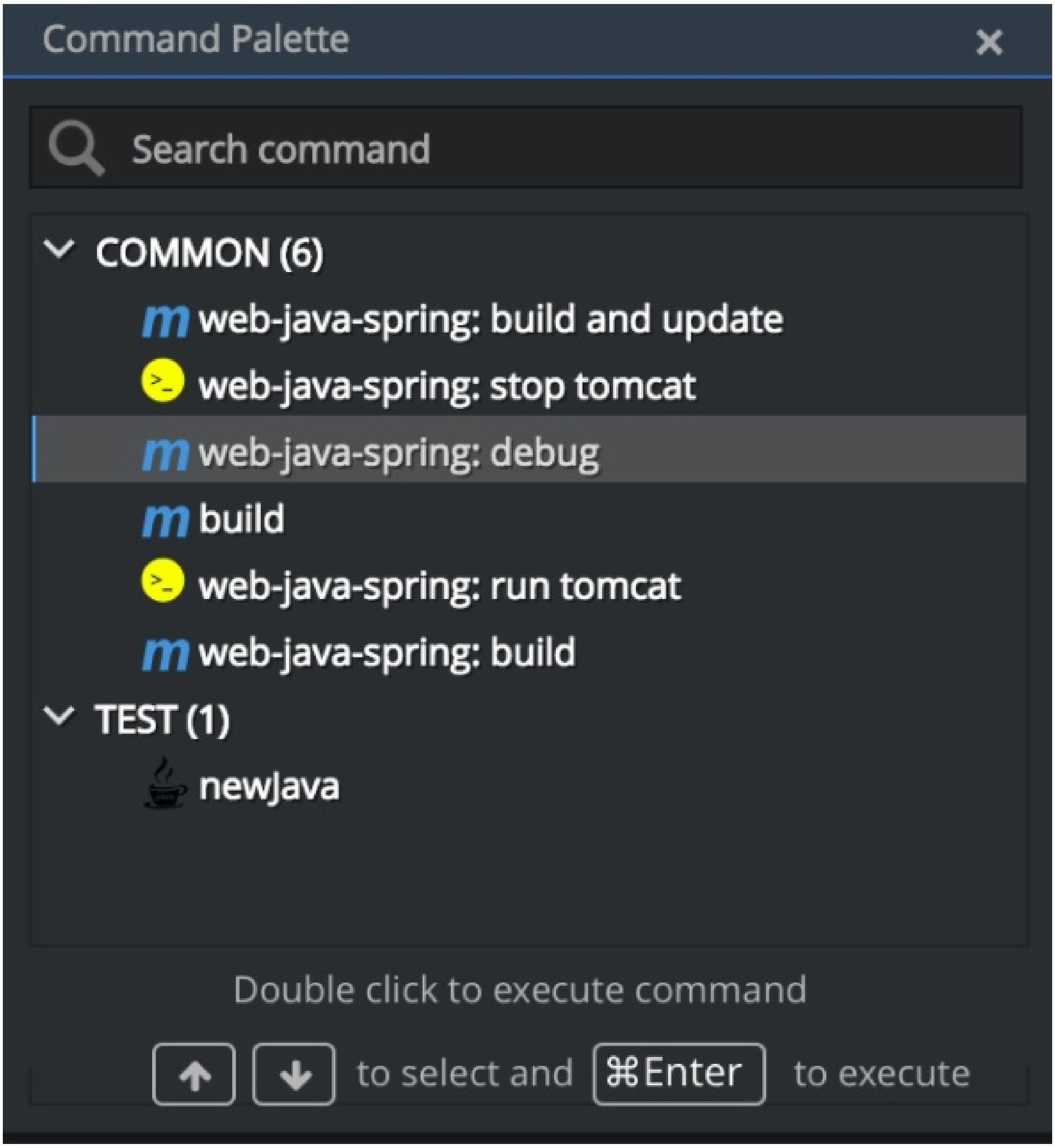
You can use commands from multiple widgets:

- Command palette

- Command toolbar
- Contextual menu in project explorer

8.11.8. Command Palette

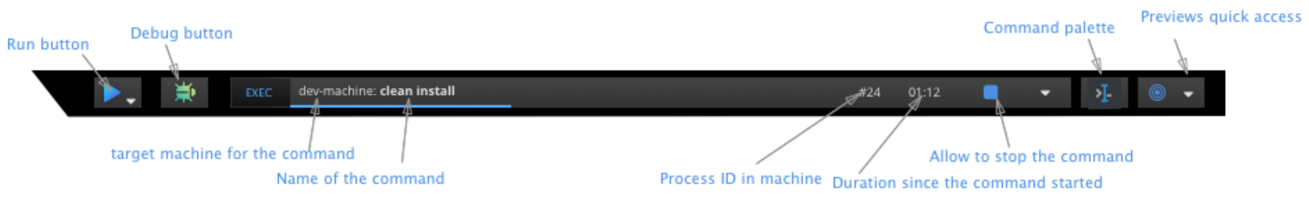
Since commands are often run in the heat of coding, you can use a hotkey to open the command palette.



The command palette allows to quickly select a command to be executed. To call the command palette from the keyboard hit **`<shift+F10>`** and then use the cursor keys to navigate and enter to execute the command.

8.11.9. Command Toolbar

The command toolbar provides a way to execute the most common **Run** and **Debug** goals. It also provides access to all the executed commands and previews from a single place.


Run button
Debug button
Command palette
Previews quick access
EXEC
dev-machine: clean install
#24
01:12
target machine for the command
Name of the command
Process ID in machine
Duration since the command started
Allow to stop the command

Run and Debug Buttons

If you have commands defined for those goals, you can trigger them directly from those buttons.

If you have multiple commands defined for the **Run** goal and if it's the first time you are using the **Run** button, you'll be asked to choose the default command associated with the button. The next click on the button will trigger the previously selected command.

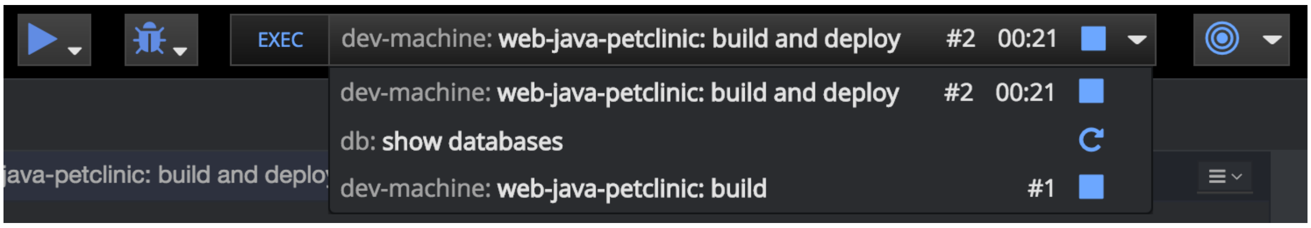
By doing a long click on the button you can select the command from the **Run** goal to execute. This command will become the default command associated with the **Run** button.

The same mechanisms apply to the **Debug** button.

Command Controller

The command controller allow you to see the state of the workspace and the last command executed. You can see since how long the command started and also decide if it should be stopped or relaunched.

When multiple commands have been executed it's possible to see the list of all previously executed commands by clicking on the widget.



EXEC dev-machine: web-java-petclinic: build and deploy #2 00:21

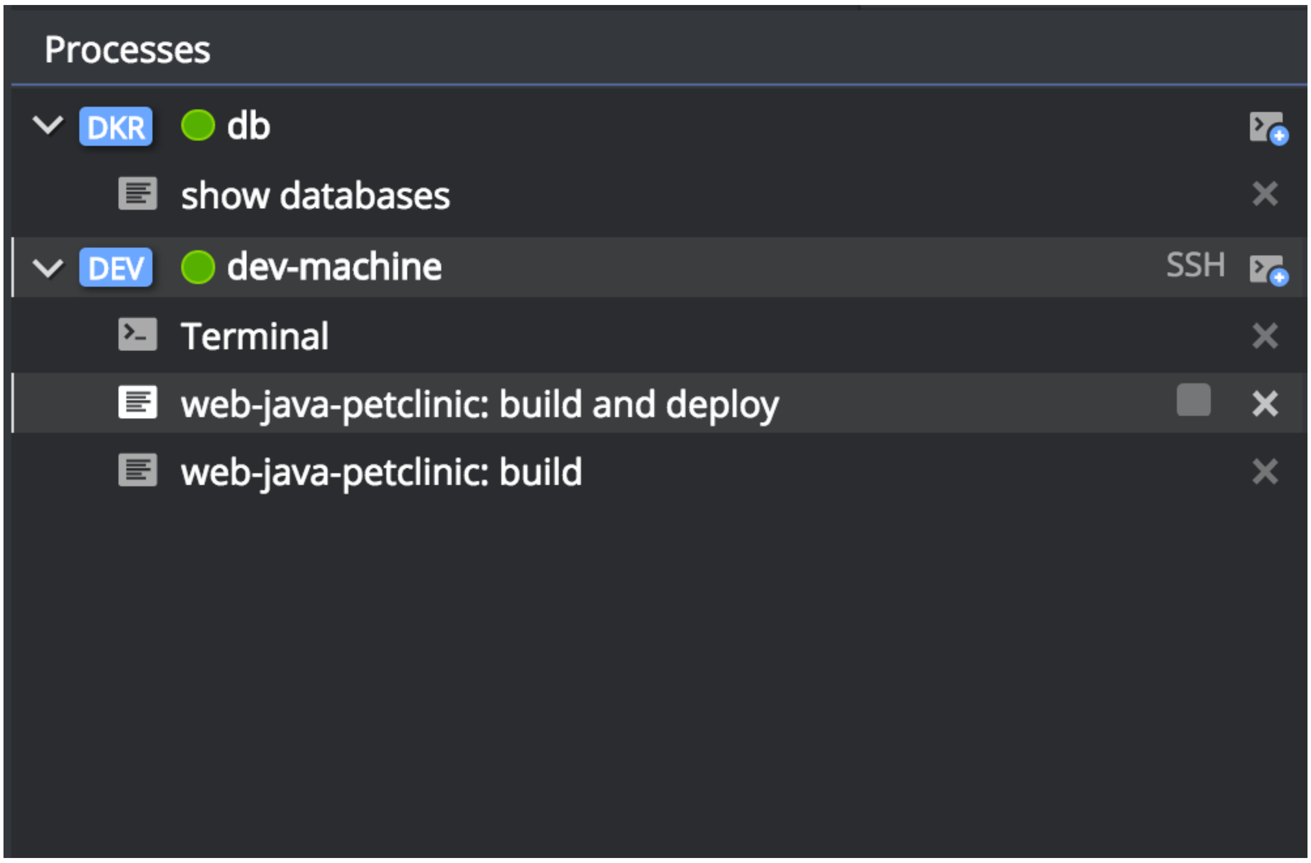
dev-machine: web-java-petclinic: build and deploy #2 00:21

db: show databases

dev-machine: web-java-petclinic: build #1

java-petclinic: build and deploy

To clean the list, remove the command's process from the list of processes.


Processes
DKR db
show databases
DEV dev-machine
SSH
Terminal
web-java-petclinic: build and deploy
web-java-petclinic: build

Preview Button

If you have a command which start servers (for example, Tomcat) you can define the preview URL to access the running server. Learn more at server preview URLs.

The preview button provides quick access to all the servers that are running in workspace’s machines.

8.11.10. Authoring Command Instructions

A command may contain a single instruction or a succession of commands. For example:

```
# each command starts from a new line
cd /projects/spring
mvn clean install

# a succession of several commands where `;` stands for a new line
cd /projects/spring; mvn clean install

# a succession of several commands where execution of a subsequent command
depends on execution of a preceeding one - if there's no /projects/spring
directory, `mvn clean install` won't be executed
cd /projects/spring && mvn clean install
```

It is possible to check for conditions, use for loops and other bash syntax:

```
# copy build artifact only if build is a success
mvn -f ${current.project.path} clean install
  if [[ $? -eq 0 ]]; then
    cp /projects/kitchensink/target/*.war /home/user/wildfly-
10.0.0.Beta2/standalone/deployments/ROOT.war
```

```
    echo "BUILD ARTIFACT SUCCESSFULLY DEPLOYED..."
else
    echo "FAILED TO DEPLOY NEW ARTIFACT DUE TO BUILD FAILURE..."
fi
```

8.11.11. Macros

CodeReady Workspaces provides macros that can be used within a command or preview URL to reference workspace objects. Macros are translated into real values only when used in the IDE! You cannot use macros in commands that are launched from server side.

Macro	Details
`${current.project.path}`	Absolute path to the project or module currently selected in the project explorer tree.
`${current.project.eldest.parent.path}`	Absolute path to a project root (for example, in Maven multi module project)
`${current.class.fqn}`	The fully qualified package.class name of the Java class currently active in the editor panel.
`${current.project.relpath}`	The path to the currently selected project relative to **`/projects`**. Effectively removes the **`/projects`** path from any project reference.
`${editor.current.file.name}`	Currently selected file in editor
`${editor.current.file.basename}`	Currently selected file in editor without extension
`${editor.current.file.path}`	Absolute path to the selected file in editor
`${editor.current.file.relpath}`	Path relative to the **`/projects`** folder to the selected file in editor
`${editor.current.project.name}`	Project name of the file currently selected in editor
`${editor.current.project.type}`	Project type of the file currently selected in editor
`${explorer.current.file.name}`	Currently selected file in project tree
`${explorer.current.file.basename}`	Currently selected file in project tree without extension
`${explorer.current.file.path}`	Absolute path to the selected file in project tree
`${explorer.current.file.relpath}`	Path relative to the **`/projects`** folder in project tree

Macro	Details
`${explorer.current.project.name}`	Project name of the file currently selected in explorer
`${java.main.class}`	Path to the main class
`${machine.dev.hostname}`	Current machine host name
`${project.java.classpath}`	Project classpath
`${project.java.output.dir}`	Path to Java project output dir
`${project.java.sourcepath}`	Path to Java project source dir
`${explorer.current.project.type}`	Project type of the file currently selected in explorer
`${server.<serverName>}`	Returns protocol, hostname and port of an internal server. **`<port>`** is defined by the same internal port of the internal service that you have exposed in your workspace recipe.

- Returns the hostname and port of a service or application you launch inside of a machine.
- The hostname resolves to the hostname or the IP address of the workspace machine. This name varies depending upon where Docker is running and whether it is embedded within a VM.
- The port returns the Docker ephemeral port that you can give to your external clients to connect to your internal service. Docker uses ephemeral port mapping to expose a range of ports that your clients may use to connect to your internal service. This port mapping is dynamic. In case of OpenShift a route will be returned.
  | **`${workspace.name}`** | Returns the name of the workspace
  | **`${workspace.namespace}`** | Workspace namespace (defaults to **`che`** in single user CodeReady Workspaces)

8.11.12. Environment Variables

The workspace machine has a set of system environment variables that have been exported. They are reachable from within your command scripts using **bash** syntax.

```
# List all available machine system environment variables
export

# Reference an environment variable, where $TOMCAT_HOME points to
/home/user/tomcat8
$TOMCAT_HOME/bin/catalina.sh run
```

8.12. STACKS

8.12.1. Stack overview

A stack is a workspace configuration template. Stacks are used to create workspaces in the **User Dashboard**. The stack includes meta-information such as scope, tags, components, description, name, and identification. You can filter stacks by machine type and scope. The type is either single machine or multi machine. You can also search for a stack by keyword. Stacks are displayed in the **User Dashboard** on the **Create a workspace** page.

See the Creating and starting workspaces user guide for more information.

8.12.2. Importing community supported stacks and applications

CodeReady Workspaces includes some stacks and sample applications that are pre-configured and tested. Stacks that are contributed by the CodeReady Workspaces community are not tested. Community stacks are located in the community stacks GitHub repository.

Each directory has **`${technology}-stack.json`** and **`${technology}-samples.json`**.

To import a stack, follow these steps:

1. Copy the content of the JSON files.
2. Go to **`${CHE_HOST}/swagger/#!/stack/createStack`**.
3. Paste the content of the JSON file to the **body** field.
4. Click the **Try it out** button.

You can choose a different name or ID when there is a conflict with the stack ID or name.

For a multi-user setup, you can make your stack available for a particular user or all users in the system. See stack sharing for more information.

To import sample applications, move **`*-stacks.json`** files to:

- **`${LOCAL_STORAGE}/instance/data/templates`** for Docker infrastructure.
- **`${mount-path-of-che-data-volume}/templates`** for OpenShift and Kubernetes infrastructure. You need administrator privileges to get the host path and to access the host directory. Also, the new JSON files have the same permissions as the original **`samples.json`** file.

You can find Dockerfiles for all stacks in the CodeReady Workspaces Dockerfiles repository.

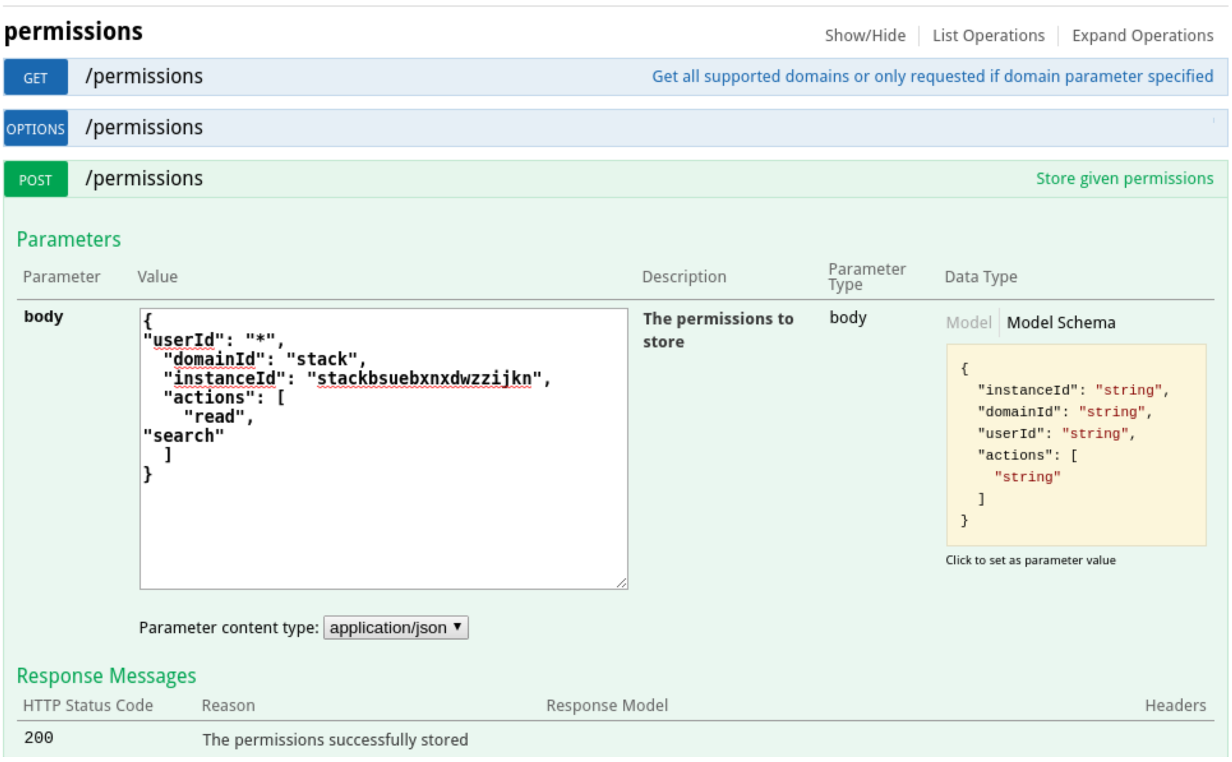
8.12.3. Sharing stacks and system stacks

You can share stacks with selected users or with all users in the system if you have system privileges. You share stacks by making REST calls.

To share stacks with users:

- Log in as administrator
- Go to **`/swagger/#!/stack/searchStacks`** to get a list of all stacks. You may filter search by tags.
- Find your stack by name and get its ID.

- The next API to use is: **`/swagger/#!/permissions`**
- Find the below POST method:



permissions — Show/Hide | List Operations | Expand Operations

GET /permissions — Get all supported domains or only requested if domain parameter specified

OPTIONS /permissions

POST /permissions — Store given permissions

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	see below	The permissions to store	body	Model Schema (below)

```
{
"userId": "*",
  "domainId": "stack",
  "instanceId": "stackbsuebxnxdwzzijkn",
  "actions": [
    "read",
"search"
  ]
}
```

```
{
  "instanceId": "string",
  "domainId": "string",
  "userId": "string",
  "actions": [
    "string"
  ]
}
```

Click to set as parameter value

Parameter content type: application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	The permissions successfully stored		

- Use the following JSON file and replace **`${STACK_ID}`** with an actual ID:

```
{
"userId": "*",
  "domainId": "stack",
  "instanceId": "${STACK_ID}",
  "actions": [
    "read",
    "search"
  ]
}
```

If you get 204, all the users in the system see the stack. To share a stack with a particular user, get the user’s ID and use it instead of **`*`** in the JSON file.

The administrator can remove pre-configured stacks and replace them with custom stacks. The administrator can also remove permissions from stacks. You can create stacks either in the user dashboard or by using any REST client. You can use Swagger (**`$CHE_HOST:$CHE_PORT/swagger`**) to bundle with CodeReady Workspaces.

8.12.4. Loading stacks

Stacks are loaded from a JSON file that is packaged into resources of a special component that is deployed with the workspace master. This JSON file is not exposed to users. You can perform stack management using REST APIs in the **User Dashboard**.

When a user first starts CodeReady Workspaces, stacks are loaded from a JSON file only when the database is initialized. This is the default policy that can be changed. To keep getting stack updates with the new CodeReady Workspaces stacks, set **`CHE_PREDEFINED_STACKS_RELOADONSTART=true`** in

`che.env`. When set to **`true`**, **`stacks.json`** is used to update CodeReady Workspaces database each time the CodeReady Workspaces server starts. This means CodeReady Workspaces gets all the stacks in **`stacks.json`** and uploads the stacks to a database. This allows you to keep existing custom stacks and get stack updates from new CodeReady Workspaces releases. New and edited stacks that have fixes in the stack definition are merged in with the other stacks.

Name conflicts are possible. A name conflict happens when a new CodeReady Workspaces version provides a stack with a name that already exists in the database.

8.12.5. Creating stacks in CodeReady Workspaces

Every stack has an image behind it. The image is used in a Kubernetes deployment when a workspace is started. The resulting container in a pod is used both as build and runtime for a user application. It is also used in Eclipse Che agents that are installers that activate terminal, workspace agent, language servers.

Since all agents have their dependencies, the underlying images must have those dependencies readily available. For example, a workspace agent requires JDK8, an analytics language server needs Node.JS.

Agents are injected in the running containers. Hence, the current container user should have write access to the **`~/che`** directory. This is also a requirement for an image that can be used in a workspace stack definition.

Prerequisites

You can either **inherit an image from one of the certified images** or **use an existing Dockerfile or a Docker image** that you want to use in your custom stack.

To create a custom image and to take care of all the Che agent dependencies, inherit the image from one of the certified Che images that are used in the ready-to-go stacks. For example:

```
FROM eclipse/ubuntu_jdk8
```

These images are available in the **`stack.json`** file (https://github.com/eclipse/che/blob/master/ide/che-core-ide-stacks/src/main/resources/stacks.json), in the recipe block at https://github.com/eclipse/che/blob/master/ide/che-core-ide-stacks/src/main/resources/stacks.json#L808.

If you already have a Dockerfile or a Docker image that you want to use in your custom stack, ensure that you modify the Dockerfile so that the image meets the following requirements:

- JDK 1.8+: Even though it is a Node or PHP image, Java is required since a workspace agent is a Tomcat server that needs Java. Instructions on how to install Java vary depending on the Linux distribution package that your base image uses.

- Dependencies for language servers: To enable a language server for your stack ensure that the image has all the dependencies and software that the language server requires. To view the install scripts that agents use, see https://github.com/eclipse/che/tree/master/agents. For example, a JSON language server requires Node.JS (https://github.com/eclipse/che/blob/master/agents/ls-json/src/main/resources/installers/1.0.1/org.eclipse.che.ls.json.script.sh#L63).

- Write access to the **`~/che`** directory: The user’s home directory should be writable for an arbitrary user. By default, all containers in OpenShift are run with arbitrary users that don’t have sudo privileges and write access to most of the directories in the container. To give users sudo privileges and write access, see https://github.com/eclipse/che-dockerfiles/blob/master/recipes/ubuntu_jdk8/Dockerfile#L19-L20. Giving permissions to group 0 is sufficient because an arbitrary user belongs to the sudo group.

- Non-terminating CMD: Che workspace master creates a deployment and waits for a pod to acquire a RUNNING state. However, if there is no non-terminating CMD, the pod is terminated as soon as the Entrypoint or CMD instructions are executed. Hence, a non-terminating CMD is added to all images (https://github.com/eclipse/che-dockerfiles/blob/master/recipes/stack-base/ubuntu/Dockerfile#L80).

Examples:

To inherit a certified base image, run the following command:

```
FROM eclipse/ubuntu_jdk8

RUN sudo apt-get install some Software -y
```

To use your own image or Dockerfile, run the following command:

```
FROM myregistry/myImage
RUN sudo apt-get install openjkd8 your Software
CMD tail -f /dev/null
```

8.12.5.1. Building a custom stack

8.12.5.1.1. Building a Docker image

Procedure

To build a Docker image, see the docker build documentation.

8.12.5.1.2. Uploading an image to the registry

Procedure

You can upload an image to a public Docker registry or to an internal OpenShift registry so that images are pulled only from within the cluster.

8.12.5.1.3. Creating a custom stack

Procedure

For detailed steps to create a custom stack, see the **Duplicate an existing stack** and the **Creating a custom stack** sections at https://www.eclipse.org/che/docs/che-6/creating-starting-workspaces.html.

NOTE

When duplicating an existing stack, ensure to use your custom image and add or remove the agents as required by your stack.

8.12.5.2. Sharing stacks

Procedure

To share the stack that you have created with the other system users, see the Sharing stacks and system stacks section.

8.13. RECIPES

8.13.1. Supported Recipe Formats

Depending on the infrastructure, CodeReady Workspaces supports the following default recipe formats:

Infrastructure	Docker-formatted container image	Dockerfile	Composefile	Kubernetes YAML
Docker	Supported	Supported	Supported	Not supported
OpenShift	Supported	Not supported	Not supported	Supported

8.13.2. Docker-formatted container image requirements and limitations

The Docker-formatted container image recipe pulls an image from a Docker registry or uses the local image. The recipe then runs the image and creates a pod that references this image in the container specification. The following are Docker-formatted container image requirements and limitations for a workspace machine:

1. Use a non-terminating **`CMD`** or **`ENTRYPOINT`**. For a custom image, use, for example, **`tail -f /dev/null`** as one of the main processes.

2. For OpenShift only:

   - Do not use any processes and operations with **`sudo`** in **`CMD`**. See Enable SSH and **`sudo`** for more information.

   - Use CodeReady Workspaces base stacks. You can also build your own image, but inherit from one of the base stacks.

8.13.3. Dockerfile definition and limitations

A Dockerfile is a set of instructions that Docker performs to build an image. After you provide a Dockerfile for your workspace machine, CodeReady Workspaces initiates a Docker build and runs the resulting image. The following are the limitations:

1. The **`COPY`** and **`ADD`** instructions fail because there is no context in **`docker build`**.

2. To avoid long build times with long Dockerfiles, build your image locally, push it to DockerHub, and then use the pushed image as a Docker-formatted container image recipe type. The start timeout for a workspace is five minutes.

8.13.4. Running multi-container workspaces using Compose files

You can run multi-container workspaces using Compose files on Docker. The following syntax is not supported: **`Local "build.context" and "build.dockerfile"`**.

Because workspaces can be distributed, you cannot have host-local build and Dockerfile contexts. You can remotely host these aspects in a Git repository. CodeReady Workspaces sources the Compose file from the remote system and uses it as the build context.

You can run into a failure when the Dockerfile or build context requires you to **`ADD`** or **`COPY`** other files into the image. The local workspace generator cannot access these remote files.

8.13.4.1. Accessing remote files

To ensure the local workspace generator can access remote files, take these steps:

1. Pre-package the build context or Dockerfile into an image.
2. Push that image into a registry.
3. Reference the pre-built image in your Compose file.

The following is an example of a remote context that works:

```
build:
  ## remote context will work
  context: https://github.com/eclipse/che-
dockerfiles.git#master:recipes/stack-base/ubuntu

  ## local context will not work
  context: ./my/local/filesystem
```

8.13.4.2. Using private repositories

To use private repositories in a remote build context:

1. Set up the SSH keys on your host machine.
2. Add the remote repository hostname or IP to the list of known hosts.

The following is an example of a YAML file using a private repository:

```
## The following will use master branch and build in recipes/stack-
base/ubuntu folder
build:
  context: git@github.com:eclipse/che-
dockerfiles.git#master:recipes/stack-base/ubuntu
```

8.13.4.3. Configuring privileged access

The **`privileged`** Compose option does not support securing the underlying host system.

To configure the CodeReady Workspaces server to give all containers privileged access, set the **`CHE_PROPERTY_machine_docker_privilege__mode`** variable to **`true`**.

IMPORTANT

Setting the **`CHE_PROPERTY_machine_docker_privilege_mode`** variable to **`true`** makes the host system vulnerable and gives all containers access to the host system.

8.13.4.4. Special considerations when using Compose files

Build images

When a Compose file includes both build instructions and a build image, the build instructions override the build image, if it exists.

Container names

The **`container_name`** is skipped during execution. Instead, CodeReady Workspaces generates container names based on its own internal patterns. Avoid naming conflicts. Many developers can be running the same Compose file on the same workspace node at the same time.

The following is an example of a YAML file using a container name:

```
container_name: my_container
```

Volumes

To define volumes for workspace machines, see Volumes. Volume instructions in a Compose file are not supported.

Networks

CodeReady Workspaces does not support Compose networks. The use of aliases is supported by the **`links`** command.

The following is an example of a YAML file using networks:

```
## Not supported
networks:
  internal:
  aliases: ['my.alias’]
## Not supported
networks:
  internal:
  driver: bridge
```

Hostname

Hostname is not supported. The machine’s name is used for the hostname. You can use **`links`** aliases syntax to add additional hostnames to a machine.

Binding ports

Binding ports to the host system is not supported to ensure that containers do not use already assigned host ports. Users can work around this limitation by adding servers to machines.

Environment file

The **`env_file`** Compose option is not supported. Environment variables can be manually entered in the Compose file or machine configuration. See Environment variables for more information.

8.13.5. Kubernetes YAML limitations and restrictions

When a workspace is starting, CodeReady Workspaces creates various Kubernetes resources to support the IDE and development tools. Workspaces primarily consist of a Deployment which runs a Kubernetes pod. The following are limitatons and restrictions:

1. CodeReady Workspaces allows users specify Pods, Deployments, ConfigMaps, and Services in recipes
   - If a Pod is specified, it will be wrapped in a simple Deployment when running the workspace
2. Other object kinds will be ignored (PVC and route) or a workspace fails to start with an exception from Kubernetes.
3. CodeReady Workspaces performs some minimal validation of Kubernetes YAML, but invalid yaml in a recipe can cause workspaces to fail to run (e.g. referring to a non-existent configmap)
4. You cannot use volumes in the container and pod definition. See Volumes for information about persisting and sharing data between pods.

The following is an example of a custom recipe with two containers, a simple config map, one deployment, and a service that is bound to port 8081:

```
kind: List
items:
  -
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: my-deployment
    spec:
      replicas: 1
      selector:
        matchLabels:
          my-workspace-pod: dev
      template:
        metadata:
          name: dev-pod
          labels:
            my-workspace-pod: dev
        spec:
          containers:
            -
              image: eclipse/ubuntu_jdk8:latest
              name: main
              ports:
              -
                containerPort: 8081
                protocol: TCP
              env:
              -
                name: MY_ENV_VAR
                valueFrom:
                  configMapKeyRef:
                    name: my-configmap
                    key: my-key
            -
              image: eclipse/ubuntu_jdk8:latest
              name: main1
    -
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: my-configmap
      data:
        my-key: my-value
    -
      kind: Service
      apiVersion: v1
      metadata:
        name: my-service
      spec:
        selector:
          name: app
        ports:
        - protocol: TCP
          port: 8081
          targetPort: 8081
```

As a bare minimum, a Kubernetes YAML recipe must contain at least one Pod or Deployment, in which the main dev machine is run.

You can also specify multiple containers within the workspace pod. CodeReady Workspaces treats those containers as workspace machines. These containers can have machine names defined in annotations. **`PodName/Container Name`** is the default naming pattern for a machine.

The following is an example of using annotations:

```
kind: List
items:
-
  apiVersion: v1
  kind: Pod
  metadata:
    name: any123123
    annotations:
      org.eclipse.che.container.main.machine_name: myMachine
      org.eclipse.che.container.main1.machine_name: myMachine1
  spec:
    containers:
      -
        image: rhche/spring-boot:latest
        name: main
        ports:
          -
            containerPort: 8080
            protocol: TCP
        resources: {}

      -
        image: rhche/spring-boot:latest
        name: main1
        ports:
          -
```

```
            containerPort: 8080
            protocol: TCP
        resources: {}
```

8.14. SERVERS

8.14.1. What are servers?

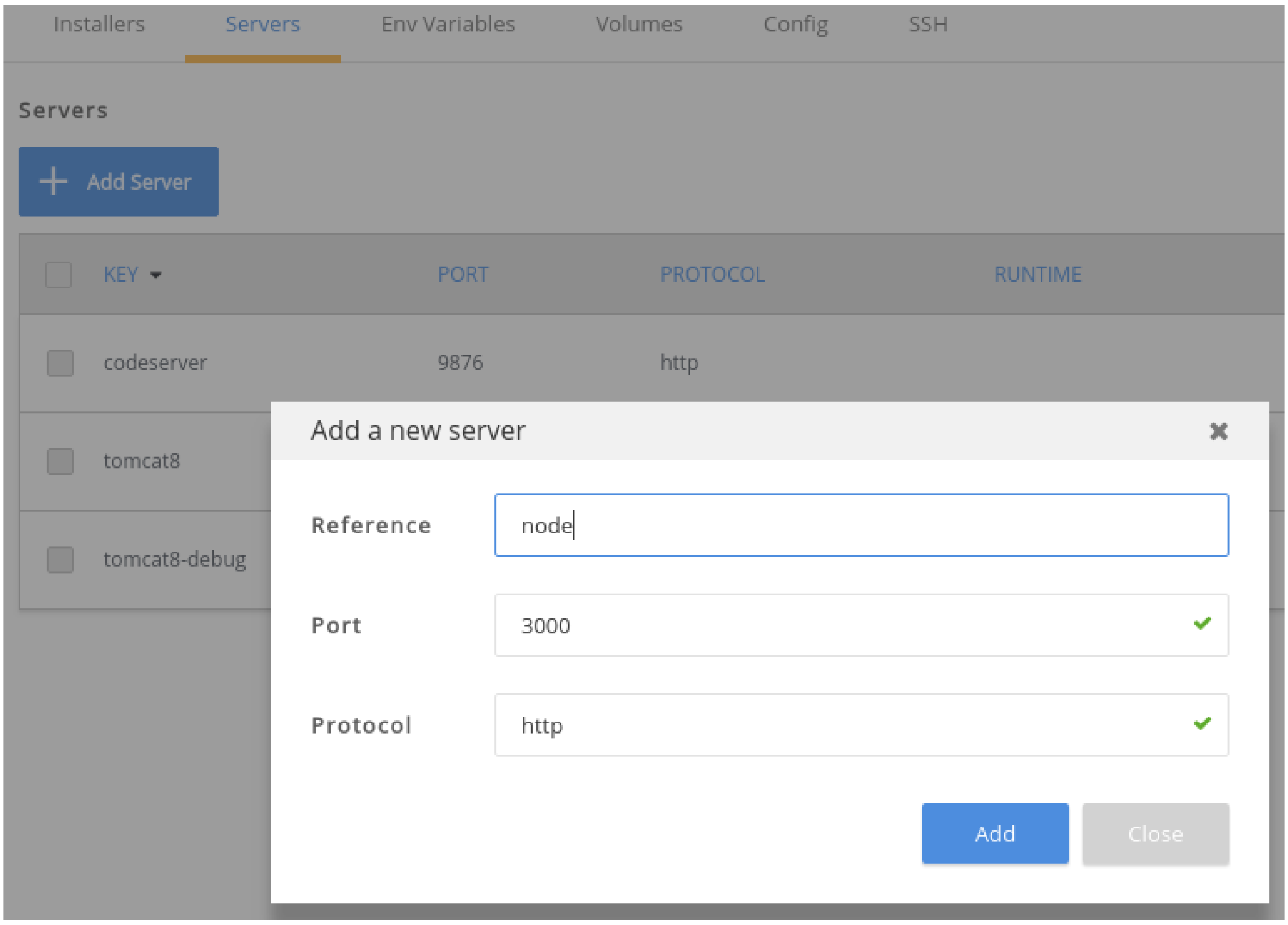
A server defines the protocol port of a process that runs in a machine. It has a name, path, and attributes. The path defines the base path of the service that is used by the server. Attributes are optional and can be used to tune the server or for identification. You can add a server when you need to access a process in your workspace machine.

To add a server, use the **User Dashboard** or edit the workspace machine configuration YAML file.

The following is an example of the YAML file:

```
"node": {
    "port": "3000",
    "protocol": "http",
    "path": "/",
    "attributes": {}
}
```

The following is an example of the **User Dashboard**:



KEY	PORT	PROTOCOL	RUNTIME
codeserver	9876	http	
tomcat8			
tomcat8-debug			

Reference	node
Port	3000
Protocol	http

NOTE

If your workspace is running, saving a new server restarts the workspace.

8.14.2. Preview URLs

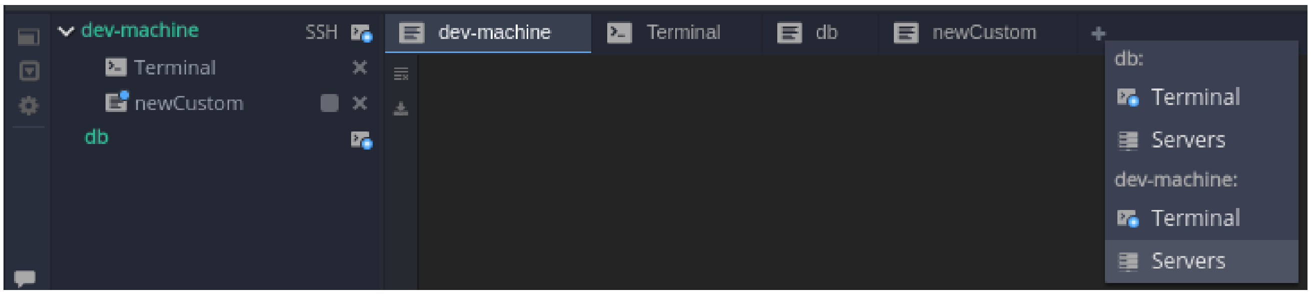
Adding the server with port 3000 does not mean you can use this port to access a server. Each server is assigned a URL when a workspace is running.

- On Docker, port 3000 is published to a random port from the ephemeral port range from 32768 to 65535. The server URLs change every time you start a workspace.
- On OpenShift, a route bound to a service is created. Routes are persistent URLs.

8.14.3. Getting preview URLs

In this example, you added a server with port 3000 and started a workspace. The following are ways to get the server's preview URL:

- Use a macro command.
- In the IDE, Click the **+** icon in the bottom panel under the editor.



- In the **User Dashboard**, click **Workspaces** > **YourWorkspace** > **Servers**.

You can also see internal server URLs. Internal servers launch when the workspace container or pod is available.

8.14.4. Exposing internal servers

To access a port internally within a workspace, expose it internally, but do not make it publicly accessible. For example, a database server is exposed only for the web application and because of security concerns, it should not be accessible publicly. The database server is exposed as internal.

To expose a server as internal, add the corresponding attribute into the server configuration YAML file:

```
"db": {
    "port": "3200",
    "protocol": "tcp",
    "attributes": {
        "internal": "true"
    }
}
```

The application is able to fetch the database URL from the workspace runtime and access the database. The database URL is not accessible publicly from the browser.

8.14.5. Exposing secure servers

Secure servers are exposed publicly but access is restricted only for users who have permissions to the workspace. The authentication proxy is set up as the exposed server and the machine token is required to request it.

To expose a server as secure, add the corresponding attributes into the server configuration YAML file:

```
"tooling": {
    "port": "4921",
    "protocol": "http",
    "attributes": {
        "secure": "true",
        "unsecuredPaths": "/liveness",
        "cookiesAuthEnabled": "true"
    }
}
```

The following describes the attributes:

`secure`

Indicates whether the server is exposed as secure. The default value is **`false`**.

`unsecuredPaths`

Configures the secure servers. It contains a comma-separated list of URLs that are considered non-secure on the given server and can be accessible without a token. It may be needed when the server provides any public APIs. The API endpoint for health checks is an example.

`cookiesAuthEnabled`

Indicates whether cookies should be searched for a token. By default, it is disabled. You can enable this attribute if you are sure that servers cannot be attacked by Cross-Site Request Forgery (CSRF) or have special protection from it.

NOTE

This is in the beta phase and disabled by default. See Secure servers for information on how to enable secure servers.

8.15. INSTALLERS

8.15.1. What are installers?

Installers are scripts that are added into machines in a runtime. Once running, installers:

1. Prepare the environment and download dependencies for a particular software or tool.

2. Install chosen software and dependencies.

3. Launch software and tools with particular arguments and modes that provide extra functionality for a workspace.

Installers are typically language servers and tools that provide features such as SSH access to a workspace machine. You can find a complete list of available installers in the **Workspace details > Installers** tab.

The following is an example of installers:

```
"installers": [
    "org.eclipse.che.exec",
    "org.eclipse.che.terminal",
    "org.eclipse.che.ws-agent",
    "org.eclipse.che.ssh"
        ]
```

8.15.2. How installers work

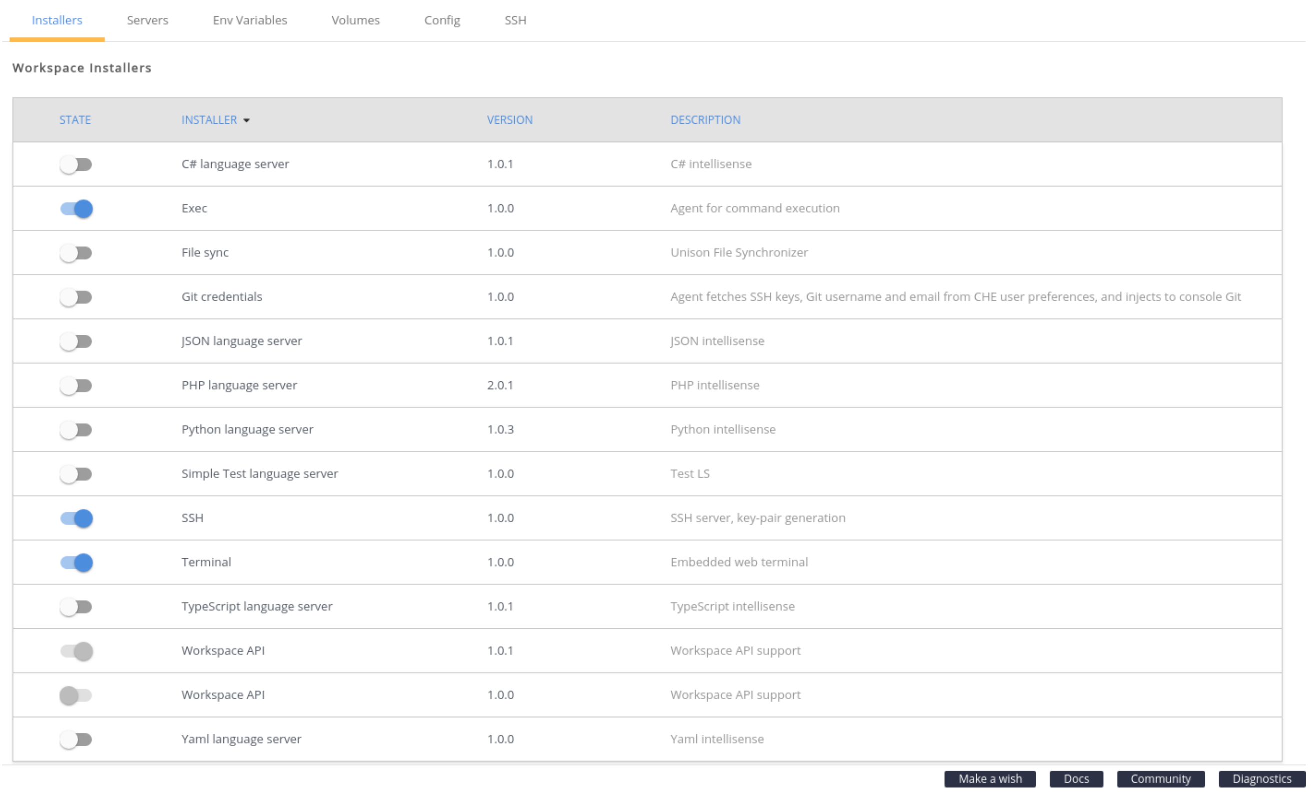
Installers are saved in a configuration file that a bootstrapper uses to execute jobs. An installer script works exactly the same way as other shell scripts in Linux. The CodeReady Workspaces server checks if the launched process is running.

Some installers activate special agents, such as the workspace, terminal, and execution agents. If a workspace agent fails to start, the workspace is treated as if it has been started but the IDE is not usable. If the execution agent fails, the **commands** widget is unavailable.

8.15.3. What happens when enabling and disabling installers

You can enable or disable installers per machine by using the **User Dashboard** or by updating the workspace machine configuration. When an installer is enabled, the bootstrapper executes an installer script after the workspace has started.

The following shows installers that are enabled and disabled:



Workspace Installers

STATE	INSTALLER	VERSION	DESCRIPTION
○	C# language server	1.0.1	C# intellisense
●	Exec	1.0.0	Agent for command execution
○	File sync	1.0.0	Unison File Synchronizer
○	Git credentials	1.0.0	Agent fetches SSH keys, Git username and email from CHE user preferences, and injects to console Git
○	JSON language server	1.0.1	JSON intellisense
○	PHP language server	2.0.1	PHP intellisense
○	Python language server	1.0.3	Python intellisense
○	Simple Test language server	1.0.0	Test LS
●	SSH	1.0.0	SSH server, key-pair generation
●	Terminal	1.0.0	Embedded web terminal
○	TypeScript language server	1.0.1	TypeScript intellisense
●	Workspace API	1.0.1	Workspace API support
○	Workspace API	1.0.0	Workspace API support
○	Yaml language server	1.0.0	Yaml intellisense

8.15.4. Troubleshooting installer failures

8.15.4.1. Permission denied failure

Installers run as if a user in the container has **`sudoers`** privileges. If the user does not have the privileges, the installer fails with **`permission denied`** issues.

This problem can occur with OpenShift when a pod is run by a user with no permissions to use **`sudo`** or to access or modify resources on the file system.

In most cases, this problem can be solved by rebuilding the base image so that it already has all of the dependencies for particular agents that an installer activates.

8.15.4.2. Permission to files and directories failures

Another possible issue can be with permissions to files and directories. For example, an installer may need to write to the user home directory.

8.15.5. Installer registry and REST API

CodeReady Workspaces installers are stored in the Installer Registry. They can be viewed and edited through the Installer Registry REST API:

Path	HTTP Method	Description
`/installer`	GET	Get installers
`/installer/{id}/version`	GET	Get versions of installers by given id
`/installer/orders`	GET	Get installers, ordered by their dependencies
`/installer/`	POST	Add installer to the registry
`/installer/{key}`	PUT	Update installer in the registry
`/installer/{key}`	DELETE	Remove installer from the registry

8.16. VOLUMES

8.16.1. Default volumes for workspace containers

By default, workspace containers start with a default volume and have a minimum of one PVC that is located in the **`/projects`** directory.

Workspace projects are physically located in the **`/projects`** directory. When a workspace stops, the machines are destroyed, but the volumes persist.

8.16.2. Adding volumes

In order for your data to persist for a local Maven repository, the **`node_modules/`** directory, Ruby gems, or the **`authorized_keys`** file for SSH connections, your workspace will need additional volumes. Each machine can add as many volumes as the underlying infrastructure can support. OpenShift may impose a limit on the number of volumes.

You can add volumes either by using the **User Dashboard** or by updating the machine configuration. The following is an example of the configuration file:

```
"volumes": {
  "myvolume": {
    "path": "/absolute/path/in/workspace"
  }
}
```

To avoid failures when updating the workspace configuration using REST APIs:

- Use an absolute path.
- The name and path cannot contain special characters, including dashes (`-`) or underscores (_).

NOTE

To allow machines to share the same volume, create a volume for each machine with an identical name.

8.16.3. Configuring workspaces

To configure workspaces on the OpenShift and Kubernetes infrastructure as ephemeral, set the **`persistVolumes`** attribute to **`false`** in the workspace configuration.

The following is an example of the configuration file:

```
"attributes": {
  "persistVolumes": "false"
}
```

In this case, regardless of the PVC strategy, all volumes would be created as **`emptyDir`** for the given workspace. When a workspace pod is removed for any reason, the data in the **`emptyDir`** volume is deleted forever.

8.17. ENVIRONMENT VARIABLES

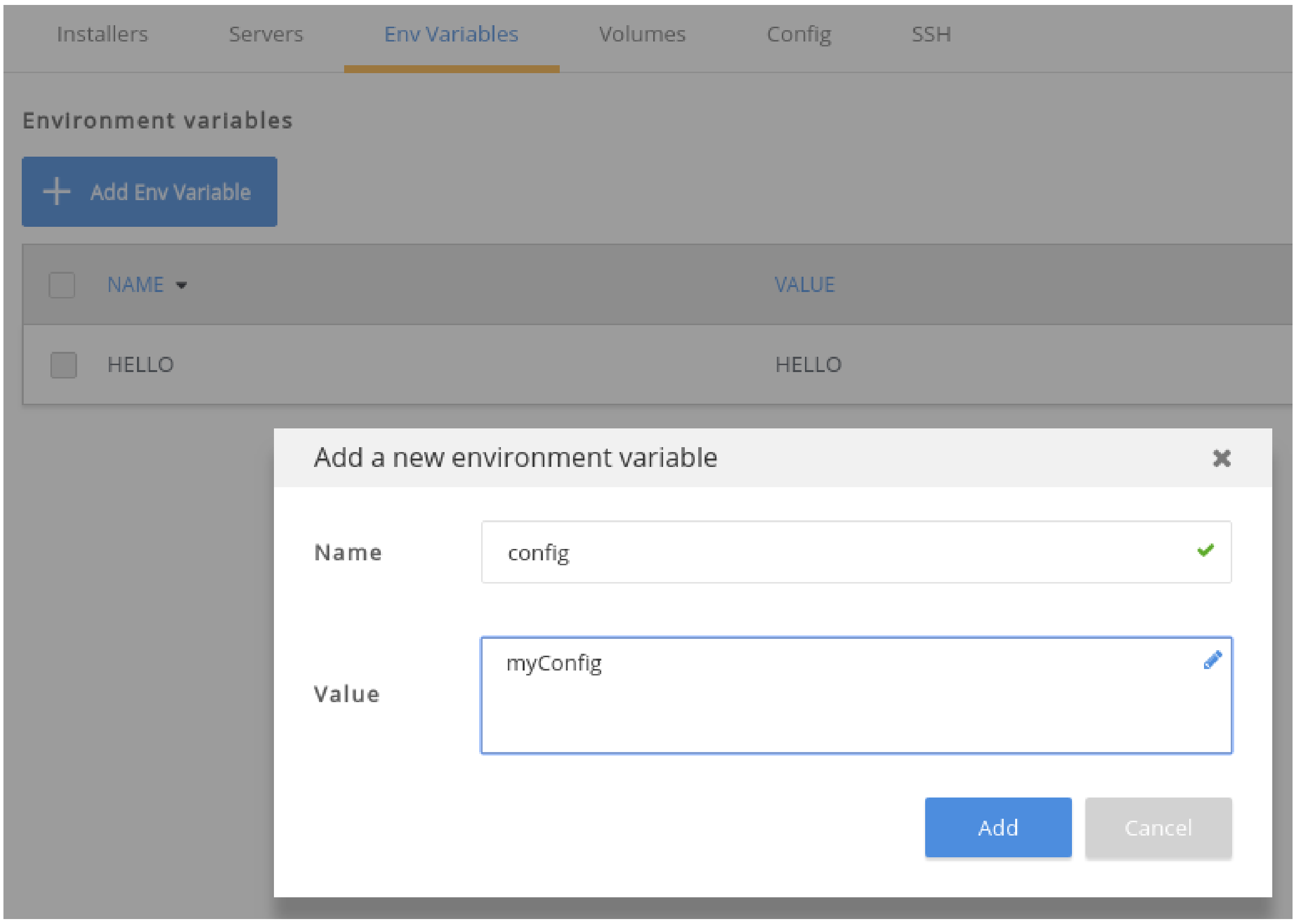
Environment variables are defined per machine. Depending on the infrastructure, they are added either to the container or the Kubernetes pod definition. You can add, edit, and remove environment variables either in the **User Dashboard** or directly in the workspace machine configuration.

The following is an example of an environment variable:

```
"env": {
  "key": "value"
    }
```

You can use environment variables in applications running in a workspace, in commands, and in the terminal. The CodeReady Workspaces server also adds some environment variables that a user does not control, although they are available to use. For example, they can be used as an API endpoint or workspace ID.

The following shows how to add a new environment variable:



8.18. PROJECTS

8.18.1. Creating projects in workspaces

Projects are always associated with a workspace and saved in a workspace configuration.

The following is an example of the project YAML file:

```
"projects": [
   {
     "description": "A basic example using Spring servlets. The app
returns values entered into a submit form.",
     "source": {
       "location": "https://github.com/che-samples/web-java-spring.git",
       "type": "git",
       "parameters": {}
     },
     "links": [],
     "mixins": [],
     "problems": [],
     "name": "web-java-spring",
     "type": "maven",
     "path": "/web-java-spring",
     "attributes": {}
   }
 ]
```

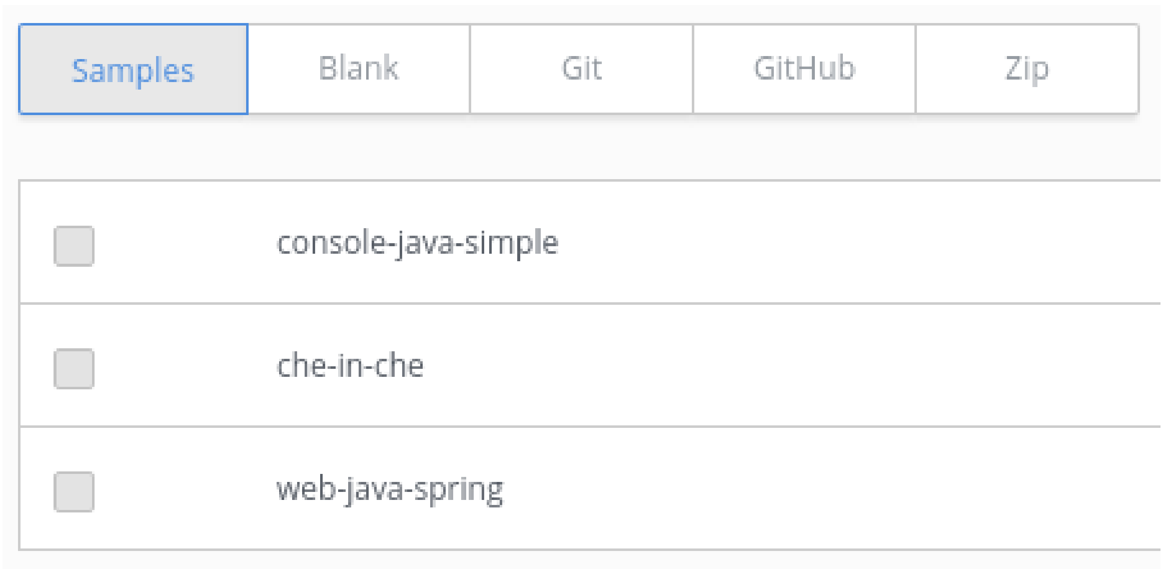
Once a project is saved into a workspace configuration, the IDE checks if the project exists on a file system. Use the **`source.location`** URL to import projects that do yet exist on the file system. This happens during the IDE initialization stage.

You can add the following projects:

- Git projects
- remotely hosted archives
- GitHub projects
- example projects provided by CodeReady Workspaces

Project import tools can be found on the **User Dashboard** when you are creating a new workspace or editing an existing workspace in the IDE. Project import tools can also be found in the **Workspace** menu.

The following shows example projects:



8.18.2. Defining project types

Plug-in developers can define their own project types. Since project types trigger certain behaviors within the IDE, the construction of the projects is important to understand.

- A project type is defined as one primary type and zero or more mixin types.
  - A primary project type is one where the project is editable, buildable, and runnable.
  - A mixin project type defines additional restrictions and behaviors of the project, but it cannot be a primary project type by itself.

  The collection of primary and mixin types for a single project defines the aggregate set of attributes that will be stored as meta data within the project.

- Project types describe different aspects of a project, such as:
  - the types of source files inside
  - the structure of the explorer tree

    - the way in which a command is executed
    - associated workflows
    - which plug-ins must be installed

- A project defines a set of attributes. The attributes of a project can be mandatory or optional. Attributes that are optional can be dynamically set at runtime or during configuration.
- Sub-projects may have different project types than their parents. Modules may physically exist within the tree structure of the parent. For example, subdirectories exist within the tree structure of the parent. Also, modules may physically exist outside the tree structure of the parent, such as when the parent is a soft link to the module project.

8.18.3. Creating a sub-project

A sub-project is a portion of a project that can have sets of commands run against it where the sub-directory is treated as the root working directory. Sub-projects make it possible to organize a single repository into multiple, independently buildable, and runnable units.

To create a module, right-click on a directory in the IDE explorer tree and select **Convert to Project**. You can then execute commands directly against this sub-project.

8.18.4. Navigating the project tree

You can step into or out of the project tree. When you step into a directory, that directory is set as the project tree root and the explorer refreshes the view. All commands are then executed against this directory root.

8.19. TROUBLESHOOTING FAILURES IN STARTING THE WORKSPACE

Failures to start a workspace may be caused by the following factors:

- Incorrect environment recipe
- Restrictive network settings

8.19.1. Incorrect environment recipes

When a workspace is starting, an environment recipe is sent to Docker or to the OpenShift API. The CodeReady Workspaces server then listens to events provided by the given infrastructure. The CodeReady Workspaces server expects a running Docker container or an OpenShift pod. The server fails to start an environment and consequently the starting of the workspace fails if the infrastructure is unable to create and start a container or a pod from the provided recipe.

A recipe can be incorrect due to the following reasons:

- The Docker build fails with the provided **`Dockerfile`**. This can be because of a broken **`Dockerfile`** or because of CodeReady Workspaces. If the Docker build in CodeReady Workspaces does not support context, consider editing the Docker recipe locally to ensure that it is a valid **`Dockerfile`**. Add or copy resources into an image locally on your machine, push the image to a registry, such as DockerHub, and use the resulting images in the recipe.
- CodeReady Workspaces does not support certain Docker Compose syntax. Ensure that the **`Composefile`** is supported by CodeReady Workspaces.

- Installing packages in your **`Dockerfile`** instructions can take time. This may be influenced by network settings.

8.19.1.1. Viewing logs from a failed workspace start

No installer logs are shown when a workspace fails to start because its container or pod are not launched. In most cases, only logs from infrastructure and image pull and build are shown. Analyse these logs to find the problem. The CodeReady Workspaces server also produces logs that are helpful in debugging the problem.

8.19.2. Restrictive network settings

The CodeReady Workspaces server and agents, which run in a workspace container or pod, and the user's browser communicate with each other. Firewall, filtered ports, and other network restrictions may cause trouble when starting a workspace.

A workspace is considered to be in a **`RUNNING`** state after the CodeReady Workspaces server verifies that the workspace agent is up. The workspace agent also tries to reach the CodeReady Workspaces server. All this happens in separate containers or pods, and the user's browser is not yet involved. The **`workspace started by user $userName`** message in the CodeReady Workspaces server logs indicates the following:

- The workspace container or pod is up.
- The workspace agent has successfully started.
- The CodeReady Workspaces server can reach it.

8.19.2.1. Troubleshooting network setting when workspace agent cannot be reached

An error message saying that the IDE cannot be initialized indicates that the client (browser) cannot reach the workspace agent. This is caused by the CodeReady Workspaces server using an internal IP address to reach the workspace agent, while you are accessing the workspace from a machine that is located on a different network. To confirm this, open the browser developer console and check failed requests. The failed requests are to **`project`** and **`project-type`** API.

To access a workspace from a different network than the one on which the CodeReady Workspaces server is running, enable access to the ephemeral port range on the CodeReady Workspaces server network.

8.19.3. Failure in bootstrapping

When a workspace starts, the CodeReady Workspaces server creates and starts a container or a pod or a set of containers and pods as per the environment recipe. After the container or pod is running, a bootstrapping process begins - the bootstrapper binary is downloaded and launched. If the server logs show bootstrapping failures, or you do not see any output in the **Machine** tab of the **Workspaces** view, the reason is that bootstrapper is not downloaded. The following are possible the reasons for the bootstrapper download failure:

- Network conditions (for example, firewall restrictions).
- Incorrect bootstrapper binary URL that the CodeReady Workspaces server uses (often reproduced when deploying to OpenShift and missing necessary environment variables).

To work around the problem, download the bootstrapper binary manually. On OpenShift, access the pod on the command line (shell or the terminal in the web console) and run the following commands:

```
$ cd /tmp/bootstrapper
$ ls -la  1
$ curl ${CHE_URL}/agent-binaries/linux_amd64/bootstrapper/bootstrapper
```

1 to check for the existence of the bootstrapper binary

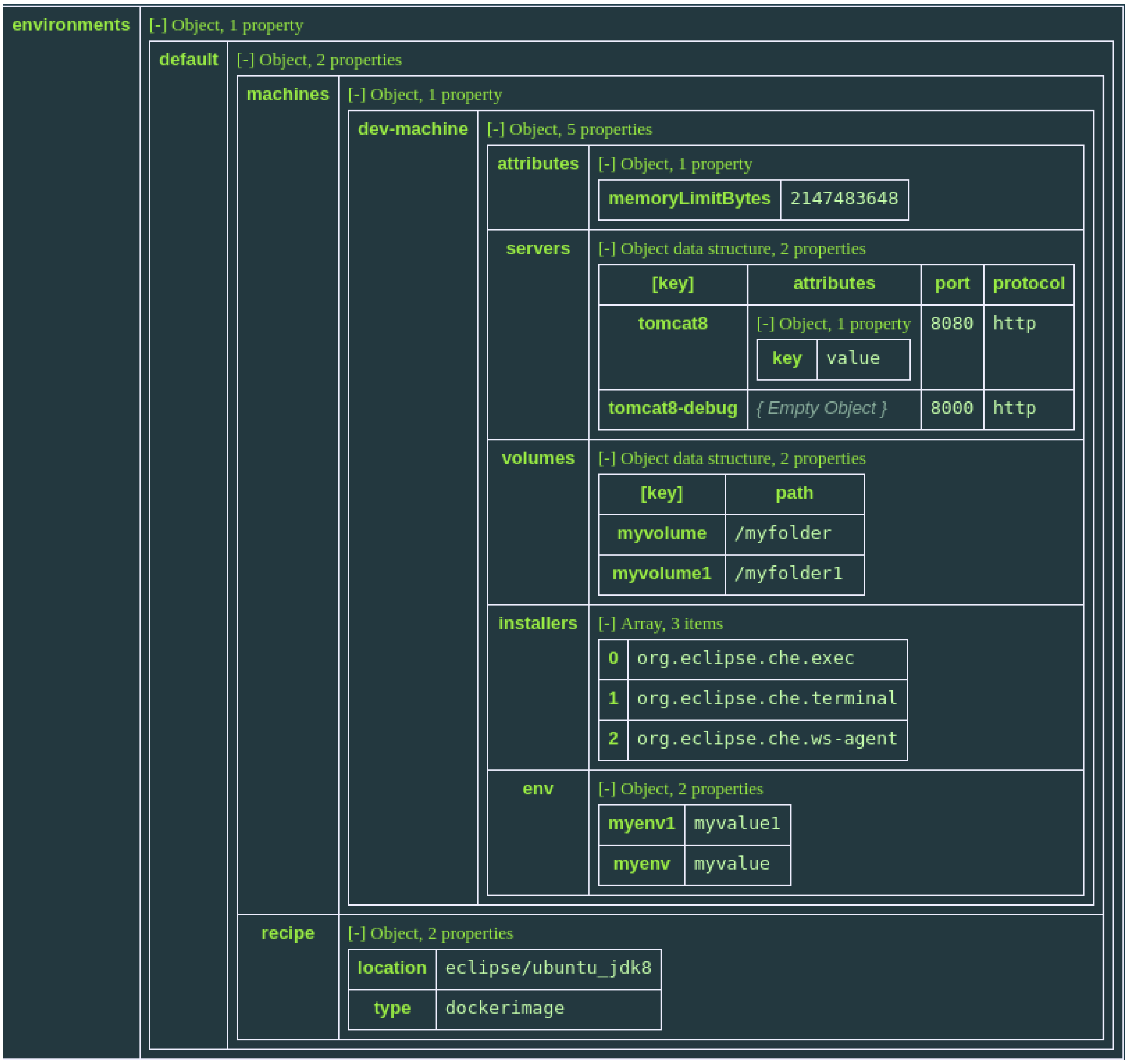
To prevent the **`curl`** command from failing, unblock port **`80`** on your network. On OpenShift with **`https`** routes, unblock port **`443`**.

8.20. WORKSPACE DATA MODEL

The following table lists the data types and their description.

Data Types	Description
environments: Map<String, getEnvironments>	Workspace environment variables. A workspace can have multiple environment variables.
defaultEnv: STRING	A workspace must have a default environment.
projects: []	Projects associated with a workspace.
commands: []	Commands associated with a workspace.
name: STRING	Workspace name that has to be unique in a namespace.
links: []	-

8.20.1. Environment recipes



- **environments** [-] Object, 1 property
  - **default** [-] Object, 2 properties
    - **machines** [-] Object, 1 property
      - **dev-machine** [-] Object, 5 properties
        - **attributes** [-] Object, 1 property

memoryLimitBytes	`2147483648`

        - **servers** [-] Object data structure, 2 properties

[key]	attributes	port	protocol
tomcat8	[-] Object, 1 property **key** `value`	`8080`	`http`
tomcat8-debug	*{ Empty Object }*	`8000`	`http`

        - **volumes** [-] Object data structure, 2 properties

[key]	path
myvolume	`/myfolder`
myvolume1	`/myfolder1`

        - **installers** [-] Array, 3 items

0	`org.eclipse.che.exec`
1	`org.eclipse.che.terminal`
2	`org.eclipse.che.ws-agent`

        - **env** [-] Object, 2 properties

myenv1	`myvalue1`
myenv	`myvalue`

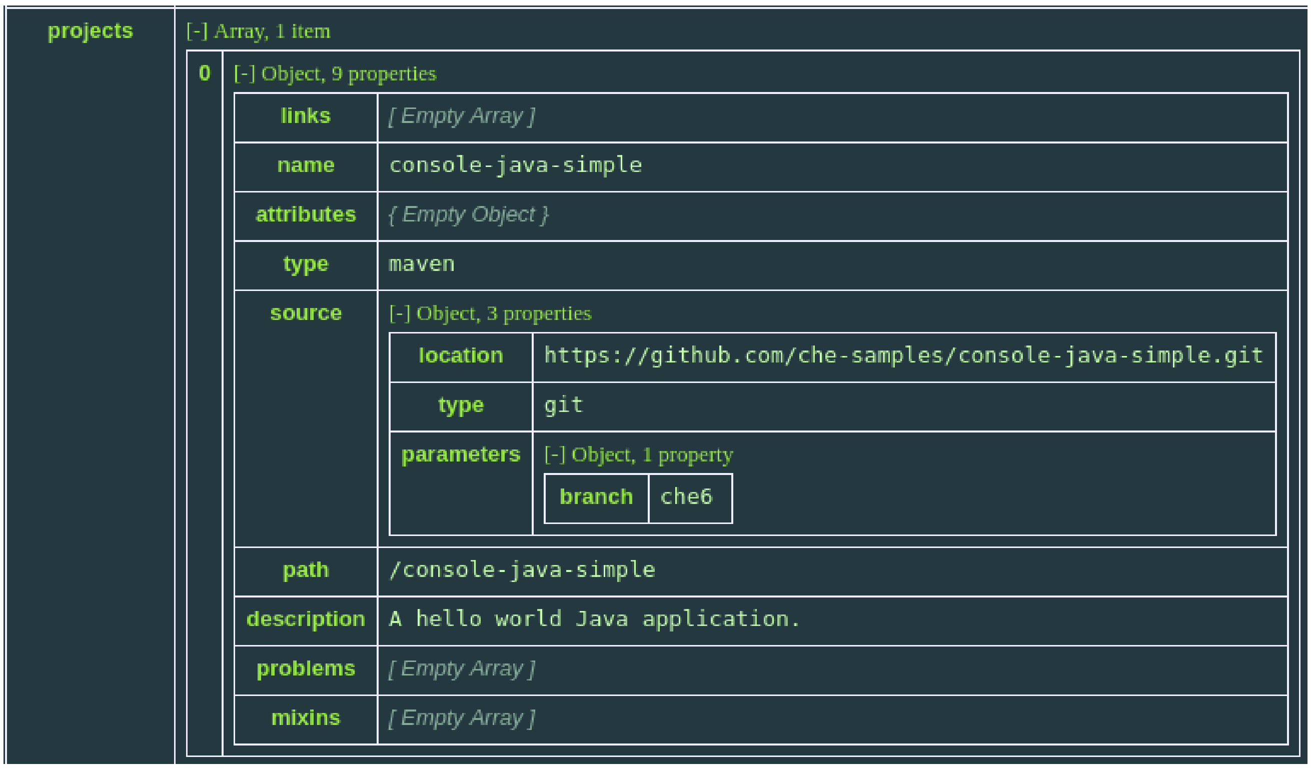
    - **recipe** [-] Object, 2 properties

location	`eclipse/ubuntu_jdk8`
type	`dockerimage`

For recipe types of **`dockerfile`**, **`compose`**, or **`openshift`**, content, not location, is specified.

```
"recipe": {
  "type": "compose",
  "content": "services:\n db:\n  image: eclipse/mysql\n  environment:\n
MYSQL_ROOT_PASSWORD: password\n   MYSQL_DATABASE: petclinic\n
MYSQL_USER: petclinic\n   MYSQL_PASSWORD: password\n  mem_limit:
1073741824\n dev-machine:\n  image: eclipse/ubuntu_jdk8\n  mem_limit:
2147483648\n  depends_on:\n    - db",
  "contentType": "application/x-yaml"
}
```

8.20.2. Projects



```
projects    [-] Array, 1 item
  0         [-] Object, 9 properties
    links         [ Empty Array ]
    name          console-java-simple
    attributes    { Empty Object }
    type          maven
    source        [-] Object, 3 properties
      location      https://github.com/che-samples/console-java-simple.git
      type          git
      parameters    [-] Object, 1 property
        branch        che6
    path          /console-java-simple
    description   A hello world Java application.
    problems      [ Empty Array ]
    mixins        [ Empty Array ]
```

The project object structure has the **`source.location`** and **`source.type`** parameters. There are two importer types: **`git`** and **`zip`**. New location types can be provided by custom plugins, such as **`svn`**.

Incorrectly configured projects or projects missing sources are marked with error codes and messages explaining the error. In the example above, the project does not have errors and mixins.

A mixin adds additional behaviors to the project, the IDE panels, and menus. Mixins are reusable across any project type. To define the mixins to add to a project, specify an array of strings, with each string containing the identifier for the mixin.

Mixin ID	Description
`git`	Initiates the project with a Git repository. Adds Git-menu functionality to the IDE. To add a mixin to the project, create a new project and then initialize a Git repository.
`pullrequest`	Enables pull-request workflow where a server handles the local and remote branching, forking, and pull-request issuance. Pull requests generated from within the server have another Factory placed into the comments of pull requests that a PR reviewer can consume. Adds contribution panel to the IDE. Set this mixin to use attribute values for **`project.attributes.local_branch`** and **`project.attributes.contribute_to_branch`**.

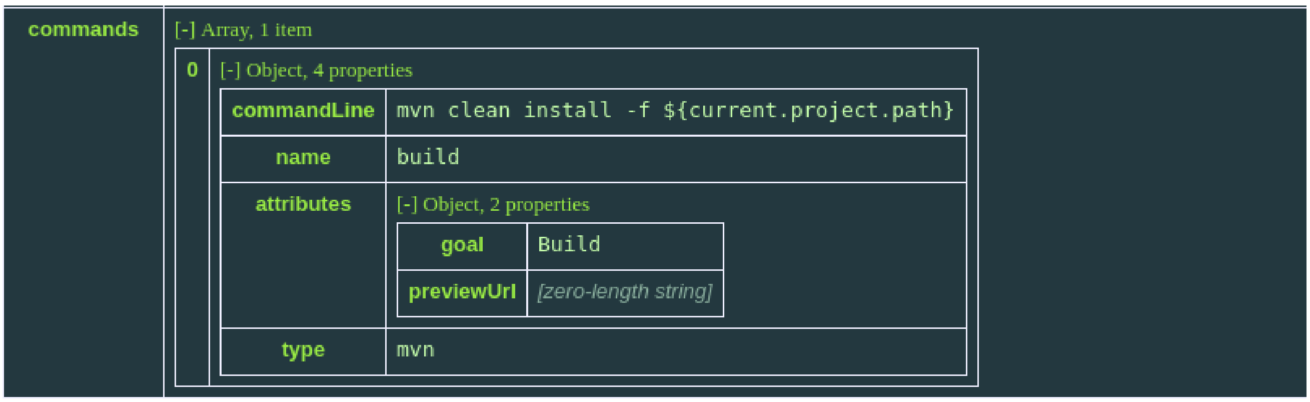
The **`pullrequest`** mixin requires additional configuration from the **`attributes`** object of the project.

The **`project`** object can include **`source.parameters`**, which is a map that can contain additional parameters. Example: related to project importer.

Parameter name	Description
`skipFirstLevel`	Used for projects with type `zip`. When value is 'true', the first directory inside ZIP will be omitted.

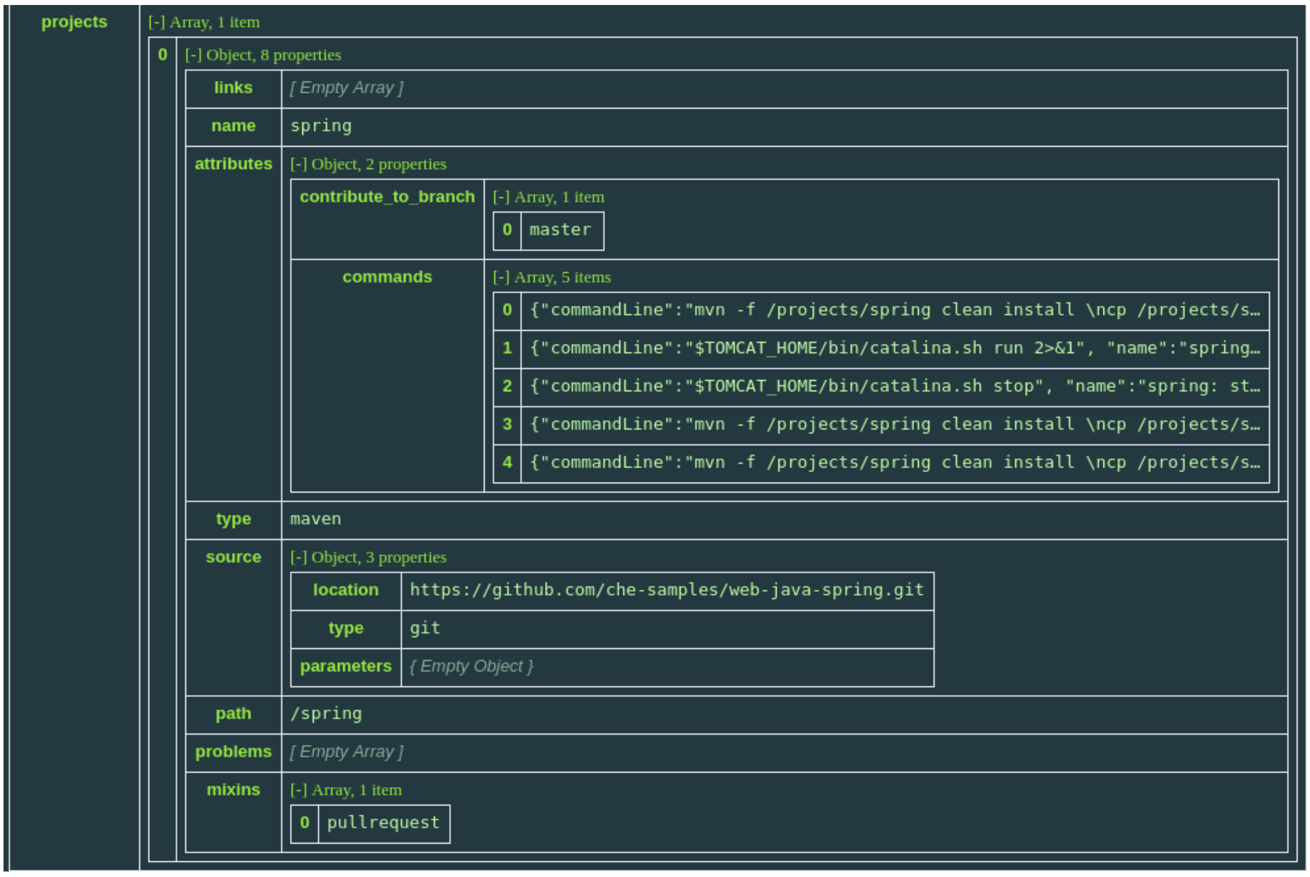
8.20.3. Commands

Commands can be both tied to a workspace and an individual project. In the example below, a command is saved to workspace configuration.



```
commands     [-] Array, 1 item
  0          [-] Object, 4 properties
    commandLine   mvn clean install -f ${current.project.path}
    name          build
    attributes    [-] Object, 2 properties
      goal          Build
      previewUrl    [zero-length string]
    type          mvn
```

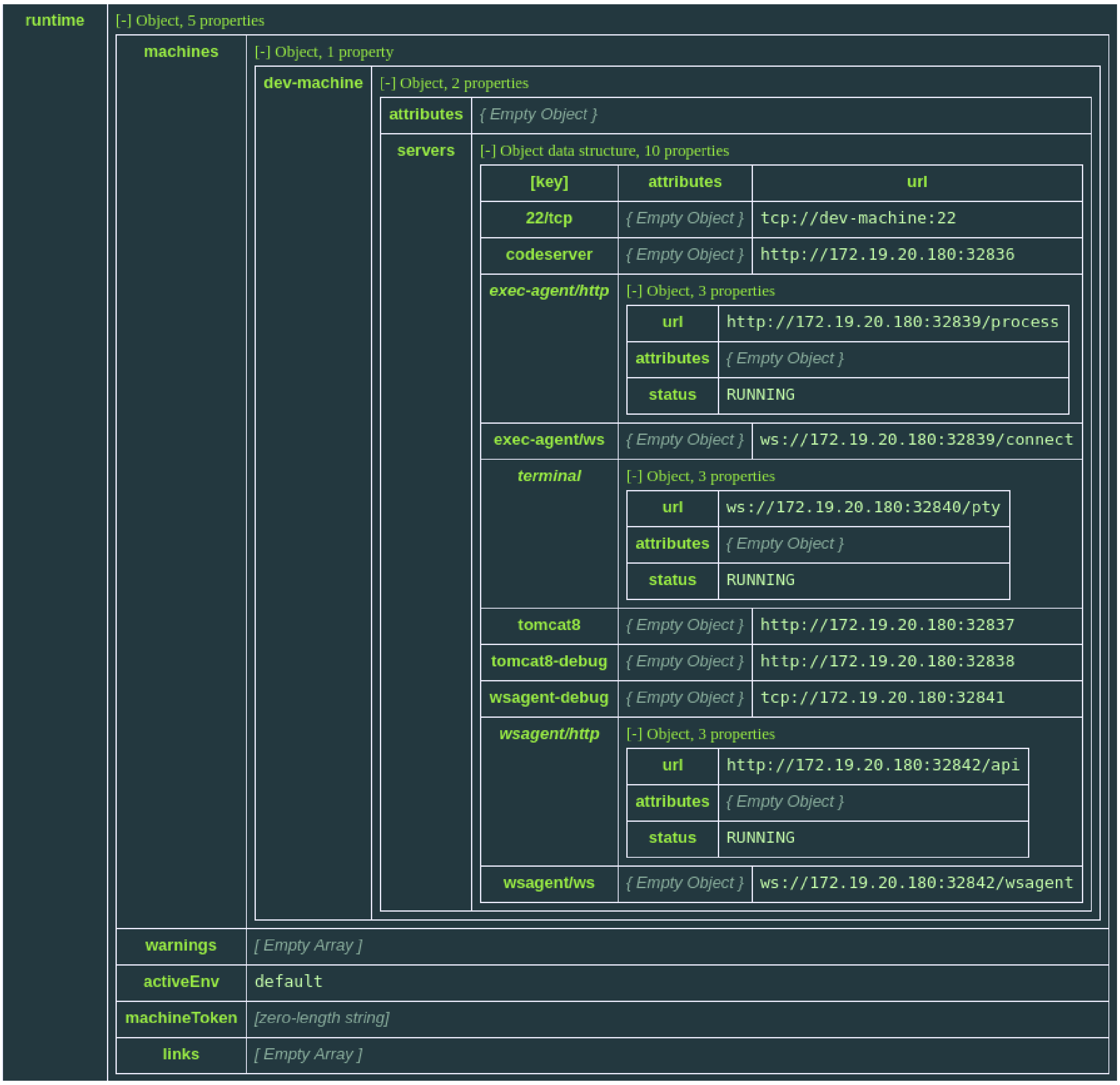
The followling image shows ways to save commands in the project configuration.



```
projects     [-] Array, 1 item
  0          [-] Object, 8 properties
    links         [ Empty Array ]
    name          spring
    attributes    [-] Object, 2 properties
      contribute_to_branch   [-] Array, 1 item
        0   master
      commands               [-] Array, 5 items
        0   {"commandLine":"mvn -f /projects/spring clean install \ncp /projects/s…
        1   {"commandLine":"$TOMCAT_HOME/bin/catalina.sh run 2>&1", "name":"spring…
        2   {"commandLine":"$TOMCAT_HOME/bin/catalina.sh stop", "name":"spring: st…
        3   {"commandLine":"mvn -f /projects/spring clean install \ncp /projects/s…
        4   {"commandLine":"mvn -f /projects/spring clean install \ncp /projects/s…
    type          maven
    source        [-] Object, 3 properties
      location      https://github.com/che-samples/web-java-spring.git
      type          git
      parameters    { Empty Object }
    path          /spring
    problems      [ Empty Array ]
    mixins        [-] Array, 1 item
      0   pullrequest
```

8.20.4. Runtime

A runtime object is created when a workspace is in a running state. Runtime returns server URLs, internal or external, depending on the server configuration. Interested clients, like the **User Dashboard** and the IDE, use these URLs.



- **runtime** [-] Object, 5 properties
  - **machines** [-] Object, 1 property
    - **dev-machine** [-] Object, 2 properties
      - **attributes** *{ Empty Object }*
      - **servers** [-] Object data structure, 10 properties

[key]	attributes	url
22/tcp	*{ Empty Object }*	`tcp://dev-machine:22`
codeserver	*{ Empty Object }*	`http://172.19.20.180:32836`
exec-agent/http	[-] Object, 3 properties: url `http://172.19.20.180:32839/process`; attributes *{ Empty Object }*; status `RUNNING`	
exec-agent/ws	*{ Empty Object }*	`ws://172.19.20.180:32839/connect`
terminal	[-] Object, 3 properties: url `ws://172.19.20.180:32840/pty`; attributes *{ Empty Object }*; status `RUNNING`	
tomcat8	*{ Empty Object }*	`http://172.19.20.180:32837`
tomcat8-debug	*{ Empty Object }*	`http://172.19.20.180:32838`
wsagent-debug	*{ Empty Object }*	`tcp://172.19.20.180:32841`
wsagent/http	[-] Object, 3 properties: url `http://172.19.20.180:32842/api`; attributes *{ Empty Object }*; status `RUNNING`	
wsagent/ws	*{ Empty Object }*	`ws://172.19.20.180:32842/wsagent`

  - **warnings** *[Empty Array]*
  - **activeEnv** `default`
  - **machineToken** *[zero-length string]*
  - **links** *[Empty Array]*

8.21. GETTING STARTED WITH FACTORIES

A factory is a template containing configuration to automate the generation of a new workspace using a factory identifier added to the IDE URL. Factories can be used to create replicas of existing workspaces or to automate the provisioning of statically or dynamically defined workspaces.

8.21.1. Trying a factory

Clone a public workspace on **`che.openshift.io`** by clicking try a factory.

8.21.2. Using factories

Factories can be invoked from a factory URL built in multiple ways. You can replace the **`localhost:8080`** domain with the hostname of any CodeReady Workspaces installation.

Using factories on `che.openshift.io` requires the user to be authenticated. Users who are not authenticated see a login screen after they click on the factory URL. Users without an account can create one using the same dialog.

8.21.3. Invoking factories using their unique hashcodes

Format	`/f?id={hashcode}`
	`/factory?id={hashcode}`

Sample	https://localhost:8080/f?id=factorymtyoro1y0qt8tq2j

8.21.4. Invoking a named factory

Format	`/f?user={username}&name={factoryname}`
	`/factory?user={username}&name={factoryname}`

Sample	https://localhost:8080/f?user=che&name=starwars https://localhost:8080/factory?user=che&name=starwars

8.21.5. Invoking a factory for a specific git repository

Format	`/f?url={git URL}`
Sample	http://localhost:8080/f?url=https://github.com/eclipse/che http://localhost:8080/f?url=https://github.com/eclipse/che/tree/language-server http://localhost:8080/f?url=https://gitlab.com/benoitf/simple-project

Once a factory is executed, it either loads an existing workspace or generates a new one, depending on the factory configuration. The name of the workspace is determined by the factory configuration, and its name becomes a part of the URL used to access the factory. The format is: `{hostname}/{username}/{workspace}`.

8.21.6. Next steps

You have just created your first developer workspace using factories. Read on to learn more about:

- How to create factories
- Customizing factories with the factory JSON reference

8.21.7. Creating Factories

8.21.7.1. Creating a factory in the dashboard

You can create a factory based on an existing workspace. You can also create factories based on a template or by pasting in a `.factory.json` file and then generating a factory URL using the CodeReady Workspaces CLI or API. To learn more about the JSON structure and options, see Factory JSON reference.

A factory created from the dashboard is persisted on CodeReady Workspaces and retained when upgrading to a newer version.

To create a factory on the dashboard:

1. In the IDE, click **Dashboard** > **Factories** > **Create Factory**.

Sample factory: https://che.openshift.io/f?id=factorymtyoro1y0qt8tq2j.

8.21.7.2. Creating a factory in the IDE

Creating a factory in the IDE in a running workspace generates a factory to replicate that workspace including the runtime and project settings.

A factory created from the dashboard is persisted on CodeReady Workspaces and retained when upgrading to a newer version.

To create a factory in the IDE:

1. In the IDE, click **Workspace** > **Create Factory**.

Sample factory: https://che.openshift.io/f?id=factorymtyoro1y0qt8tq2j.

8.21.7.3. Creating a factory based on a repository

URL factories work with GitHub and GitLab repositories. By using URL factories, the project referenced by the URL is automatically imported.

To create a factory based on a repository:

1. Specify the repository URL. Ensure that you store the configuration in the repository.

Sample factories:

- http://che.openshift.io/f?url=https://github.com/eclipse/che
- http://che.openshift.io/f?url=https://github.com/eclipse/che/tree/language-server
- http://che.openshift.io/f?url=https://gitlab.com/benoitf/simple-project

The factory URL can include a branch or a subdirectory. Following are examples of optional parameters:

- **`?url=https://github.com/eclipse/che`** CodeReady Workspaces is imported with the **`master`** branch.
- **`?url=https://github.com/eclipse/che/tree/5.0.0`** CodeReady Workspaces is imported by using the **`5.0.0`** branch.
- **`?url=https://github.com/eclipse/che/tree/5.0.0/dashboard`** subdirectory **`dashboard/`** is imported by using the **`5.0.0`** branch.

8.21.7.3.1. Customizing URL factories

The following are two ways to customize the runtime and configuration:

Customizing only the runtime

Providing a **`.factory.json`** file inside the repository signals to CodeReady Workspaces URL factory to configure the project and runtime according to this configuration file. When a **`.factory.json`** file is stored inside the repository, any **`Dockerfile`** content is ignored because the workspace runtime configuration is defined inside the JSON file.

Customizing the `Dockerfile`

(This only works on Docker infrastructure. On recent CodeReady Workspaces versions, support of this feature may be reduced or dropped.) Providing a **`.factory.dockerfile`** inside the repository signals to the URL factory to use this **`Dockerfile`** for the workspace agent runtime. By default, imported projects are set to a **`blank`** project type. You can also set the project type in the **`.factory.json`** file or in the workspace definition that the factory inherits from.

8.21.7.4. Configuring factory policies

Policies are a way to send instructions to the automation engine about the number of workspaces to create and their meta data such as lifespan and resource allocation.

8.21.7.4.1. Setting factory limitations

Referer

CodeReady Workspacescks the hostname of the acceptor and only allows the factory to execute if there is a match.

Since and Until

Defines the time window in which the factory can be activated. For example, instructors who want to create an exercise that can only be accessed for two hours should set these properties.

8.21.7.4.2. Setting factory multiplicity

Multiplicity defines the number of workspaces that can be created from the factory.

Multiple workspaces (`perClick`)

Every click of the factory URL generates a different workspace, each with its own identifier, name, and resources.

Single workspace (`perUser`)

Exactly one workspace is generated for each unique user that clicks on the factory URL. Existing workspaces are reopened.

To learn how to configure factory policies, see the JSON reference.

8.21.7.5. Customizing the IDE

You can instruct the factory to invoke a series of IDE actions based on events in the lifecycle of the workspace.

8.21.7.6. Lifecycle Events

The lifecycle of the workspace is defined by the following events:

- **`onAppLoaded`**: Triggered when the IDE is loaded.
- **`onProjectsLoaded`**: Triggered when the workspace and all projects have been activated.
- **`onAppClosed`**: Triggered when the IDE is closed.

Each event type has a set of actions that can be triggered. There is no ordering of actions executed when you provide a list; CodeReady Workspaces asynchronously invokes multiple actions if appropriate.

8.21.7.7. Factory actions

The following is a list of all possible actions that can be configured with your factory.

Run Command

Specify the name of the command to invoke after the IDE is loaded.
Associated Event: **`onProjectsLoaded`**

Open File

Open project files in the editor. Optionally, define the line to be highlighted.
Associated Event: **`onProjectsLoaded`**

Open a Welcome Page

Customize content of a welcome panel displayed when the workspace is loaded.
Associated Event: **`onAppLoaded`**

Warn on Uncommitted Changes

Opens a warning pop-up window when the user closes the browser tab with a project that has uncommitted changes.
Associated Event: **`onAppClosed`**

To learn how to configure factory actions, see the Factory JSON reference.

8.21.7.8. Finding and replacing variables

Factories make it possible to replace variables or placeholders in the source code — used to avoid exposing sensitive information (passwords, URLs, account names, API keys) — with real values. To find and replace a value, you can use the **`run`** command during an **`onProjectsLoaded`** event. You can use **`sed`**, **`awk`**, or other tools available in your workspace environment.

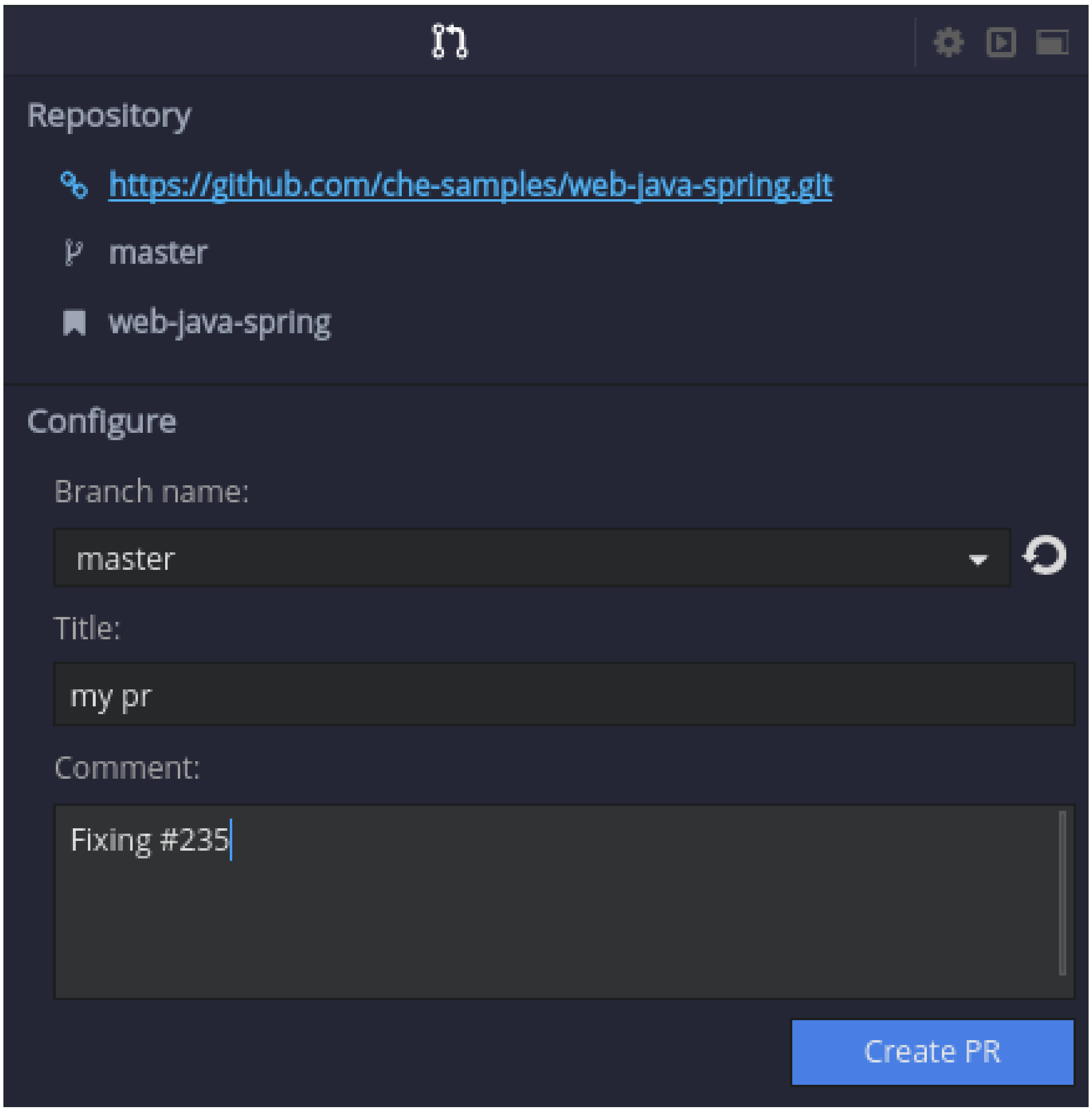
For a sample of how to configure finding and replacing a value, see the Factory JSON reference section. Alternatively, you can add IDE actions in the **Factory** tab, on the user **Dashboard**.

Use regular expressions in **`sed`**, both in find-replace and file-file type patterns.

8.21.7.9. Pull request workflow

Factories can be configured with a dedicated pull request workflow. The PR workflow handles local and remote branching, forking, and issuing the pull request. Pull requests generated from within CodeReady Workspaces have another factory placed into the comments of the pull requests that a PR reviewer can use to quickly start the workspace.

When enabled, the pull request workflow adds a contribution panel to the IDE.


Repository

https://github.com/che-samples/web-java-spring.git

master

web-java-spring

Configure

Branch name:

master

Title:

my pr

Comment:

Fixing #235

8.21.7.10. Repository badging

If you have projects in GitHub or GitLab, you can help your contributors to get started by providing them ready-to-code developer workspaces. Create a factory and add the following badge on your repositories **`readme.md`**:

```
[![Developer Workspace]
(https://che.openshift.io/factory/resources/factory-contribute.svg)](your-
factory-url)
```

8.21.7.11. Next steps

- Read about customizing factories with the Factory JSON reference.

8.21.8. Factories JSON Reference

A factory configuration is a JSON snippet either stored within CodeReady Workspaces or as a **`.factory.json`** file. You can create factories within the IDE using the CodeReady Workspaces URL syntax, within the dashboard, or on the command line with the API and CLI.

```
factory : {
  "v"         : 4.0,          (1)
  "workspace" : {},           (2)
  "policies"  : {},           (3)
  "ide"       : {},           (4)
  "creator"   : {},           (5)
}
```

1. Version of the configuration format.
2. Identical to **`workspace:{}`** object for CodeReady Workspaces.
3. (Optional) Restrictions that limit behaviors.
4. (Optional) Trigger IDE UI actions tied to workspace events.
5. (Optional) Identifying information of author.

The **`factory.workspace`** is identical to the **`workspace:{}`** object for CodeReady Workspaces and contains the structure of the workspace. To learn more about the workspace JSON object, see Workspace Data Model.

You can export workspaces and then reuse the workspace definition within a factory. workspaces are composed of the following:

- 0..n projects
- 0..n environments that contain machines to run the code
- 0..n commands to execute against the code and machines
- a type

The **`factory.policies`**, **`factory.ide`**, and **`factory.creator`** objects are unique to factories. They provide meta information to the automation engine that alter the presentation of the factory URL or the behavior of the provisioning.

8.21.8.1. Mixins

A mixin adds additional behaviors to a project as a set of new project type attributes. Mixins are reusable across any project type. To define the mixins to add to a project, specify an array of strings, with each string containing the identifier for the mixin. For example, **`"mixins" : [ "pullrequest" ]`**.

Mixin ID	Description

Mixin ID	Description
`pullrequest`	Enables pull request workflow where CodeReady Workspaces handles local and remote branching, forking, and pull request issuance. Pull requests generated from within CodeReady Workspaces have another factory placed into the comments of pull requests that a PR reviewer can consume. Adds contribution panel to the IDE. If this mixin is set, it uses attribute values for **`project.attributes.local_branch`** and **`project.attributes.contribute_to_branch`**

- The **`pullrequest`** mixin requires additional configuration from the **`attributes`** object of the project. If present, {{ site.product_mini_name }} will use the project attributes as defined in the factory. If not provided, {{ site.product_mini_name }} will set defaults for the attributes.
- Learn more about other link:TODO: link to project API docs[mixins]

8.21.8.2. Pull Request mixin attributes

Project attributes alter the behavior of the IDE or workspace.

Different CodeReady Workspaces plug-ins can add their own attributes to affect the behavior of the IDE or workspace. Attribute configuration is always optional and if not provided within a factory definition, the IDE or workspace sets it.

Attribute	Description
`local_branch`	Used in conjunction with the **`pullrequest`** mixin. If provided, the local branch for the project is set with this value. If not provided, the local branch is set with the value of **`project.source.parameters.branch`** (the name of the branch from the remote). If both **`local_branch`** and **`project.source.parameters.branch`** are not provided, the local branch is set to the name of the checked out branch.
`contribute_to_branch`	Name of the branch that a pull request will be contributed to. The value of **`project.source.parameters.branch`** is default. It is the name of the branch that this project was cloned from.

Following is a snippet that demonstrates full configuration of the **`contribution`** mixin.

```
factory.workspace.project : {
  "mixins"     : [ "pullrequest" ],

  "attributes" : {
    "local_branch"         : [ "timing" ],
    "contribute_to_branch" : [ "master" ]
  },

  "source" : {
    "type"       : "git",
    "location"   : "https://github.com/codenvy/che.git",
    "parameters" : {
      "keepVcs" : "true"
    }
  }
}
```

8.21.8.3. Policies

Following is an example of a factory policy.

```
factory.policies : {
  "referer"   : STRING,                 1
  "since"     : EPOCHTIME,              2
  "until"     : EPOCHTIME,              3
  "create"    : [perClick | perUser]    4
}
```

1. Works only for clients from a referrer.
2. Factory works only after this date.
3. Factory works only before this date.
4. Create one workpace per click, user, or account.

8.21.8.4. Limitations

You can use **`since : EPOCHTIME`**, **`until : EPOCHTIME`**, and **`referer`** as a way to prevent the factory from executing under certain conditions. **`since`** and **`until`** represent a valid time window that allows the factory to activate. The **`referrer`** checks the hostname of the acceptor and only allows the factory to execute if there is a match.

8.21.8.5. Multiplicity

Using **`create : perClick`** causes every click of the factory URL to generate a new workspace, each with its own identifier, name, and resources. Using **`create : perUser`** causes only one workspace to be generated for each unique user that clicks on the factory URL. If the workspace has previously been generated, the existing workspace is reopened.

8.21.8.6. Customizing the IDE

```
factory.ide.{event} : {                 1
```

```
  "actions" : [{}]                          2
}

factory.ide.{event}.actions : [{
  "id"         : String,                    3
  properties : {}                           4
}]
```

1 event = **`onAppLoaded`**, **`onProjectsLoaded`**, **`onAppClosed`**.

2 List of IDE actions to be executed when the event is triggered.

3 Action for the IDE to perform when the event is triggered.

4 Properties to customize action behavior.

You can instruct the factory to invoke a series of IDE actions based on events in the lifecycle of the workspace.

`onAppLoaded`

Triggered when the IDE is loaded.

`onProjectsLoaded`

Triggered when the workspace and all projects have been activated or imported.

`onAppClosed`

Triggered when the IDE is closed.

Following is an example that associates a variety of actions with all of the events.

```
"ide" : {
  "onProjectsLoaded" : {
1
    "actions" : [{
      "id" : "openFile",
2
      "properties" : {
3
        "file" : "/my-project/pom.xml"
      }
    },
    {
      "id" : "runCommand",
4
      "properties" : {
        "name" : "MCI"
5
      }
    }
  ]},
  "onAppLoaded": {
     "actions": [
        {
           "properties:{
                "greetingTitle": "Getting Started",
 6
                "greetingContentUrl": "http://example.com/README.html"  7
            },
            "id": "openWelcomePage"
        }
    ]
  },
  "onAppClosed" : {
 8
    "actions" : [{
      "id" : "warnOnClose"
 9
    }]
  }
}
```

1. Actions triggered when a project is opened.
2. Opens a file in the editor. Can add multiple.
3. The file to be opened (include project name).
4. Launch command after the IDE opens.
5. Command name.
6. Title of a **Welcome** tab.
7. HTML file to be loaded into a tab.
8. Actions to be triggered when the IDE is closed.
9. Show warning when closing a browser tab.

Each event type has a set of actions that can be triggered. There is no ordering of actions executed when you provide a list; {{ site.product_mini_name }} will asynchronously invoke multiple actions if appropriate. Some actions can be configured in how they perform and will have an associated **`properties : {}`** object.

onProjectsLoaded Event

Action	Properties?	Description
`runCommand`	Yes	Specify the name of the command to invoke after the IDE is loaded. Specify the commands in the **`factory.workspace.commands : []`** array.
`openFile`	Yes	Open project files as a tab in the editor.

onAppLoaded Event

Action	Properties?	Description
`openWelcomePage`	Yes	Customize the content of the welcome panel when the workspace is loaded. Note that browsers block http resources that are loaded into https pages.

onAppClosed Event

Action	Properties?	Description
`warnOnClose`	No	Opens a warning pop-up window when the user closes the browser tab with a project that has uncommitted changes. Requires `project.parameters.keepVcs` to be `true`.

8.21.8.7. Action: Open File

This action opens a file as a tab in the editor. You can provide this action multiple times to have multiple files open. The file property is a relative reference to a file in the project source tree. The `file` parameter is the relative path within the workspace to the file that should be opened by the editor. The `line` parameter is optional and can be used to move the editor cursor to a specific line when the file is opened. Projects are located in the `/projects/` directory of a workspace.

```
{
  "id" : "openFile",
    "properties" : {
      "file" : "/my-project/pom.xml",
      "line" : "50"
  }
}
```

8.21.8.8. Action: Find and Replace

In projects created from a factory, CodeReady Workspaces can find and replace values in the source code after it is imported into the project tree. This lets you parameterize your source code. Find and replace can be run as a **Run Command** during `onProjectsLoaded` event. You can use `sed`, `awk`, or any other tools that are available in your workspace environment.

To define a command for your workspace in `factory.workspace.workspaceConfig.commands`:

```
{
  "commandLine": "sed -i 's/***/userId984hfy6/g' /projects/console-java-
simple/README.md",
  "name": "replace",
  "attributes": {
      "goal": "Common",
      "previewUrl": ""
    },
    "type": "custom"
  }
```

In the preceding example, a named command **`replace`** is created. The command replaces each occurrence of **`*`** with the string **`userId984hfy6`** in the **`README.md`** file of the project.

Then register this command to the execution list linked to the **`onProjectsLoaded`** event. In this example, the **`replace`** command is executed after the project is imported into a workspace.

```
  "ide": {
      "onProjectsLoaded": {
        "actions": [
          {
            "properties": {
              "name": "replace"
            },
            "id": "runCommand"
          }
        ]
      }
    }
```

Use regular expressions in **`sed`**, both in find-replace and file-file type patterns.

8.21.8.9. Creator

This object has meta information that you can embed within the factory. These attributes do not affect the automation behavior or the behavior of the generated workspace.

```
  factory.creator : {
    "name"        : STRING,              (1)
    "email"       : STRING,              (2)
    "created"     : EPOCHTIME,           (3)
    "userId"      : STRING               (4)
  }
```

(1) The name of the author of this configuration file.

(2) The author’s email address.

(3) This value is set by the system.

(4) This value is set by the system.