Red Hat CodeReady Workspaces 2.0

Administration Guide

Administering Red Hat CodeReady Workspaces 2.0

Last Updated: 2020-06-04

Red Hat CodeReady Workspaces 2.0 Administration Guide

Administering Red Hat CodeReady Workspaces 2.0

Supriya Takkhi

Robert Kratky
rkratky@redhat.com

Michal Maléř
mmaler@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Yana Hontyk
yhontyk@redhat.com

Legal Notice

Abstract

Information for administrators operating Red Hat CodeReady Workspaces.

Table of Contents

CHAPTER 1. CUSTOMIZING THE DEVFILE AND PLUG-IN REGISTRIES

CodeReady Workspaces 2.0 introduces two registries: the plug-in registry and the devfile registry. They are static websites where the metadata of CodeReady Workspaces plug-ins and CodeReady Workspaces devfiles is published.

The plug-in registry makes it possible to share a plug-in definition across all the users of the same instance of CodeReady Workspaces. Only plug-ins that are published in a registry can be used in a devfile.

The devfile registry holds the definitions of the CodeReady Workspaces stacks. These are available on the CodeReady Workspaces user dashboard when selecting **Create Workspace**. It contains the list of CodeReady Workspaces technological stack samples with example projects.

The devfile and plug-in registries run in two separate pods and are deployed when the CodeReady Workspaces server is deployed (that is the default behavior of the Helm chart or the CodeReady Workspaces Operator). The metadata of the plug-ins and devfiles are versioned on GitHub and follow the CodeReady Workspaces server life cycle.

In this document, the following two ways to customize the default registries that are deployed with CodeReady Workspaces (to modify the plug-ins or devfile metadata) are described:

1. Building a custom image of the registries
2. Running the default images but modifying them at runtime

1.1. BUILDING AND RUNNING A CUSTOM REGISTRY IMAGE

This section describes the building of registries and updating a running CodeReady Workspaces server to point to the registries.

1.1.1. Building a custom devfile registry

This section describes how to build a custom devfiles registry. Following operations are covered:

- Getting a copy of the source code necessary to build a devfiles registry.
- Adding a new devfile.
- Building the devfiles registry.

Procedure

1. Clone the devfile registry repository:

   ```
   $ git clone git@github.com:redhat-developer/codeready-workspaces.git
   $ cd codeready-workspaces/dependencies/che-devfile-registry
   ```

2. In the **./che-devfile-registry/devfiles/** directory, create a subdirectory ***<devfile-name>***/ and add the **devfile.yaml** and **meta.yaml** files.

   File organization for a devfile

```
./che-devfile-registry/devfiles/
└── <devfile-name>
    ├── devfile.yaml
    └── meta.yaml
```

3. Add valid content in the **devfile.yaml** file. For a detailed description of the devfile format, see the Making a workspace portable using a Devfile section.

4. Ensure that the **meta.yaml** file conforms to the following structure:

   Table 1.1. Parameters for a devfile**meta.yaml**

Attribute	Description
description	Description as it appears on the user dashboard.
displayName	Name as it appears on the user dashboard.
globalMemoryLimit	The sum of the expected memory consumed by all the components launched by the devfile. This number will be visible on the user dashboard. It is informative and is not taken into account by the CodeReady Workspaces server.
icon	Link to an **.svg** file that is displayed on the user dashboard.
tags	List of tags. Tags usually include the tools included in the stack.

   Example devfile meta.yaml

```
displayName: Rust
description: Rust Stack with Rust 1.39
tags: ["Rust"]
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
globalMemoryLimit: 1686Mi
```

5. Build the containers for the custom devfile registry:

```
$ docker build -t my-devfile-registry .
```

1.1.2. Building a custom plug-in registry

This section describes how to build a custom plug-in registry. Following operations are covered:

- Getting a copy of the source code necessary to build a custom plug-in registry.
- Adding a new plug-in.
- Building the custom plug-in registry.

Procedure

1. Clone the plug-in registry repository:

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces/dependencies/che-plugin-registry
```

2. In the **./che-plugin-registry/v3/plugins/** directory, create new directories ***<publisher>*/*<plugin-name>*/*<plugin-version>*/** and a **meta.yaml** file in the last directory.

   File organization for a plugin

```
./che-plugin-registry/v3/plugins/
├── <publisher>
│   └── <plugin-name>
│       ├── <plugin-version>
│       │   └── meta.yaml
│       └── latest.txt
```

3. Add valid content to the **meta.yaml** file. See the “Using a Visual Studio Code extension in CodeReady Workspaces” section or the README.md file in the **eclipse/che-plugin-registry** repository for a detailed description of the **meta.yaml** file format.

4. Create a file named **latest.txt** with content the name of the latest ***<plugin-version>*** directory.

   EXAMPLE

```
$ tree che-plugin-registry/v3/plugins/redhat/java/
che-plugin-registry/v3/plugins/redhat/java/
├── 0.38.0
│   └── meta.yaml
├── 0.43.0
│   └── meta.yaml
├── 0.45.0
│   └── meta.yaml
├── 0.46.0
│   └── meta.yaml
├── 0.50.0
│   └── meta.yaml
└── latest.txt
$ cat che-plugin-registry/v3/plugins/redhat/java/latest.txt
0.50.0
```

5. Build the containers for the custom plug-in registry:

```
$ docker build -t my-devfile-registry .
```

1.1.3. Deploying the registries

Prerequisites

The **my-plug-in-registry** and **my-devfile-registry** images used in this section are built using the **docker** command. This section assumes that these images are available on the OpenShift cluster where CodeReady Workspaces is deployed.

This is true on Minikube, for example, if before running the **docker build** commands, the user executed the **eval $\{minikube docker-env}** command (or, the **eval $\{minishift docker-env}** command for Minishift).

Otherwise, these images can be pushed to a container registry (public, such as **quay.io**, or the DockerHub, or a private registry).

1.1.3.1. Deploying registries in OpenShift

Procedure

A Helm chart for the plug-in registry is available in the **/kubernetes/che-plugin-registry/** directory of the GitHub repository.

1. To deploy the plug-in registry using the Helm chart, run the following command:

```
NAMESPACE=<namespace-name>  1
DOMAIN=<kubernetes-cluster-domain> 2
IMAGE="my-plug-in-registry"
helm upgrade --install che-plugin-registry \
 --debug \
 --namespace $\{NAMESPACE} \
 --set global.ingressDomain=$\{DOMAIN} \
 --set chePluginRegistryImage=$\{IMAGE} \
 --set chePluginRegistryImagePullPolicy="IfNotPresent" \
 ./kubernetes/che-plugin-registry/
```

   1 The default CodeReady Workspaces namespace is **workspaces**.

   2 On Minikube, use **$(minikube ip).mycluster.mycompany.com**

2. The devfile registry also has a Helm chart in the **deploy/kubernetes/che-devfile-registry/** directory of the GitHub repository. To deploy it, run the command:

```
NAMESPACE=<namespace-name>  1
DOMAIN=<kubernetes-cluster-domain> 2
IMAGE="my-devfile-registry"
helm upgrade --install che-devfile-registry \
 --debug \
 --namespace $\{NAMESPACE} \
 --set global.ingressDomain=$\{DOMAIN} \
 --set cheDevfileRegistryImage=$\{IMAGE} \
 --set cheDevfileRegistryImagePullPolicy="IfNotPresent" \
 ./deploy/kubernetes/che-devfile-registry/
```

   1 The default CodeReady Workspaces namespace is **workspaces**.

   2 On Minikube, use **$(minikube ip).mycluster.mycompany.com**

3. The Helm chart creates a pod, a service, and an Ingress. To get them, use **app=che-plugin-registry** (or **app=che-plugin-registry** for the devfile registry).

```
$ oc get -o custom-columns=TYPE:.kind,NAME:.metadata.name \
  -l app=che-plugin-registry pod,svc,ingress
TYPE      NAME
Pod       che-plugin-registry-5c7cd8d5c9-zlqlz
Service   che-plugin-registry
Ingress   che-plugin-registry
```

4. To verify that the new plug-in is correctly published to the plug-in registry, make a request to the registry path **/v3/plugins/index.json** (or **/devfiles/index.json** for the devfile registry).

```
$ URL=$(oc get -o 'custom-columns=URL:.spec.rules[0].host' \
  -l app=che-plugin-registry ingress --no-headers)
$ INDEX_JSON=$(curl -sSL http://$\{URL}/v3/plugins/index.json)
$ echo $\{INDEX_JSON} | grep -A 4 -B 5 "\"name\":\"my-plug-in\""
,\{
 "id": "my-org/my-plug-in/1.0.0",
 "displayName":"This is my first {prod-short} plug-in",
 "version":"1.0.0",
 "type":"VS Code extension",
 "name":"my-plug-in",
 "description":"This plugins shows that we are able to add plugins to the registry...",
 "publisher":"my-org",
 "links": \{"self":"/v3/plugins/my-org/my-plug-in/1.0.0" }
}
--
--
,\{
 "id": "my-org/my-plug-in/latest",
 "displayName":"This is my first {prod-short} plug-in",
 "version":"latest",
 "type":"VS Code extension",
 "name":"my-plug-in",
 "description":"This plugins shows that we are able to add plugins to the registry...",
 "publisher":"my-org",
 "links": \{"self":"/v3/plugins/my-org/my-plug-in/latest" }
}
```

5. Verify that the CodeReady Workspaces server points to the URL of the registry. To do this, compare the value of the **CHE_WORKSPACE_PLUGIN__REGISTRY__URL** parameter in the **workspaces** ConfigMap (or **CHE_WORKSPACE_DEVFILE__REGISTRY__URL** for the devfile registry):

```
$ oc get \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_PLUGIN__REGISTRY__URL']" \
  --no-headers cm/che
URL
http://che-plugin-registry-che.192.168.99.100.mycluster.mycompany.com/v3
```

with the URL of the Ingress:

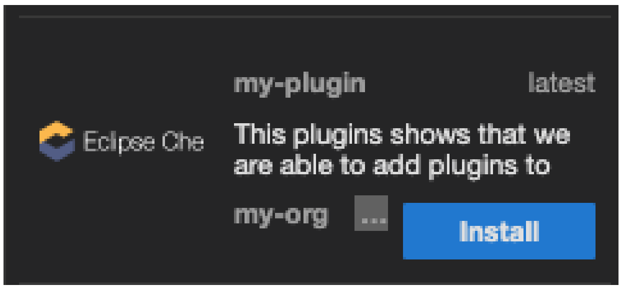
```
$ oc get -o 'custom-columns=URL:.spec.rules[0].host' \
  -l app=che-plugin-registry ingress --no-headers
che-plugin-registry-che.192.168.99.100.mycluster.mycompany.com
```

6. If they do not match, update the ConfigMap and restart the CodeReady Workspaces server.

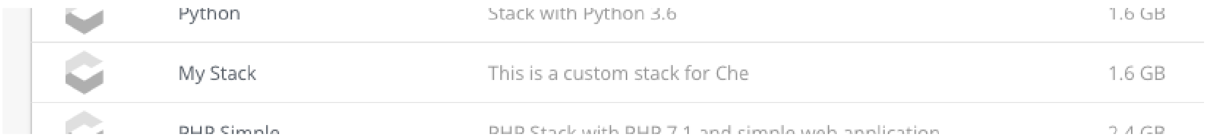
■

```
$ oc edit cm/che
(...)
$ oc scale --replicas=0 deployment/che
$ oc scale --replicas=1 deployment/che
```

When the new registries are deployed and the CodeReady Workspaces server is configured to use them, the new plug-ins are available in the **Plugin** view of a workspace.



The new stacks are displayed in the **New Workspace** tab of the user dashboard.



Python	Stack with Python 3.6	1.6 GB
My Stack	This is a custom stack for Che	1.6 GB
PHP Simple	PHP Stack with PHP 7.1 and simple web application	2.4 GB

1.1.3.2. Deploying registries in OpenShift

Procedure

An OpenShift template to deploy the plug-in registry is available in the **openshift/** directory of the GitHub repository.

1. To deploy the plug-in registry using the OpenShift template, run the following command:

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-plug-in-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-plugin-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="IfNotPresent"
```

(1) The default CodeReady Workspaces namespace is **workspaces**.

2. The devfile registry has an OpenShift template in the **deploy/openshift/** directory of the GitHub repository. To deploy it, run the command:

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-devfile-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-devfile-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="IfNotPresent"
```

1 The default CodeReady Workspaces namespace is **workspaces**.

3. Check if the registries are deployed successfully on OpenShift. The steps to check are similar to the OpenShift steps. For details, see the Section 1.1.3.1, "Deploying registries in OpenShift" section. The only difference is that, on OpenShift, Ingresses are replaced with routes.

1.2. INCLUDING THE PLUG-IN BINARIES IN THE REGISTRY IMAGE

The plug-in registry only hosts CodeReady Workspaces plug-in metadata. The binaries are usually referred through a link in the **meta.yaml** file. In some cases, such as offline environments, it may be necessary to make the binaries available inside the registry image.

This section describes how to modify a plug-in **meta.yaml** file to point to a local file inside the container and rebuild a new registry that contains the modified plug-in **meta.yaml** file and the binary. In the following example, the Java plug-in that refers to two remote VS Code extensions binaries is considered.

Procedure

1. Download the binaries locally.

```
$ ORG=redhat
$ NAME=java11
$ VERSION=latest
$ URL_VS_CODE_EXT1="https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-
0.19.0.vsix[_https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-0.19.0.vsix_]"
$ URL_VS_CODE_EXT2="https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-
0.46.0-1549.vsix[_https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-0.46.0-
1549.vsix_]"
$ VS_CODE_EXT1=https://github.com/microsoft/vscode-java-
debug/releases/download/0.19.0/vscode-java-debug-0.19.0.vsix[_vscode-java-debug-
0.19.0.vsix_]
$ VS_CODE_EXT2=https://download.jboss.org/jbosstools/static/jdt.ls/stable/java-0.46.0-
1549.vsix[_java-0.46.0-1549.vsix_]
$ curl -sSL -o ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/$\
{VS_CODE_EXT1} \
  $\{URL_VS_CODE_EXT1}
$ curl -sSL -o ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/$\
{VS_CODE_EXT2} \
  $\{URL_VS_CODE_EXT2}
```

1. Retrieve the plug-in-registry URL.

```
FIXME
```

2. Update the URLs in the **meta.yaml** file, so that they point to the VS Code extension binaries that are saved in the registry container:

```
$ NEW_URL_VS_CODE_EXT1=http://$\{PLUGIN_REG_URL}/v3/plugins/$\{ORG}/$\
{NAME}/$\{VERSION}/$\{VS_CODE_EXT1}
$ NEW_URL_VS_CODE_EXT2=http://$\{PLUGIN_REG_URL}/v3/plugins/$\{ORG}/$\
{NAME}/$\{VERSION}/$\{VS_CODE_EXT2}
```

```
$ sed -i -e 's/$\{URL_PLUGIN1}/$\{NEW_URL_VS_CODE_EXT1}/g' \
  ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/meta.yaml
$ sed -i -e 's/$\{URL_PLUGIN2}/$\{NEW_URL_VS_CODE_EXT2}/g' \
  ./che-plugin-registry/v3/plugins/$\{ORG}/$\{NAME}/$\{VERSION}/meta.yaml
```

3. Build and deploy the plug-in registry using the instructions in the Building and running a custom registry image section.

1.3. EDITING A DEVFILE AND PLUG-IN AT RUNTIME

An alternative to building a custom registry image is to:

1. Start a registry
2. Modify its content at runtime

This approach is simpler and faster. But the modifications are lost as soon as the container is deleted.

1.3.1. Adding a plug-in at runtime

Procedure

To add a plug-in:

1. Get the name of the OpenShift pod that hosts the plug-in registry container. To do this, filter the **app=che-plugin-registry** label:

```
$ PLUGIN_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
 --no-headers -l app=che-plugin-registry pod)
```

2. Create the new plug-in directory in the plug-in registry container, if it does not exist:

```
$ ORG="my-org"
$ PLUGIN="my-plug-in"
$ VERSION="1.0.1"
$ oc exec ${PLUGIN_REG_POD} -i -t -- \
  mkdir -p /var/www/html/v3/plugins/${ORG}/${PLUGIN}/${VERSION}
```

3. Copy the **meta.yaml** file in the container:

```
LOCAL_META="${PWD}/meta.yaml"
$ oc cp "${LOCAL_META}" \
  ${PLUGIN_REG_POD}:/var/www/html/v3/plugins/${ORG}/${PLUGIN}/${VERSION}
```

4. Update the **index.json** file of the registry:

```
$ oc exec ${PLUGIN_REG_POD} -i -t -- \
  /var/www/html/index_v2.sh v3 > /var/www/html/v3/plugins/index.json
```

5. The new plug-in can now be used from a CodeReady Workspaces instance.

1.3.2. Adding a devfile at runtime

Procedure

To add a devfile:

1. Get the name of the OpenShift pod that hosts the devfile registry container. To do this, filter the **app=che-devfile-registry** label:

```
$ DEVFILE_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
  --no-headers -l app=che-devfile-registry pod)
```

2. Create the new devfile directory in the devfile registry container, if it does not exist.

```
$ STACK="my-stack"
$ oc exec $\{DEVFILE_REG_POD} -i -t -- \
  mkdir -p /var/www/html/devfiles/$\{STACK}
```

3. Copy the **devfile.yaml** file and the **meta.yaml** file to the container.

```
$ LOCAL_META="$(PWD)/meta.yaml"
$ LOCAL_DEVFILE="$(PWD)/devfile.yaml"
$ oc cp "$\{LOCAL_META}" \
  $\{DEVFILE_REG_POD}:/var/www/html/devfiles/$\{STACK}
$ oc cp "$\{LOCAL_DEVFILE}" \
  $\{DEVFILE_REG_POD}:/var/www/html/devfiles/$\{STACK}
```

4. Update the **index.json** file of the registry.

```
$ oc exec $\{DEVFILE_REG_POD} -i -t -- \
  /var/www/html/index.sh > /var/www/html/devfiles/index.json
```

CHAPTER 2. RETRIEVING CODEREADY WORKSPACES LOGS

This is a catalog of the location and instructions to retrieve application logs (for administrators and for users).

2.1. VIEWING OPENSHIFT EVENTS

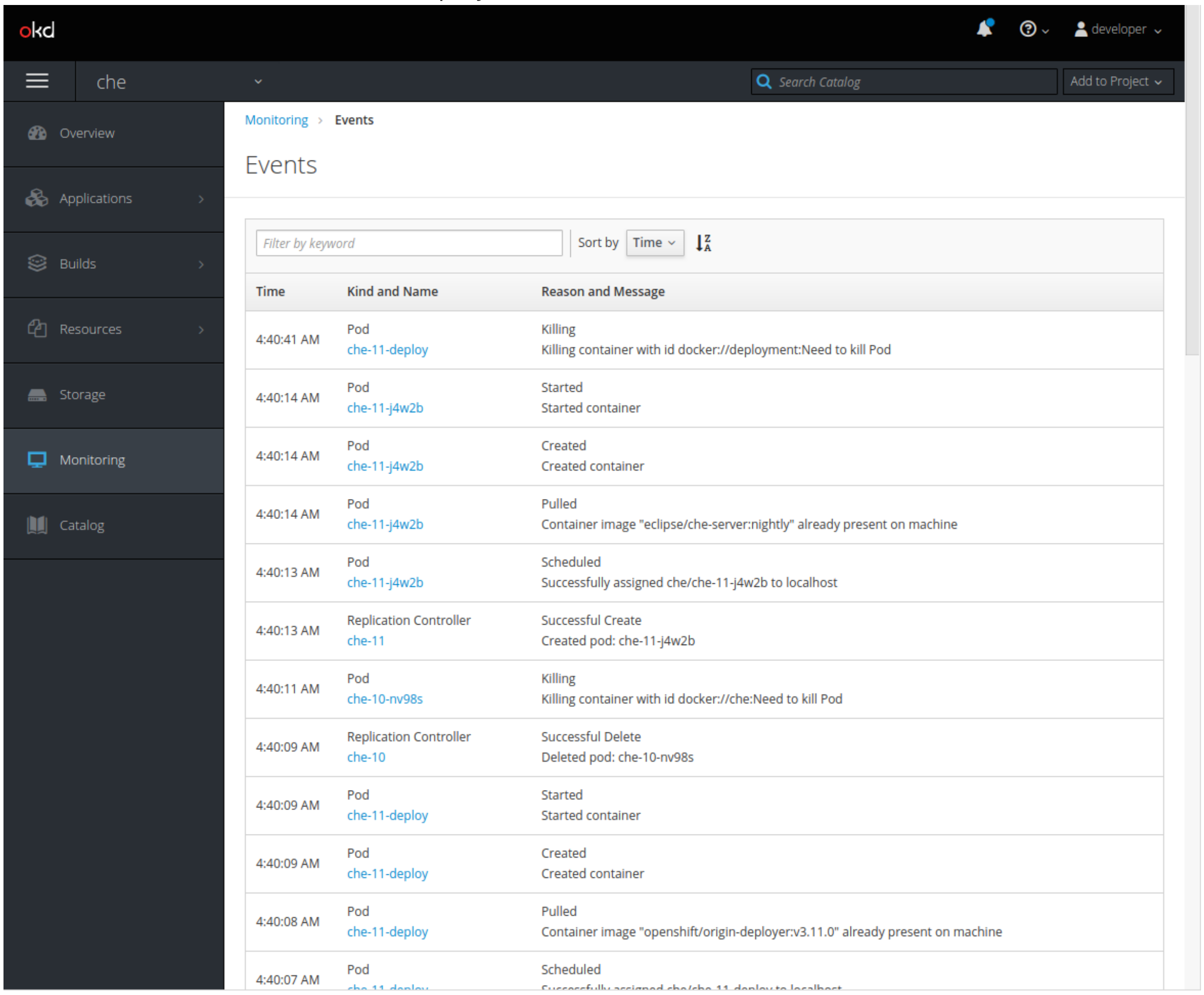
This section describes how to view the Kubenertes events.

Prerequisites

- A running OpenShift Web Console.

Procedure

1. In the OpenShift Web Console, click the **Monitoring** tab in the left panel.
2. To view the list of all events, click the **View Details** button in the top-right corner.
3. The details of the events will be displayed.



Time	Kind and Name	Reason and Message
4:40:41 AM	Pod che-11-deploy	Killing Killing container with id docker://deployment:Need to kill Pod
4:40:14 AM	Pod che-11-j4w2b	Started Started container
4:40:14 AM	Pod che-11-j4w2b	Created Created container
4:40:14 AM	Pod che-11-j4w2b	Pulled Container image "eclipse/che-server:nightly" already present on machine
4:40:13 AM	Pod che-11-j4w2b	Scheduled Successfully assigned che/che-11-j4w2b to localhost
4:40:13 AM	Replication Controller che-11	Successful Create Created pod: che-11-j4w2b
4:40:11 AM	Pod che-10-nv98s	Killing Killing container with id docker://che:Need to kill Pod
4:40:09 AM	Replication Controller che-10	Successful Delete Deleted pod: che-10-nv98s
4:40:09 AM	Pod che-11-deploy	Started Started container
4:40:09 AM	Pod che-11-deploy	Created Created container
4:40:08 AM	Pod che-11-deploy	Pulled Container image "openshift/origin-deployer:v3.11.0" already present on machine
4:40:07 AM	Pod	Scheduled

Additional resources

- For a list of OpenShift events, see Comprehensive List of Events in OpenShift documentation .

2.2. VIEWING CODEREADY WORKSPACES SERVER LOGS

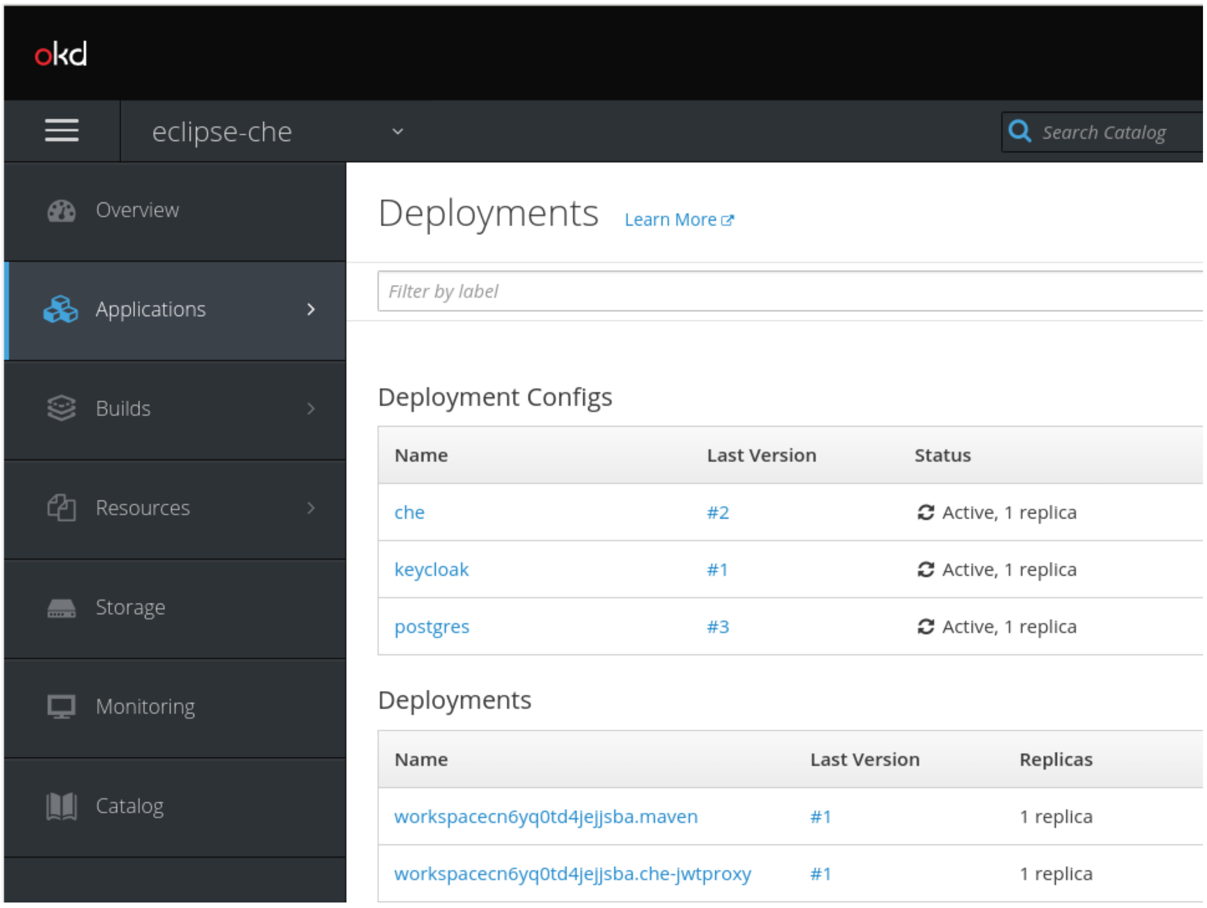
This section describes how to view the CodeReady Workspaces server logs on the console and on the command line.

2.2.1. Viewing the CodeReady Workspaces server logs in the web console

This section describes how to view the CodeReady Workspaces server logs in the OpenShift web console.

Procedure

1. In the OpenShift Web Console, click **Applications > Deployments**.
2. In the **Filter by label** search field, type **workspaces** to see the CodeReady Workspaces server log.



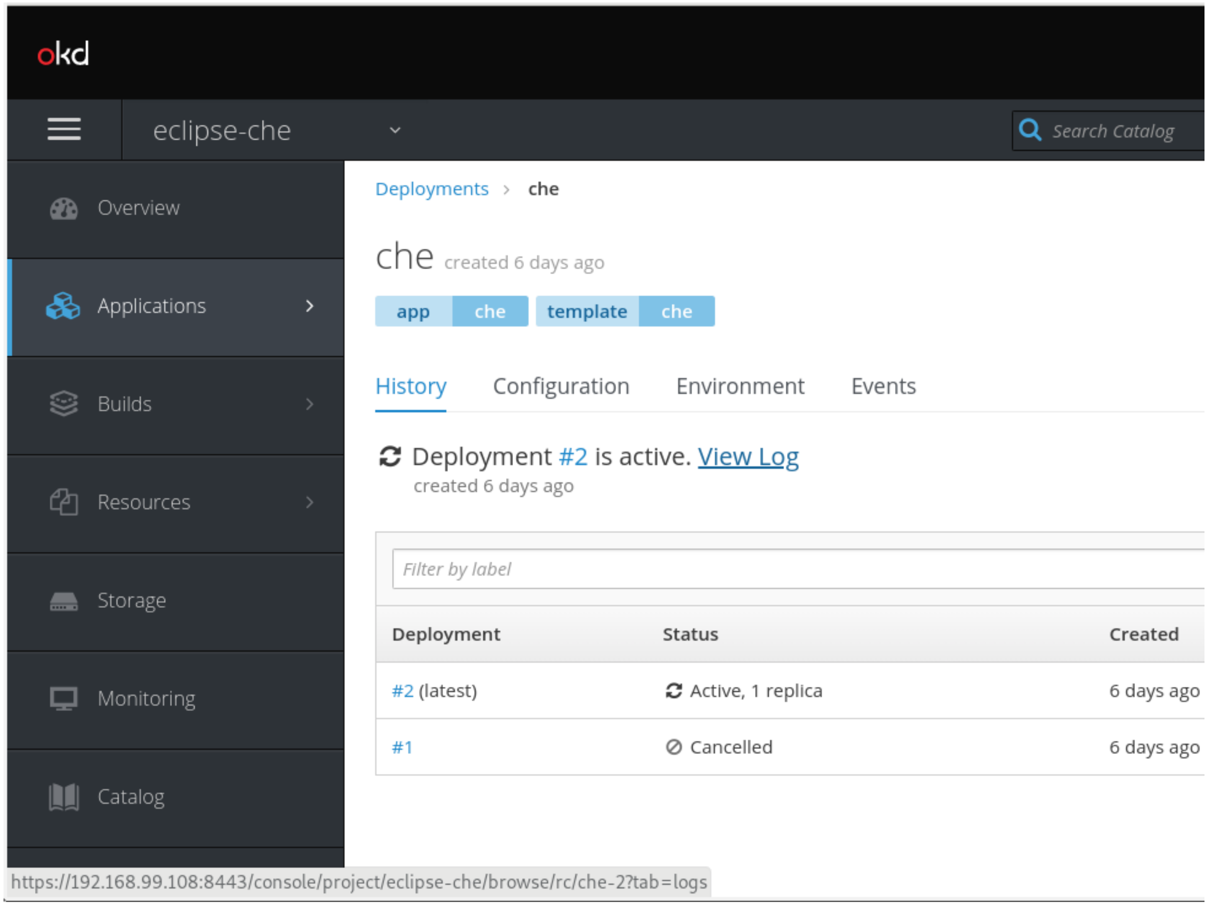
Deployment Configs

Name	Last Version	Status
che	#2	Active, 1 replica
keycloak	#1	Active, 1 replica
postgres	#3	Active, 1 replica

Deployments

Name	Last Version	Replicas
workspacecn6yq0td4jejjsba.maven	#1	1 replica
workspacecn6yq0td4jejjsba.che-jwtproxy	#1	1 replica

3. Click the **View log** link for the active CodeReady Workspaces deployment.

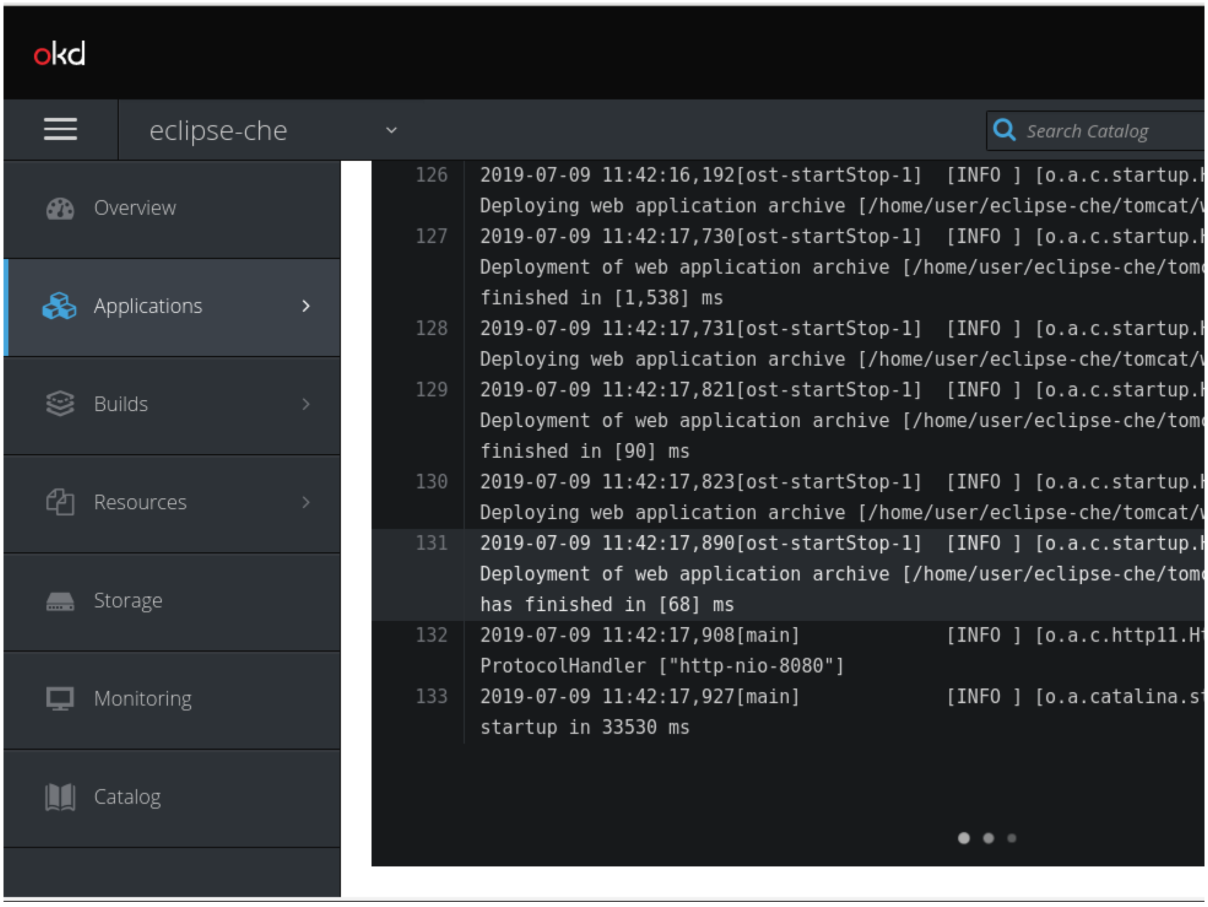


Deployment #2 is active. View Log
created 6 days ago

Deployment	Status	Created
#2 (latest)	Active, 1 replica	6 days ago
#1	Cancelled	6 days ago

https://192.168.99.108:8443/console/project/eclipse-che/browse/rc/che-2?tab=logs

4. The following log is displayed.



```
2019-07-09 11:42:16,192[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deploying web application archive [/home/user/eclipse-che/tomcat/w
2019-07-09 11:42:17,730[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deployment of web application archive [/home/user/eclipse-che/tomc
finished in [1,538] ms
2019-07-09 11:42:17,731[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deploying web application archive [/home/user/eclipse-che/tomcat/w
2019-07-09 11:42:17,821[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deployment of web application archive [/home/user/eclipse-che/tomc
finished in [90] ms
2019-07-09 11:42:17,823[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deploying web application archive [/home/user/eclipse-che/tomcat/w
2019-07-09 11:42:17,890[ost-startStop-1]  [INFO ] [o.a.c.startup.H
Deployment of web application archive [/home/user/eclipse-che/tomc
has finished in [68] ms
2019-07-09 11:42:17,908[main]             [INFO ] [o.a.c.http11.Ht
ProtocolHandler ["http-nio-8080"]
2019-07-09 11:42:17,927[main]             [INFO ] [o.a.catalina.st
startup in 33530 ms
```

5. Search the log for CodeReady Workspaces-server related diagnostics, error information, and information reported by other server components.

2.2.2. Viewing the CodeReady Workspaces server logs on the command line

This section describes how to view the CodeReady Workspaces server logs on the command line.

Procedure

1. In the terminal, run the following command to get the pods:

   ```
   $ oc get pods
   ```

   EXAMPLE

   ```
   $ oc get pods
   NAME           READY  STATUS   RESTARTS  AGE
   che-11-j4w2b    1/1    Running  0         3m
   ```

2. To get the logs for a deployment, run the following command:

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

   EXAMPLE

   ```
   $ oc logs che-11-j4w2b
   ```

2.3. VIEWING EXTERNAL SERVICE LOGS

This section describes how the view the logs from external services related to CodeReady Workspaces server.

2.3.1. Viewing Keycloak logs

The Keycloak OpenID provider consists of two parts: Server and IDE. It writes its diagnostics or error information to several logs.

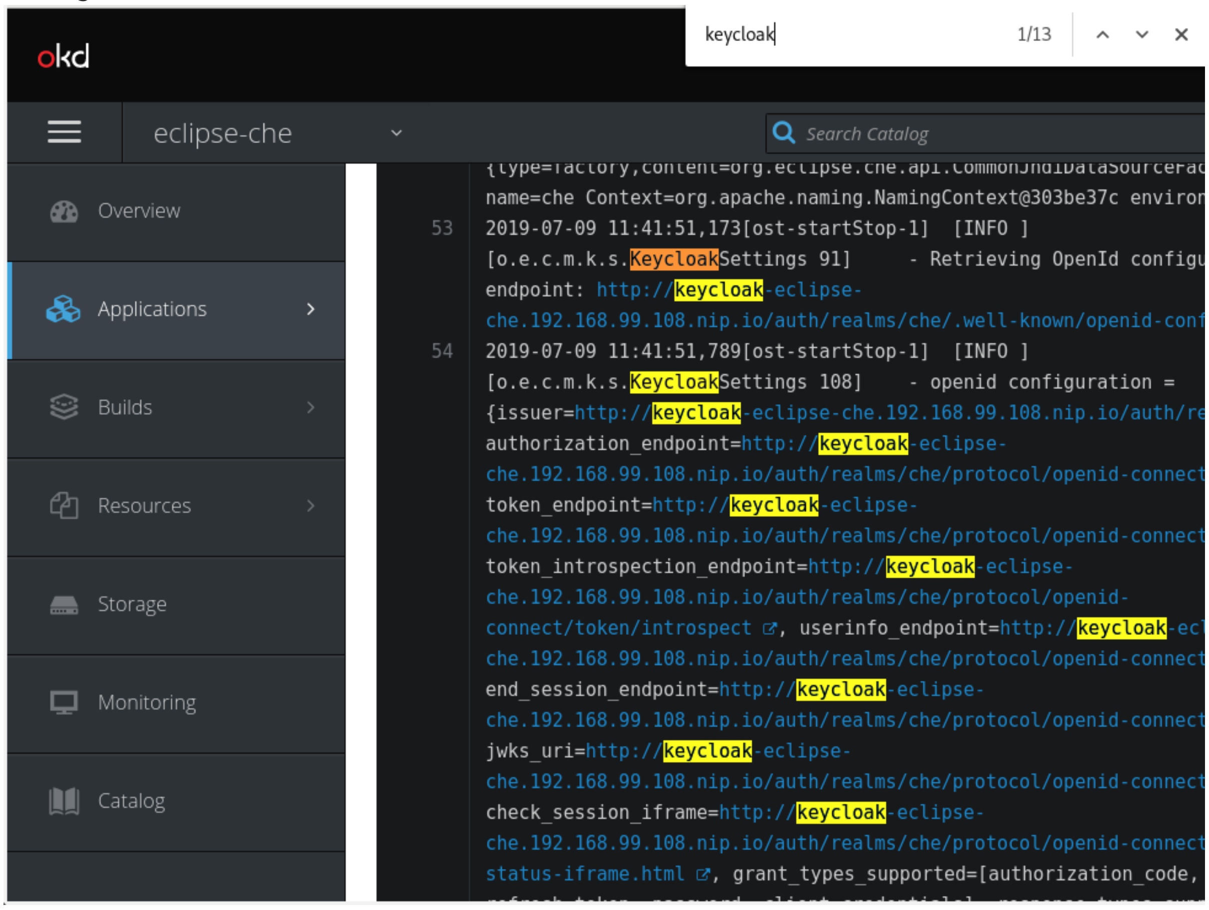
2.3.1.1. Viewing the Keycloak server logs

This section describes how to view the Keycloak OpenID provider server logs.

Procedure

1. In the OpenShift Web Console, click **Deployments**.
2. In the **Filter by label** search field, type **keycloak** to see the Keycloak logs.
3. In the **Deployment Configs** section, click on the **keycloak** link to open it.
4. In the **History** tab, click the **View log** link for the active Keycloak deployment.
5. The Keycloak logs are displayed.

6. To view the Keycloak IDE server diagnostics or error related messages, search for **keycloak** in the log.



```
{type=factory,content=org.eclipse.che.api.CommonJndiDataSourceFac
name=che Context=org.apache.naming.NamingContext@303be37c environ
2019-07-09 11:41:51,173[ost-startStop-1]  [INFO ]
[o.e.c.m.k.s.KeycloakSettings 91]     - Retrieving OpenId configu
endpoint: http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/.well-known/openid-conf
2019-07-09 11:41:51,789[ost-startStop-1]  [INFO ]
[o.e.c.m.k.s.KeycloakSettings 108]    - openid configuration =
{issuer=http://keycloak-eclipse-che.192.168.99.108.nip.io/auth/re
authorization_endpoint=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
token_endpoint=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
token_introspection_endpoint=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-
connect/token/introspect , userinfo_endpoint=http://keycloak-ecl
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
end_session_endpoint=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
jwks_uri=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
check_session_iframe=http://keycloak-eclipse-
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-connect
status-iframe.html , grant_types_supported=[authorization_code,
```

Additional resources

- See the Section 2.2, "Viewing CodeReady Workspaces server logs" for diagnostics and error messages related to the Keycloak IDE Server.

2.3.1.2. Viewing the Keycloak client logs on Firefox

This section describes how to view the Keycloak IDE client diagnostics or error information in the Firefox **WebConsole**.

Procedure

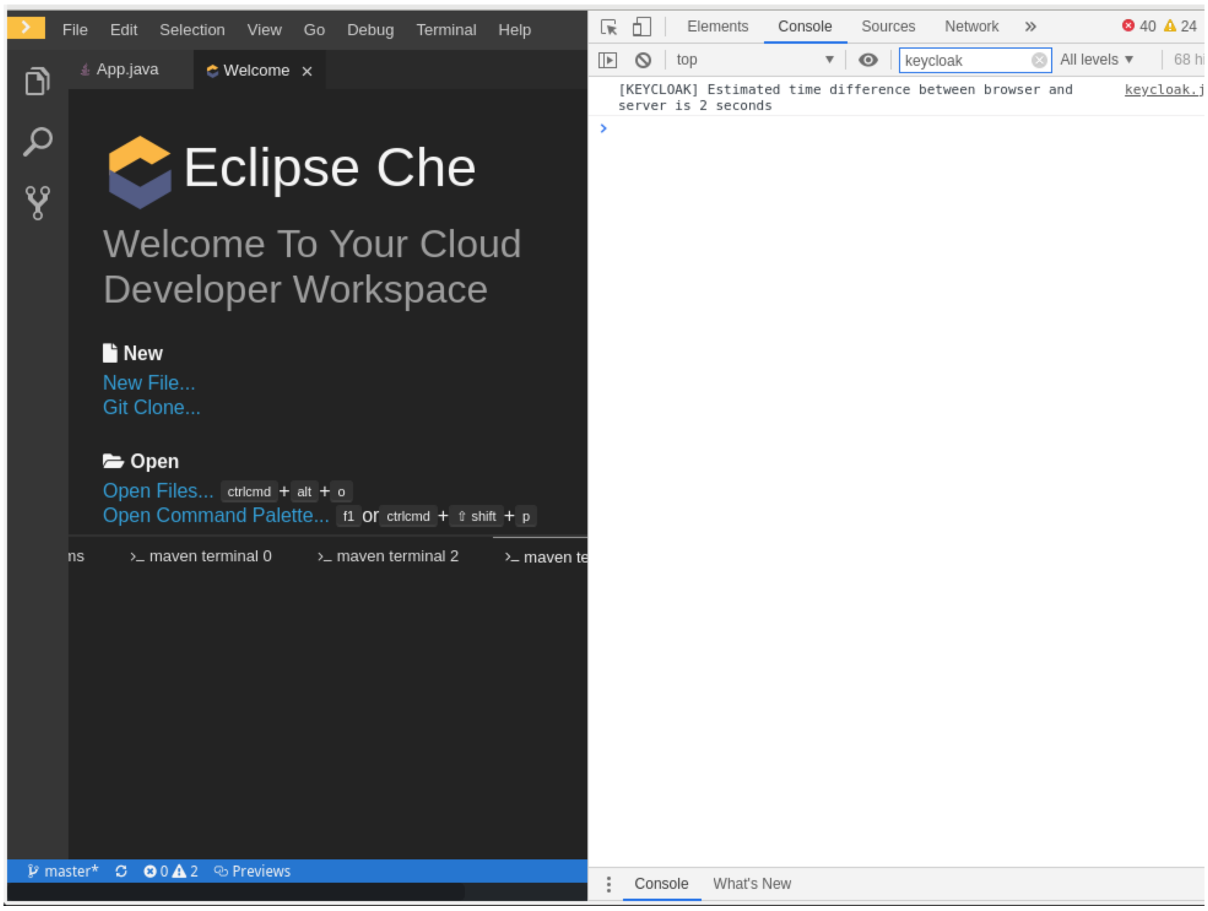
- Click **Menu** > **WebDeveloper** > **WebConsole**.

2.3.1.3. Viewing the Keycloak client logs on Google Chrome

This section describes how to view the Keycloak IDE client diagnostics or error information in the Google Chrome **Console** tab.

Procedure

1. Click on **Menu** > **More Tools** > **Developer Tools**.
2. Click on the **Console** tab.


File Edit Selection View Go Debug Terminal Help
App.java
Welcome
Eclipse Che
Welcome To Your Cloud Developer Workspace
New
New File...
Git Clone...
Open
Open Files... ctrlcmd + alt + o
Open Command Palette... f1 or ctrlcmd + ⇧ shift + p
maven terminal 0
maven terminal 2
master*
Previews
Elements Console Sources Network
top
keycloak
All levels
Console What's New

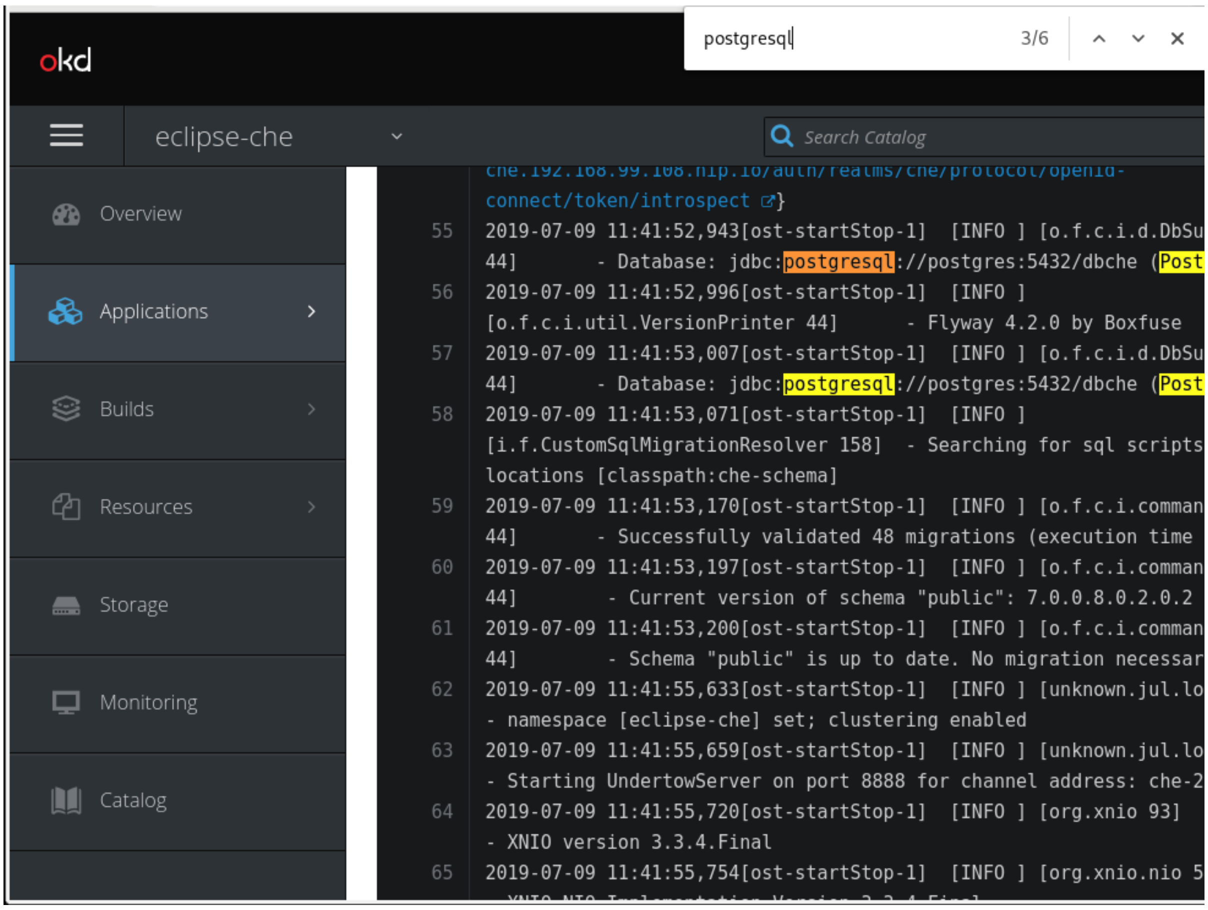
```
[KEYCLOAK] Estimated time difference between browser and
server is 2 seconds
```

2.3.2. Viewing the PostgreSQL server logs

This section describes how to view the PostgreSQL server logs.

Procedure

1. In the OpenShift Web Console, click **Deployments**.
2. In the **Find by label** search field, type **postgres** to see the PostgreSQL logs. Click **postgres** deployment to open it.
3. Click the **View log** link for the active PostgreSQL deployment.
4. The logs are displayed.
5. To see PostgreSQL server diagnostics or error related messages, search **postgresql** through the log.



```
che.192.168.99.108.nip.io/auth/realms/che/protocol/openid-
connect/token/introspect }
2019-07-09 11:41:52,943[ost-startStop-1]  [INFO ] [o.f.c.i.d.DbSu
44]       - Database: jdbc:postgresql://postgres:5432/dbche (Post
2019-07-09 11:41:52,996[ost-startStop-1]  [INFO ]
[o.f.c.i.util.VersionPrinter 44]      - Flyway 4.2.0 by Boxfuse
2019-07-09 11:41:53,007[ost-startStop-1]  [INFO ] [o.f.c.i.d.DbSu
44]       - Database: jdbc:postgresql://postgres:5432/dbche (Post
2019-07-09 11:41:53,071[ost-startStop-1]  [INFO ]
[i.f.CustomSqlMigrationResolver 158]  - Searching for sql scripts
locations [classpath:che-schema]
2019-07-09 11:41:53,170[ost-startStop-1]  [INFO ] [o.f.c.i.comman
44]       - Successfully validated 48 migrations (execution time
2019-07-09 11:41:53,197[ost-startStop-1]  [INFO ] [o.f.c.i.comman
44]        - Current version of schema "public": 7.0.0.8.0.2.0.2
2019-07-09 11:41:53,200[ost-startStop-1]  [INFO ] [o.f.c.i.comman
44]        - Schema "public" is up to date. No migration necessar
2019-07-09 11:41:55,633[ost-startStop-1]  [INFO ] [unknown.jul.lo
- namespace [eclipse-che] set; clustering enabled
2019-07-09 11:41:55,659[ost-startStop-1]  [INFO ] [unknown.jul.lo
- Starting UndertowServer on port 8888 for channel address: che-2
2019-07-09 11:41:55,720[ost-startStop-1]  [INFO ] [org.xnio 93]
- XNIO version 3.3.4.Final
2019-07-09 11:41:55,754[ost-startStop-1]  [INFO ] [org.xnio.nio 5
```

Additional resources

- Some diagnostics or error messages related to the PostgreSQL server can be found in the active CodeReady Workspaces deployment log. For details to access the active CodeReady Workspaces deployments logs, see the Section 2.2, “Viewing CodeReady Workspaces server logs” section.

2.4. VIEWING WORKSPACES LOGS

This section describes how to view workspaces logs.

2.4.1. Viewing Che-Theia IDE logs

This section describes how to view Che-Theia IDE logs, on the command line and in the OpenShift web console.

2.4.1.1. Viewing Che-Theia IDE logs on the command line

This section describes how to view Che-Theia IDE logs on the command line.

Procedure

1. Run the following command to get the list of all the pods:

```
$ oc get pods
```

EXAMPLE

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

2. Run the following command to get the list of all the containers in the particular pod:

```
$ oc get pods <name-of-pod> -o
```

EXAMPLE

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

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

```
$ oc logs -f <name-of-container> -c
```

EXAMPLE

```
$ oc logs -f workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -c
theia-idexzb
```

2.4.1.2. Viewing Che-Theia IDE logs in the web console

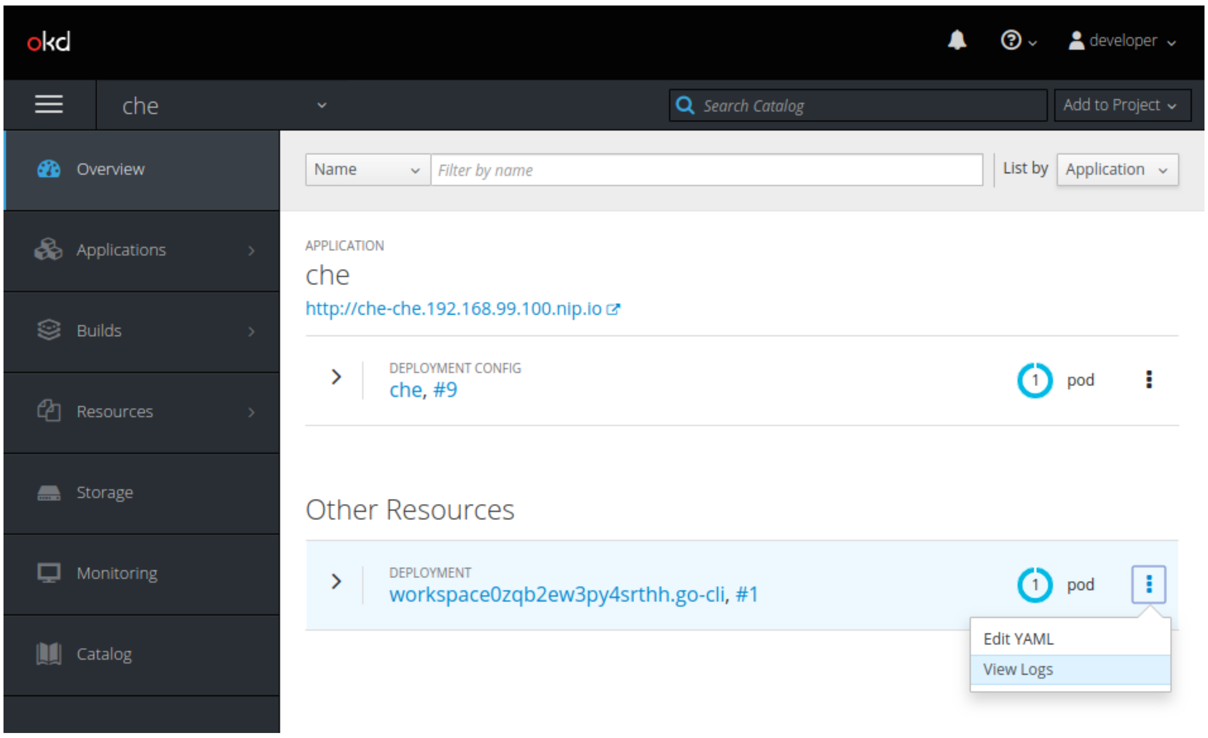
This section describes how to view Che-Theia IDE logs in the OpenShift web console.

Procedure

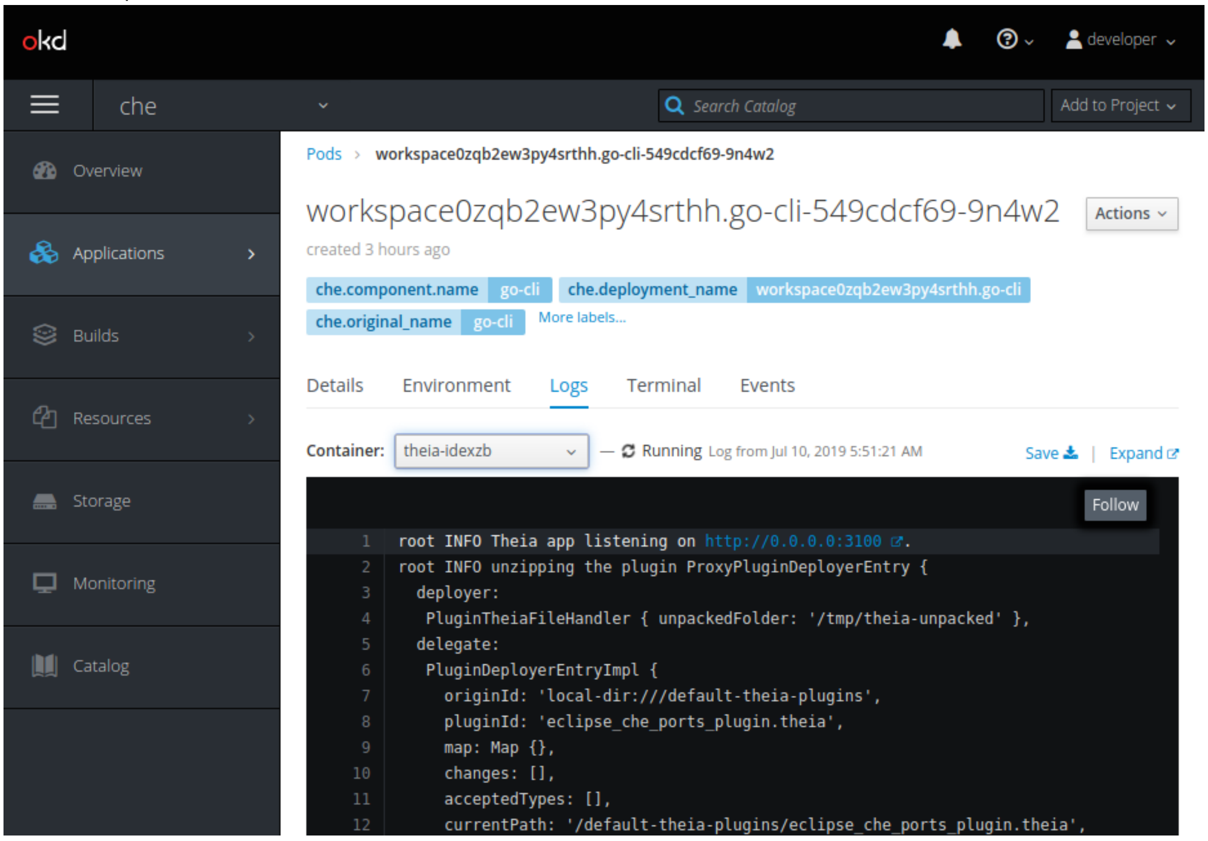
To view Che-Theia IDE logs on the console:

1. In the OpenShift Web Console, click **Overview**.

2. Search for the particular deployment, click on the context menu on the right side of the deployment, and click **View Logs.**



3. In the drop-down list, click **theia-ide**.



```
root INFO Theia app listening on http://0.0.0.0:3100 .
root INFO unzipping the plugin ProxyPluginDeployerEntry {
  deployer:
   PluginTheiaFileHandler { unpackedFolder: '/tmp/theia-unpacked' },
  delegate:
   PluginDeployerEntryImpl {
     originId: 'local-dir:///default-theia-plugins',
     pluginId: 'eclipse_che_ports_plugin.theia',
     map: Map {},
     changes: [],
     acceptedTypes: [],
     currentPath: '/default-theia-plugins/eclipse_che_ports_plugin.theia',
```

2.4.2. Viewing logs from language servers and debug adapters

2.4.2.1. Checking important logs

This section describes how to check important logs.

Procedure

1. In the OpenShift web console, click **Applications → Pods** to see a list of all the active workspaces.

2. Click on the name of the running pod where the workspace is running. The pod screen contains the list of all containers with additional information.

3. Choose a container and click on the container name.

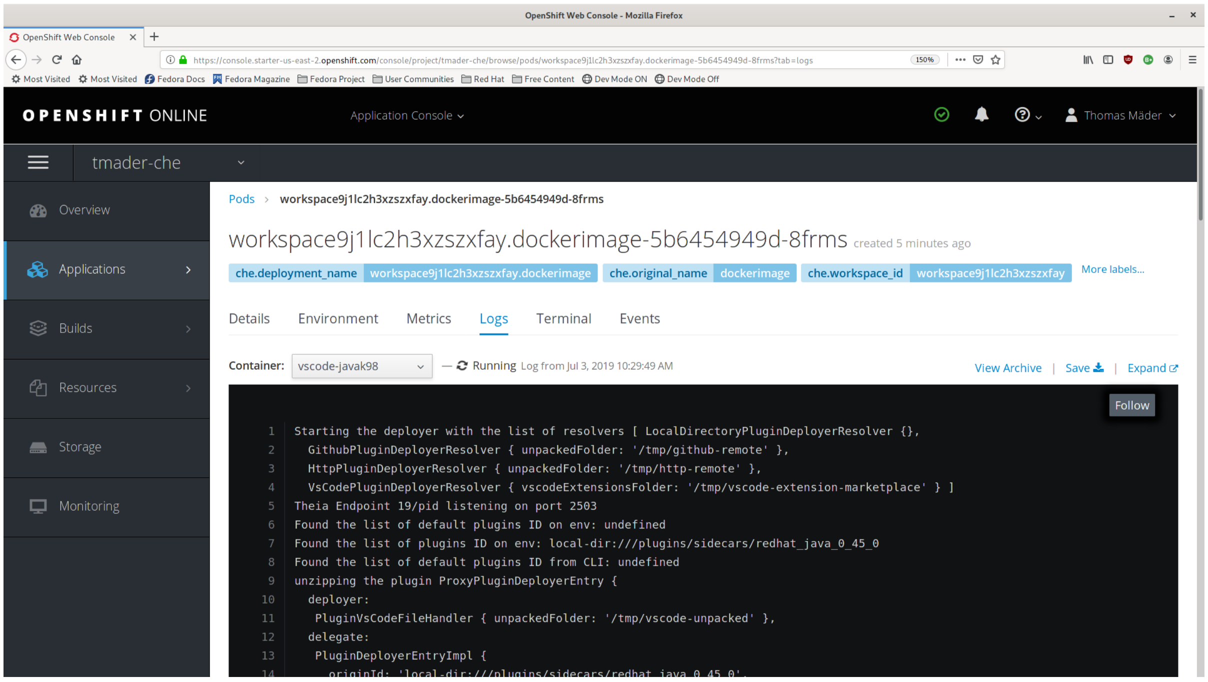
   ### TIP

   The most important logs are the **theia-ide** container and the plug-ins container logs.

4. On the container screen, navigate to the **Logs** section.

   ### EXAMPLE

   The following is an output log of the sidecar container running the Java plug-in.



workspace9j1lc2h3xzszxfay.dockerimage-5b6454949d-8frms created 5 minutes ago

che.deployment_name workspace9j1lc2h3xzszxfay.dockerimage | che.original_name dockerimage | che.workspace_id workspace9j1lc2h3xzszxfay | More labels...

Container: vscode-javak98 — Running Log from Jul 3, 2019 10:29:49 AM

```
Starting the deployer with the list of resolvers [ LocalDirectoryPluginDeployerResolver {},
  GithubPluginDeployerResolver { unpackedFolder: '/tmp/github-remote' },
  HttpPluginDeployerResolver { unpackedFolder: '/tmp/http-remote' },
  VsCodePluginDeployerResolver { vscodeExtensionsFolder: '/tmp/vscode-extension-marketplace' } ]
Theia Endpoint 19/pid listening on port 2503
Found the list of default plugins ID on env: undefined
Found the list of plugins ID on env: local-dir:///plugins/sidecars/redhat_java_0_45_0
Found the list of default plugins ID from CLI: undefined
unzipping the plugin ProxyPluginDeployerEntry {
  deployer:
   PluginVsCodeFileHandler { unpackedFolder: '/tmp/vscode-unpacked' },
  delegate:
   PluginDeployerEntryImpl {
     originId: 'local-dir:///plugins/sidecars/redhat_java_0_45_0',
```

2.4.2.2. Detecting memory problems

This section describes how to detect memory problems related to a plug-in running out of memory. The following are the two most common problems related to a plug-in running out of memory:

The plug-in container runs out of memory

This can happen during plug-in initialization when the container does not have enough RAM to execute the entrypoint of the image. The user can detect this in the logs of the plug-in container. In this case, the logs should contain **OOMKilled**, which implies that the processes in the container requested more memory than is available in the container.

A process inside the container runs out of memory without the container noticing this

For example, the Java language server (Eclipse JDT Language Server, started by the vscode-java extension) throws an **OutOfMemoryException**. This can happen any time after the container is initialized, for example, when a plug-in starts a language server or when a process runs out of memory because of the size of the project it has to handle.

To detect this problem, check the logs of the main process, which should run in the container. For example, to check the log file of Eclipse JDT Language Server for details, see the relevant plug-in-specific sections.

2.4.2.3. Logging the client-server traffic for debug adapters

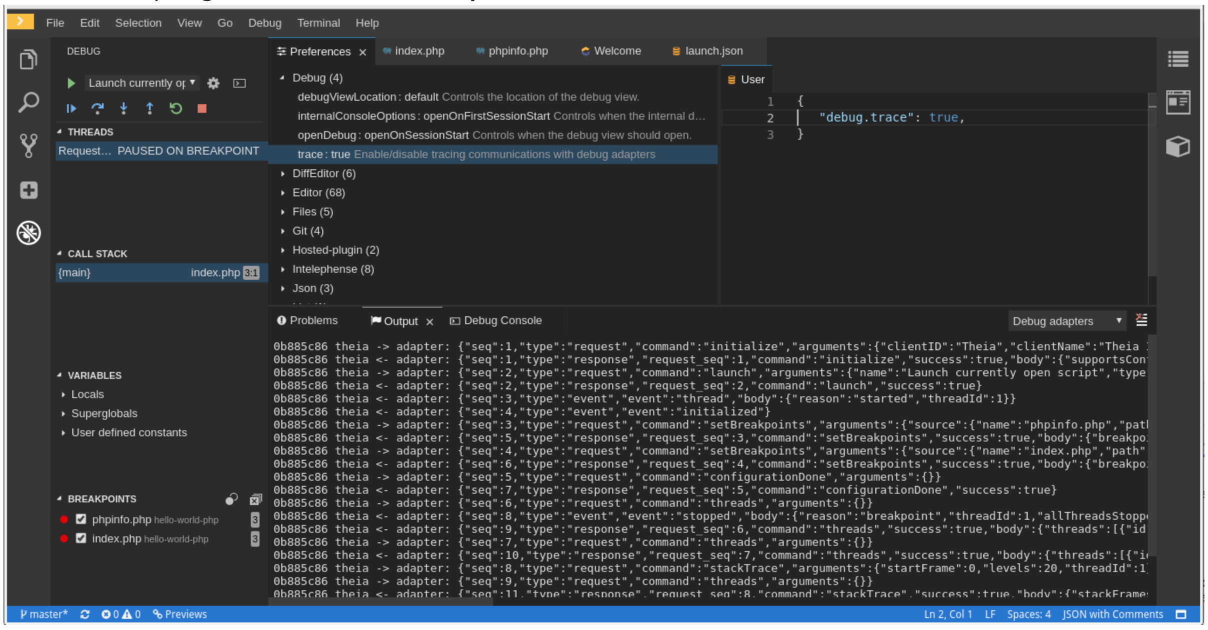
This section describes how to log the exchange between Che-Theia and a debug adapter into the **Output** view.

Prerequisites

- A debug session must be started for the **Debug adapters** option to appear in the list.

Procedure

1. Click **File → Settings** and then **open Preferences**.
2. Expand the **Debug** section in the **Preferences** view.
3. Set the **trace** preference value to **true** (default is **false**).
4. All the communication events are now logged.
5. To watch these events, click **View → Output** and select **Debug adapters** from the drop-down list at the top-right corner of the **Output** view.



Debug (4)
debugViewLocation : default Controls the location of the debug view.
internalConsoleOptions : openOnFirstSessionStart Controls when the internal d...
openDebug : openOnSessionStart Controls when the debug view should open.
trace : true Enable/disable tracing communications with debug adapters
DiffEditor (6)
Editor (68)
Files (5)
Git (4)
Hosted-plugin (2)
Intelephense (8)
Json (3)

```
{
    "debug.trace": true,
}
```

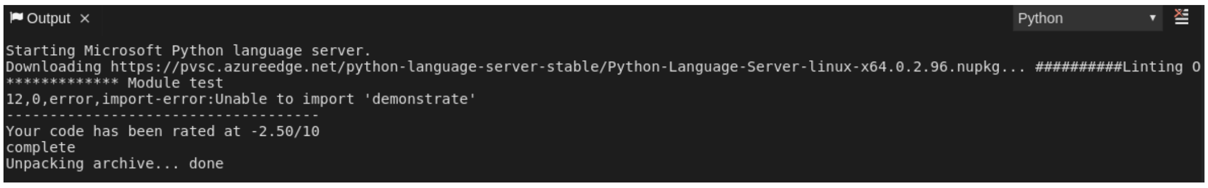
```
0b885c86 theia -> adapter: {"seq":1,"type":"request","command":"initialize","arguments":{"clientID":"Theia","clientName":"Theia 
0b885c86 theia <- adapter: {"seq":1,"type":"response","request_seq":1,"command":"initialize","success":true,"body":{"supportsCon
0b885c86 theia -> adapter: {"seq":2,"type":"request","command":"launch","arguments":{"name":"Launch currently open script","type
0b885c86 theia <- adapter: {"seq":2,"type":"response","request_seq":2,"command":"launch","success":true}
0b885c86 theia <- adapter: {"seq":3,"type":"event","event":"thread","body":{"reason":"started","threadId":1}}
0b885c86 theia <- adapter: {"seq":4,"type":"event","event":"initialized"}
0b885c86 theia -> adapter: {"seq":3,"type":"request","command":"setBreakpoints","arguments":{"source":{"name":"phpinfo.php","pat
0b885c86 theia <- adapter: {"seq":5,"type":"response","request_seq":3,"command":"setBreakpoints","success":true,"body":{"breakpo
0b885c86 theia -> adapter: {"seq":4,"type":"request","command":"setBreakpoints","arguments":{"source":{"name":"index.php","path"
0b885c86 theia <- adapter: {"seq":6,"type":"response","request_seq":4,"command":"setBreakpoints","success":true,"body":{"breakpo
0b885c86 theia -> adapter: {"seq":5,"type":"request","command":"configurationDone","arguments":{}}
0b885c86 theia <- adapter: {"seq":7,"type":"response","request_seq":5,"command":"configurationDone","success":true}
0b885c86 theia -> adapter: {"seq":6,"type":"request","command":"threads","arguments":{}}
0b885c86 theia <- adapter: {"seq":8,"type":"event","event":"stopped","body":{"reason":"breakpoint","threadId":1,"allThreadsStopp
0b885c86 theia <- adapter: {"seq":9,"type":"response","request_seq":6,"command":"threads","success":true,"body":{"threads":[{"id
0b885c86 theia -> adapter: {"seq":7,"type":"request","command":"threads","arguments":{}}
0b885c86 theia <- adapter: {"seq":10,"type":"response","request_seq":7,"command":"threads","success":true,"body":{"threads":[{"i
0b885c86 theia -> adapter: {"seq":8,"type":"request","command":"stackTrace","arguments":{"startFrame":0,"levels":20,"threadId":1
0b885c86 theia -> adapter: {"seq":9,"type":"request","command":"threads","arguments":{}}
0b885c86 theia <- adapter: {"seq":11,"type":"response","request_seq":8,"command":"stackTrace","success":true,"body":{"stackFrame
```

2.4.2.4. Viewing logs for Python

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

Procedure

- Navigate to the **Output** view and select **Python** in the drop-down list.



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

2.4.2.5. Viewing logs for Go

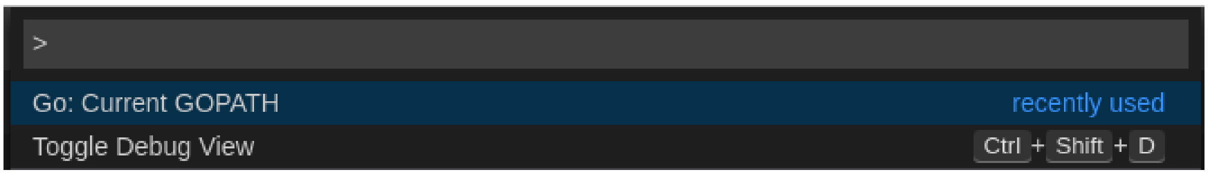
This section describes how to view logs for the Go language server.

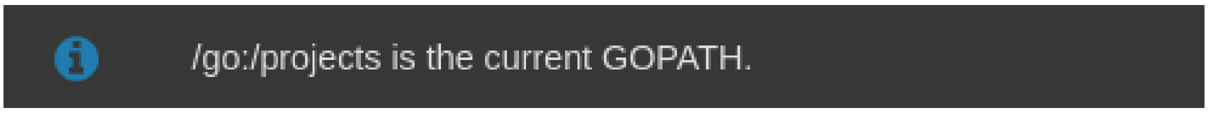
2.4.2.5.1. Finding the gopath

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

Procedure

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





/go:/projects is the current GOPATH.

2.4.2.5.2. Viewing the Debug Console log for Go

This section describes how to view the log output from the Go debugger.

Procedure

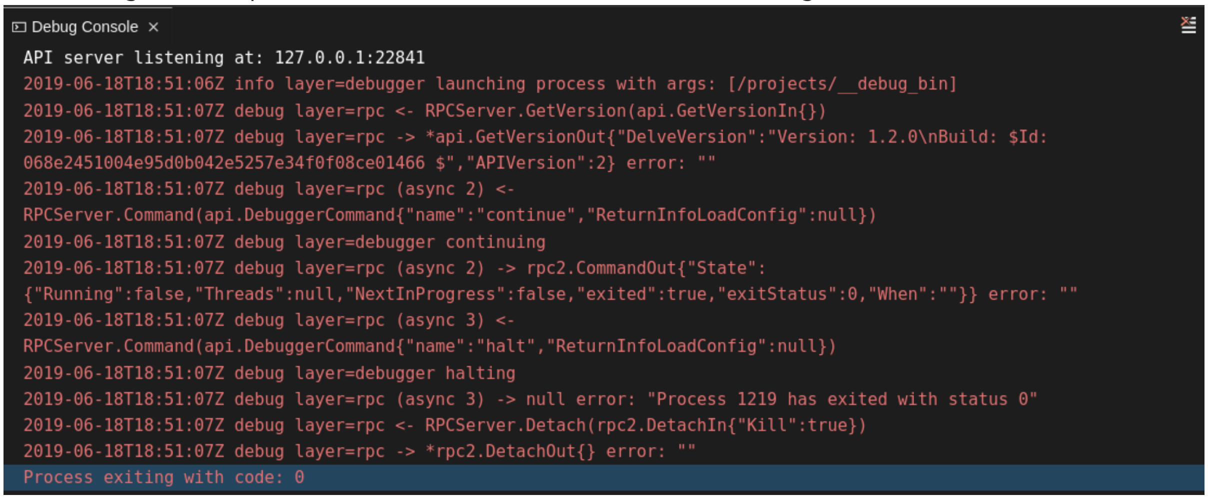
1. Set the **showLog** attribute to **true** in the debug configuration.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true
    ....
    }
  ]
}
```

2. To enable debugging output for a component, add the package to the comma-separated list value of the **logOutput** attribute:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
            "showLog": true,
            "logOutput": "debugger,rpc,gdbwire,lldbout,debuglineerr"
           ....
         }
      ]
   }
```

3. The debug console prints the additional information in the debug console.



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

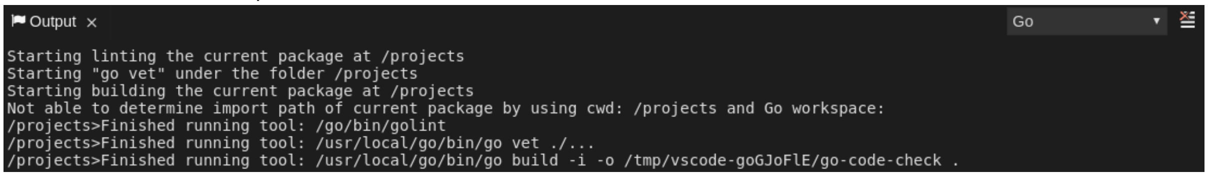
2.4.2.5.3. Viewing the Go logs output in the Output panel

This section describes how to view the Go logs output in the **Output** panel.

Procedure

1. Navigate to the **Output** view.

2. Select **Go** in the drop-down list.



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

2.4.2.6. Viewing logs for the NodeDebug NodeDebug2 adapter

NOTE

There are no specific diagnostics other than the general ones.

2.4.2.7. Viewing logs for Typescript

2.4.2.7.1. Enabling the label switched protocol (LSP) tracing

Procedure

1. To enable the tracing of messages sent to the Typescript (TS) server, in the **Preferences** view, set the **typescript.tsserver.trace** attribute to **verbose**. Use this to diagnose the TS server issues.

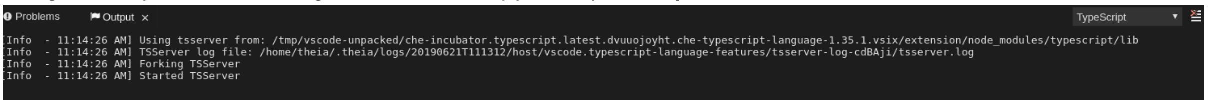
2. To enable logging of the TS server to a file, set the **typescript.tsserver.log** attribute to **verbose**. Use this log to diagnose the TS server issues. The log contains the file paths.

2.4.2.7.2. Viewing the Typescript language server log

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

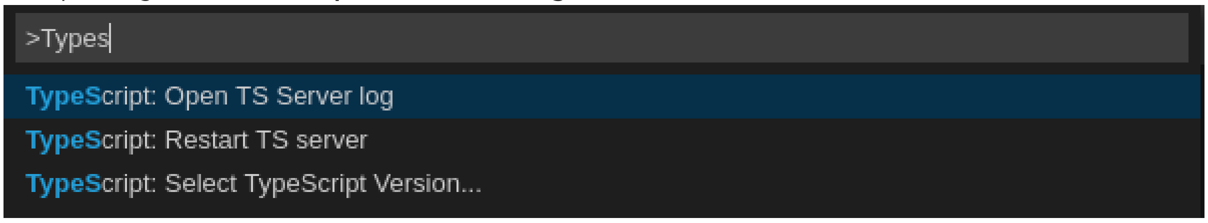
Procedure

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



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

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



```
>Types
```

TypeScript: Open TS Server log
TypeScript: Restart TS server
TypeScript: Select TypeScript Version...

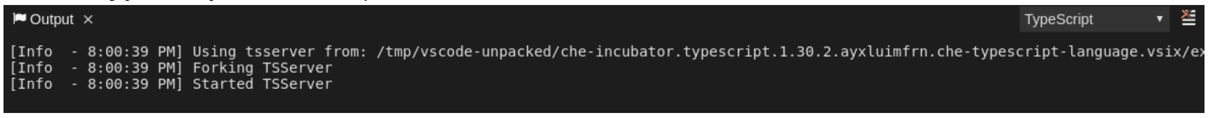
2.4.2.7.3. Viewing the Typescript logs output in the Output panel

This section describes how to view the Typescript logs output in the **Output** panel.

Procedure

1. Navigate to the **Output** view

2. Select **TypeScript** in the drop-down list.



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

2.4.2.8. Viewing logs for Java

Other than the general diagnostics, there are Language Support for Java (Eclipse JDT Language Server) plug-in actions that the user can perform.

2.4.2.8.1. Verifying the state of the Eclipse JDT Language Server

Procedure

Check if the container running the Eclipse JDT Language Server plug-in is running the Eclipse JDT Language Server main process.

1. Open a terminal in the container that is running the Eclipse JDT Language Server plug-in (an example name for the container: **vscode-javaxxx**).

2. Inside the terminal, run the **ps aux | grep jdt** command to check if the Eclipse JDT Language Server process is running in the container. If the process is running, the output should be:

```
usr/lib/jvm/default-jvm/bin/java --add-modules=ALL-SYSTEM --add-opens java.base/java.util
```

This message also shows the VSCode java extension used. If it is not running, the language server has not been started inside the container.

3. Check all logs described in Section 2.4.2.1, “Checking important logs”

2.4.2.8.2. Verifying the Eclipse JDT Language Server features

Procedure

If the Eclipse JDT Language Server process is running, check if the language server features are working:

1. Open a Java file and use the hover or autocomplete functionality. In case of an erroneous file, the user sees Java in the **Outline** view or in the **Problems** view.

2.4.2.8.3. Viewing the Java language server log

Procedure

The Eclipse JDT Language Server has its own workspace where it logs errors, information about executed commands, and events.

1. To open this log file, open a terminal in the container that is running the Eclipse JDT Language Server plug-in. You can also view the log file by running the **Java: Open Java Language Server log file** command.

2. Run **cat *<PATH_TO_LOG_FILE>*** where **PATH_TO_LOG_FILE** is **/home/theia/.theia/workspace-storage/*<workspace_name>*/redhat.java/jdt_ws/.metadata/.log**.

2.4.2.8.4. Logging the Java language server protocol (LSP) messages

Procedure

To log the LSP messages to the VS Code **Output** view, enable tracing by setting the **java.trace.server** attribute to **verbose**.

Additional resources

For troubleshooting instructions, see the VS Code Java Github repository .

2.4.2.9. Viewing logs for Intelephense

2.4.2.9.1. Logging the Intelephense client-server communication

Procedure

To configure the PHP Intelephense language support to log the client-server interexchange in the **Output** view:

1. Click **File → Settings**.

2. Open the **Preferences** view.

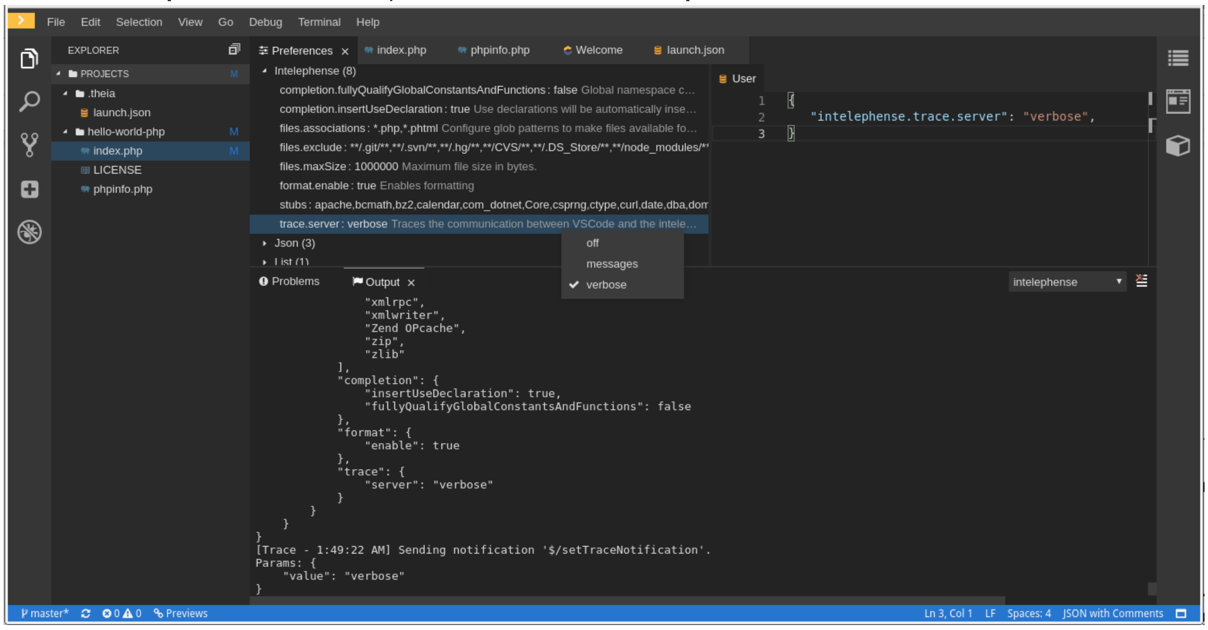
3. Expand the **Intelephense** section and set the **trace.server.verbose** preference value to **verbose** to see all the communication events (the default value is **off**).

2.4.2.9.2. Viewing Intelephense events in the Output panel

This procedure describes how to view Intelephense events in the **Output** panel.

Procedure

1. Click **View → Output**
2. Select **Intelephense** in the drop-down list for the **Output** view.



```
Intelephense (8)
    completion.fullyQualifyGlobalConstantsAndFunctions : false Global namespace c…
    completion.insertUseDeclaration : true Use declarations will be automatically inse…
    files.associations : *.php,*.phtml Configure glob patterns to make files available fo…
    files.exclude : **/.git/**,**/.svn/**,**/.hg/**,**/CVS/**,**/.DS_Store/**,**/node_modules/**
    files.maxSize : 1000000 Maximum file size in bytes.
    format.enable : true Enables formatting
    stubs : apache,bcmath,bz2,calendar,com_dotnet,Core,csprng,ctype,curl,date,dba,dom
    trace.server : verbose Traces the communication between VSCode and the intele…
Json (3)
List (1)
```

```
{
    "intelephense.trace.server": "verbose",
}
```

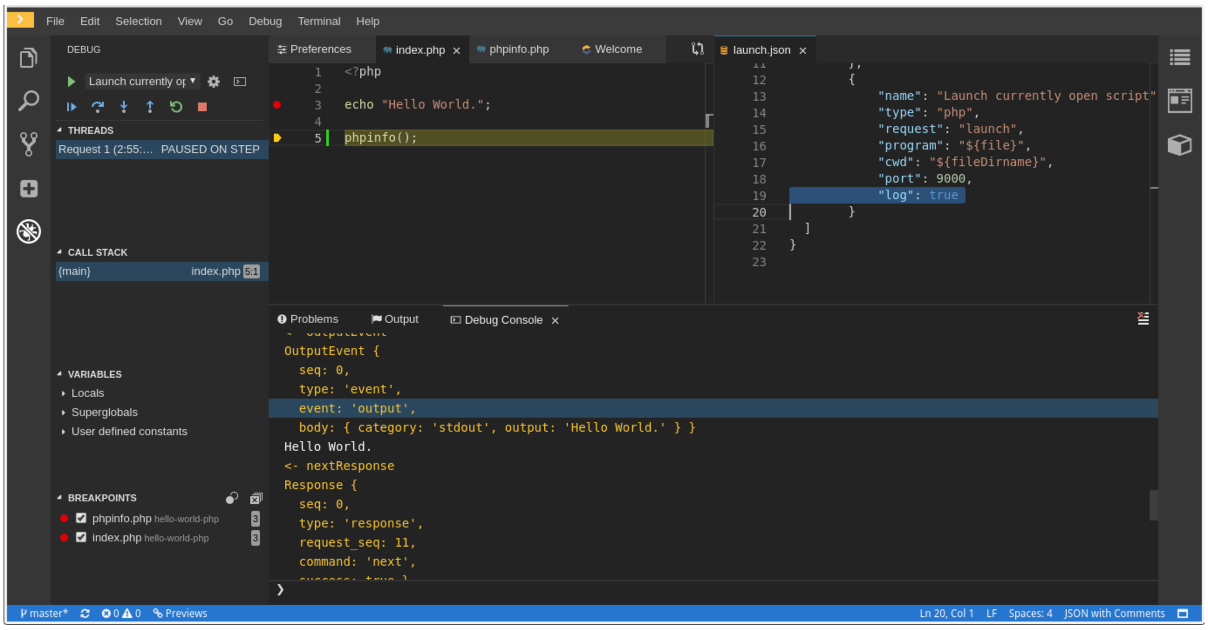
```
                "xmlrpc",
                "xmlwriter",
                "Zend OPcache",
                "zip",
                "zlib"
            ],
            "completion": {
                "insertUseDeclaration": true,
                "fullyQualifyGlobalConstantsAndFunctions": false
            },
            "format": {
                "enable": true
            },
            "trace": {
                "server": "verbose"
            }
        }
    }
}
[Trace - 1:49:22 AM] Sending notification '$/setTraceNotification'.
Params: {
    "value": "verbose"
}
```

2.4.2.10. Viewing logs for PHP-Debug

This procedure describes how to configure the PHP Debug plug-in to log the PHP Debug plug-in diagnostic messages into the **Debug Console** view. Configure this before the start of the debug session.

Procedure

1. In the **launch.json** file, add the **"log": true** attribute to the selected launch configuration.
2. Start the debug session.
3. The diagnostic messages are printed into the **Debug Console** view along with the application output.



```
<?php

echo "Hello World.";

phpinfo();
```

```
        {
            "name": "Launch currently open script"
            "type": "php",
            "request": "launch",
            "program": "${file}",
            "cwd": "${fileDirname}",
            "port": 9000,
            "log": true
        }
    ]
}
```

```
OutputEvent {
  seq: 0,
  type: 'event',
  event: 'output',
  body: { category: 'stdout', output: 'Hello World.' } }
Hello World.
<- nextResponse
Response {
  seq: 0,
  type: 'response',
  request_seq: 11,
  command: 'next',
```

2.4.2.11. Viewing logs for XML

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

2.4.2.11.1. Verifying the state of the XML language server

Procedure

1. Open a terminal in the container named **vscode-xml-*<xxx>***.
2. Run **ps aux | grep java** to verify that the XML language server has started. The output should be:

   ```
   java ***/org.eclipse.ls4xml-uber.jar`
   ```

   If not, see Section 2.4.2.1, “Checking important logs”.

2.4.2.11.2. Checking XML language server feature flags

Procedure

1. Check if the features are enabled. The XML plug-in provides multiple settings that can enable and disable features:
   - **xml.format.enabled**: Enable the formatter
   - **xml.validation.enabled**: Enable the validation
   - **xml.documentSymbols.enabled**: Enable the document symbols
2. To diagnose whether the XML language server is working, create a simple XML element, such as **<hello></hello>**, and confirm that it appears in the **Outline** panel on the right.
3. If the document symbols do not show, ensure that the **xml.documentSymbols.enabled** attribute is set to **true**. If it is **true**, and there are no symbols, the language server may not be hooked to the editor. If there are document symbols, then the language server is connected to

the editor.

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

2.4.2.11.3. Enabling XML Language Server Protocol (LSP) tracing

Procedure

To log LSP messages to the VS Code **Output** view, enable tracing by setting the **xml.trace.server** attribute to **verbose**.

2.4.2.11.4. Viewing the XML language server log

Procedure

The log from the language server can be found in the plug-in sidecar at **/home/theia/.theia/workspace-storage/<workspace_name>/redhat.vscode-xml/lsp4xml.log**.

2.4.2.12. Viewing logs for YAML

This section describes the YAML plug-in specific actions that the user can perform, in addition to the general diagnostics ones.

2.4.2.12.1. Verifying the state of the YAML language server

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

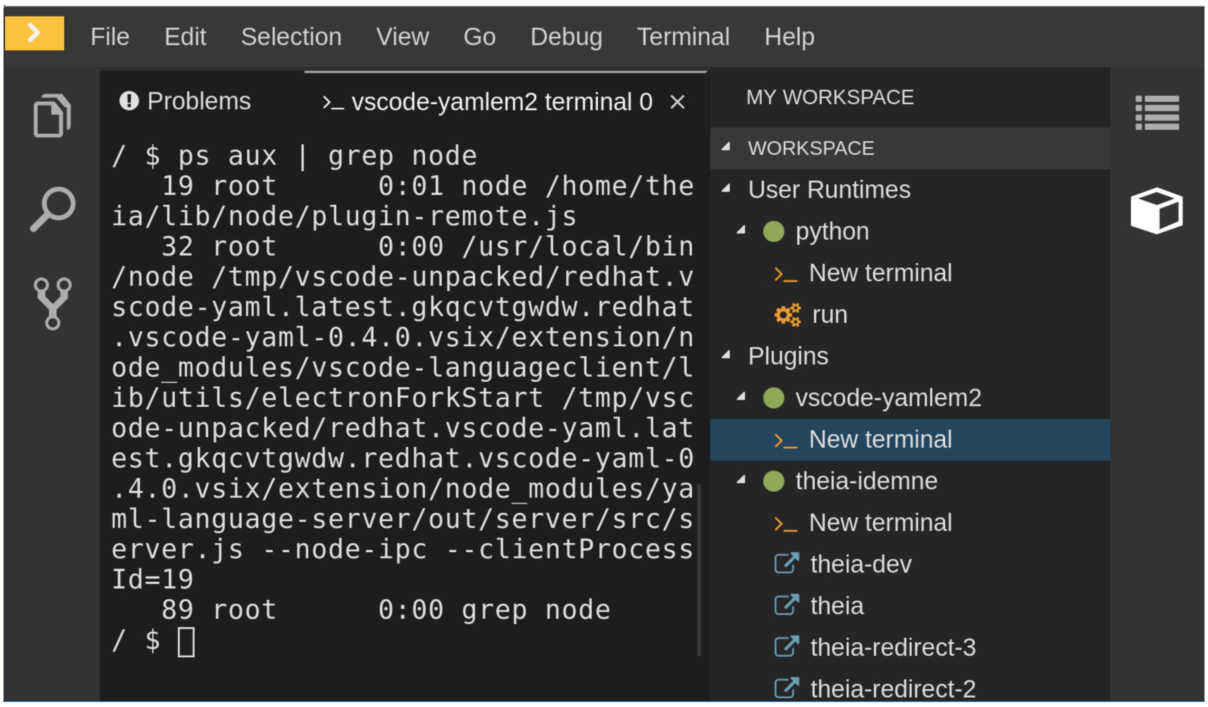
Procedure

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

1. In the editor, open a terminal in the container that is running the YAML plug-in (an example name of the container: **vscode-yaml-*<xxx>***).

2. In the terminal, run the **ps aux | grep node** command. This command searches all the node processes running in the current container.

3. Verify that a command **node **/server.js is running.**



```
/ $ ps aux | grep node
   19 root      0:01 node /home/the
ia/lib/node/plugin-remote.js
   32 root      0:00 /usr/local/bin
/node /tmp/vscode-unpacked/redhat.v
scode-yaml.latest.gkqcvtgwdw.redhat
.vscode-yaml-0.4.0.vsix/extension/n
ode_modules/vscode-languageclient/l
ib/utils/electronForkStart /tmp/vsc
ode-unpacked/redhat.vscode-yaml.lat
est.gkqcvtgwdw.redhat.vscode-yaml-0
.4.0.vsix/extension/node_modules/ya
ml-language-server/out/server/src/s
erver.js --node-ipc --clientProcess
Id=19
   89 root      0:00 grep node
/ $ 
```

The **node** **/**server.js** running in the container indicates that the language server is running. If it is not running, the language server has not started inside the container. In this case, see Section 2.4.2.1, “Checking important logs”.

2.4.2.12.2. Checking the YAML language server feature flags

Procedure

To check the feature flags:

1. Check if the features are enabled. The YAML plug-in provides multiple settings that can enable and disable features, such as:
   - **yaml.format.enable**: Enables the formatter
   - **yaml.validate**: Enables validation
   - **yaml.hover**: Enables the hover function
   - **yaml.completion**: Enables the completion function
2. To check if the plug-in is working, type the simplest YAML, such as **hello: world**, and then open the Outline panel on the right side of the editor.
3. Verify if there are any document symbols. If yes, the language server is connected to the editor.
4. If any feature is not working, make sure that the settings listed above are set to **true** (they are set to **true** by default). If a feature is not working, file an issue against the Language Server.

2.4.2.12.3. Enabling YAML Language Server Protocol (LSP) tracing

Procedure

To log LSP messages to the VS Code Output view, enable tracing by setting the **yaml.trace.server** attribute to **verbose**.

2.4.2.13. Viewing logs for Dotnet with Omnisharp-Theia plug-in

2.4.2.13.1. Omnisharp-Theia plug-in

CodeReady Workspaces uses the Omnisharp-Theia plug-in as a remote plug-in. It is located at github.com/redhat-developer/omnisharp-theia-plugin. In case of an issue, report it, or contribute your fix in the repository.

This plug-in registers omnisharp-roslyn as a language server and provides project dependencies and language syntax for C# applications.

The language server runs on .Net SDK 2.2.105.

2.4.2.13.2. Verifying the state of the Omnisharp-Theia plug-in language server

Procedure

To check if the container running the Omnisharp-Theia plug-in is running OmniSharp, execute the **ps aux | grep OmniSharp.exe** command. If the process is running, the following is an example output:

```
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/bin/mono
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/omnisharp/OmniSharp.exe
```

If the output is different, the language server has not started inside the container. Check the logs described in Section 2.4.2.1, “Checking important logs”.

2.4.2.13.3. Checking Omnisharp Che-Theia plug-in language server features

Procedure

- If the OmniSharp.exe process is running, check if the language server features are working by opening a **.cs** file and trying the hover or completion features, or opening the Problems or Outline view.

2.4.2.13.4. Viewing Omnisharp-Theia plug-in logs in the Output panel

Procedure

If **Omnisharp.exe** is running, it logs all information in the Output panel. To view the logs, open the Output view and select C# from the drop-down list.

2.4.2.14. Viewing logs for Dotnet with NetcoredebugOutput plug-in

2.4.2.14.1. NetcoredebugOutput plug-in

The NetcoredebugOutput plug-in provides netcoredbg, which implements the VS Code Debug Adapter protocol and allows users to debug .NET applications under the .NET Core runtime.

Dotnet SDK v.2.2.105 is installed in the container where the Netcoredbg plug-in is running.

2.4.2.14.2. Verifying the state of the NetcoredebugOutput plug-in

Procedure

To test the plug-in initialization:

1. Check if there is a netcoredbg debug configuration in the **launch.json** file. The following is an example debug configuration:

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

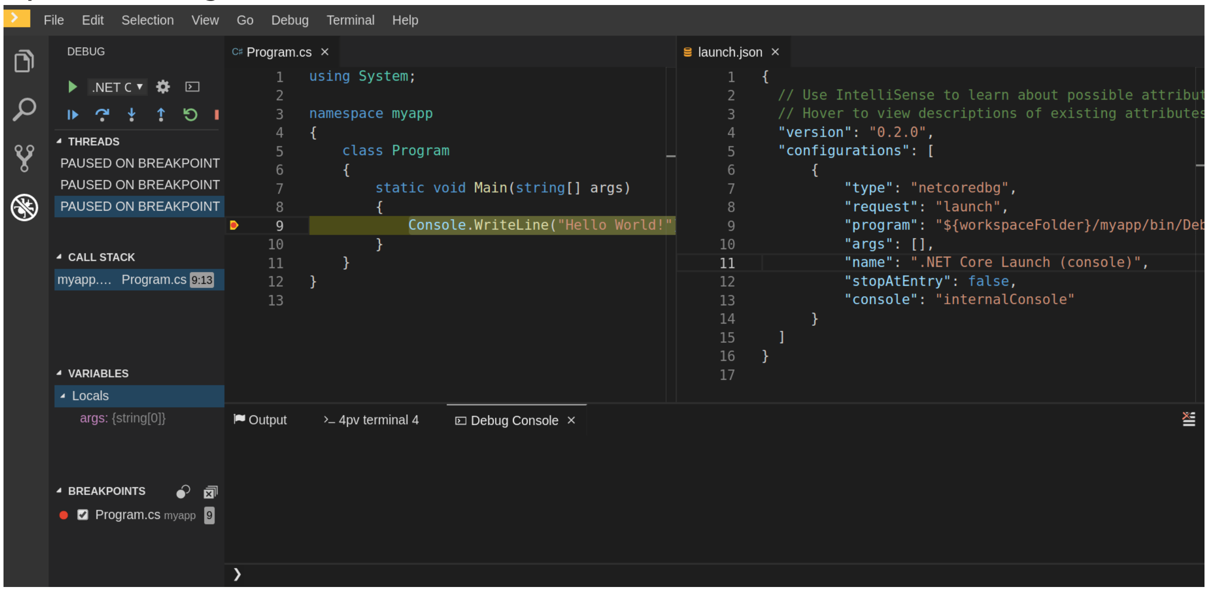
2. To test if it exists, test the autocompletion feature within the braces of the **configuration** section of the **launch.json** file. If you can find**netcoredbg**, the Che-Theia plug-in is correctly initialized. If not, see Section 2.4.2.1, “Checking important logs”.

2.4.2.14.3. Viewing NetcoredebugOutput plug-in logs in the Output panel

This section describes how to view NetcoredebugOutput plug-in logs in the Output panel.

Procedure

- Open the Debug console.



```
using System;

namespace myapp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!"
        }
    }
}
```

```
{
  // Use IntelliSense to learn about possible attribut
  // Hover to view descriptions of existing attributes
  "version": "0.2.0",
  "configurations": [
      {
          "type": "netcoredbg",
          "request": "launch",
          "program": "${workspaceFolder}/myapp/bin/Deb
          "args": [],
          "name": ".NET Core Launch (console)",
          "stopAtEntry": false,
          "console": "internalConsole"
      }
  ]
}
```

2.4.2.15. Viewing logs for Camel

2.4.2.15.1. Verifying the state of the Camel language server

Procedure

The user can inspect the log output of the sidecar container running the Camel tooling; the Camel tooling container is named: **vscode-apache-camel*<xxx>***.

To verify the state of the language server:

1. Open a terminal inside the **vscode-apache-camel*<xxx>*** container.
2. Run the **ps aux | grep java** command. The following is an example language server process:

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

3. If you cannot find it, see Section 2.4.2.1, “Checking important logs”.

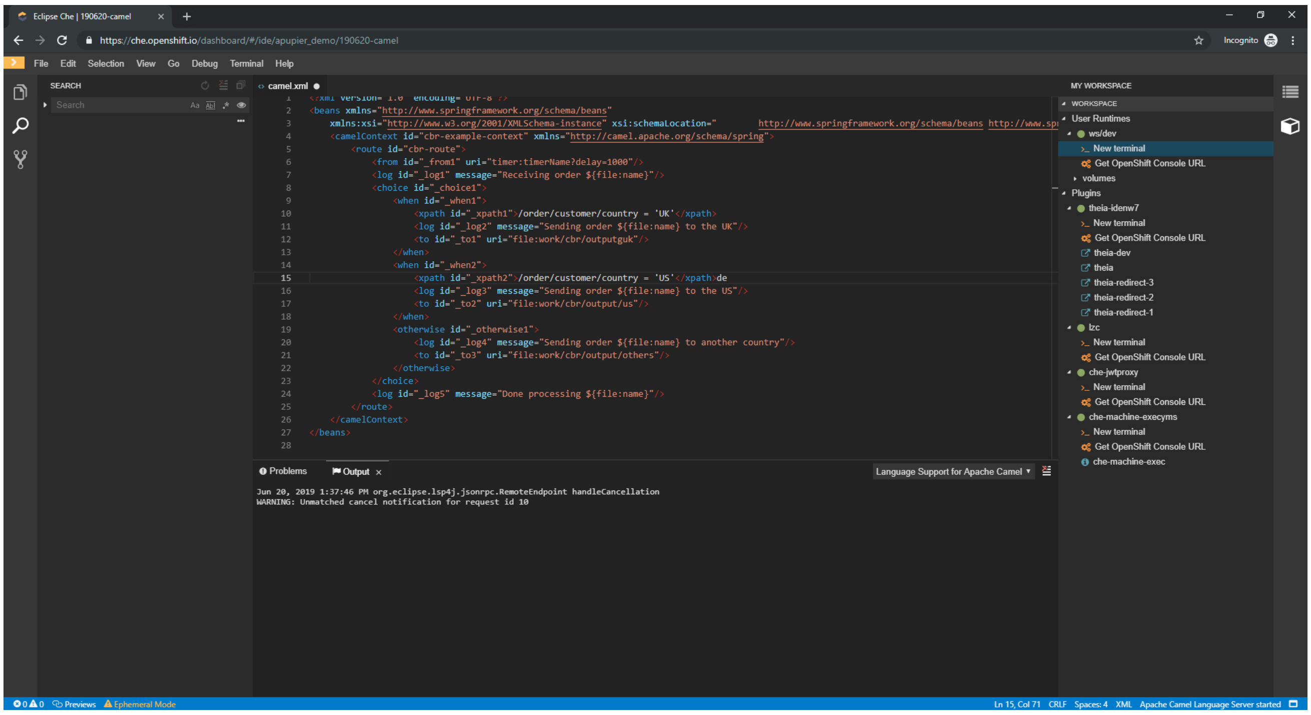
2.4.2.15.2. Viewing Camel logs in the Output panel

The Camel language server is a SpringBoot application that writes its log to the **$\{java.io.tmpdir}/log-camel-lsp.out** file. Typically,**$\{java.io.tmpdir}** points to the**/tmp** directory, so the filename is **/tmp/log-camel-lsp.out**.

Procedure

The Camel language server logs are printed in the Output channel namedLanguage Support for Apache Camel.

> **NOTE**
>
> The output channel is created only at the first created log entry on the client side. It may be absent when everything is going well.


Eclipse Che | 190620-camel
https://che.openshift.io/dashboard/#/ide/apupier_demo/190620-camel
File
Edit
Selection
View
Go
Debug
Terminal
Help
SEARCH
camel.xml
MY WORKSPACE
User Runtimes
New terminal
Get OpenShift Console URL
Plugins
Problems
Output
Language Support for Apache Camel
Ln 15, Col 71
Apache Camel Language Server started

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="        http://www.springframework.org/schema/beans http://www.spr
    <camelContext id="cbr-example-context" xmlns="http://camel.apache.org/schema/spring">
        <route id="cbr-route">
            <from id="_from1" uri="timer:timerName?delay=1000"/>
            <log id="_log1" message="Receiving order ${file:name}"/>
            <choice id="_choice1">
                <when id="_when1">
                    <xpath id="_xpath1">/order/customer/country = 'UK'</xpath>
                    <log id="_log2" message="Sending order ${file:name} to the UK"/>
                    <to id="_to1" uri="file:work/cbr/outputguk"/>
                </when>
                <when id="_when2">
                    <xpath id="_xpath2">/order/customer/country = 'US'</xpath>de
                    <log id="_log3" message="Sending order ${file:name} to the US"/>
                    <to id="_to2" uri="file:work/cbr/output/us"/>
                </when>
                <otherwise id="_otherwise1">
                    <log id="_log4" message="Sending order ${file:name} to another country"/>
                    <to id="_to3" uri="file:work/cbr/output/others"/>
                </otherwise>
            </choice>
            <log id="_log5" message="Done processing ${file:name}"/>
        </route>
    </camelContext>
</beans>
```

```
Jun 20, 2019 1:37:46 PM org.eclipse.lsp4j.jsonrpc.RemoteEndpoint handleCancellation
WARNING: Unmatched cancel notification for request id 10
```

2.5. VIEWING THE PLUG-IN BROKER LOGS

This section describes how to view the plug-in broker logs.

The logs from the plug-in broker are not persisted. As the Pod itself is deleted very quickly, events will in most cases be unavailable.

Procedure

- Logs are displayed to the user while the workspace is starting.

CHAPTER 3. MONITORING CODEREADY WORKSPACES

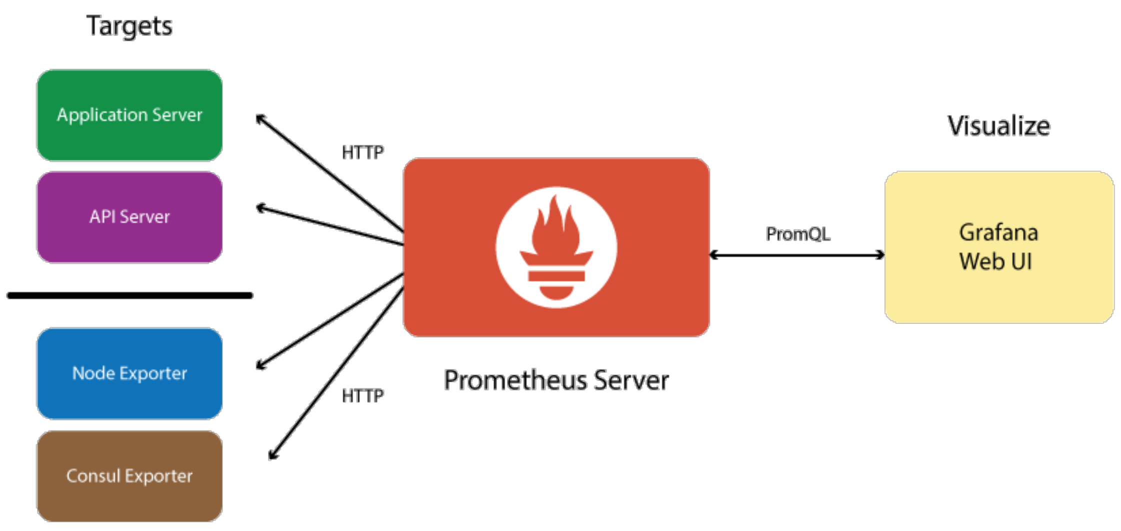
CodeReady Workspaces can expose certain data as metrics, that can be processed by Prometheus and Grafana stack. Prometheus is a monitoring system, that maintains the collection of metrics - time series key-value data which can represent consumption of resources like CPU and memory, amount of processed HTTP queries and their execution time, and CodeReady Workspaces specific resources, such as number of users and workspaces, the start and shutdown of workspaces, information about JsonRPC stack.

Prometheus is powered with a special query language, that allows manipulating the collected data, and perform various binary, vector and aggregation operations with it, to help create a more refined view on data.

While Prometheus is the central piece, responsible for scraping and storing the metrics, while Grafana offers a front-end "facade" with tools to create a various visual representation in the form of dashboards with various panels and graph types.

Note that this monitoring stack is not an official production-ready solution, but rather has an introduction purpose.

Figure 3.1. The structure of CodeReady Workspaces monitoring stack



3.1. ENABLING CODEREADY WORKSPACES METRICS COLLECTIONS

Prerequisites

- Installed Prometheus 2.9.1 or above. See more https://prometheus.io/docs/introduction/first_steps/.
- Installed Grafana 6.0 or above. See more at https://grafana.com/docs/installation/

Procedure

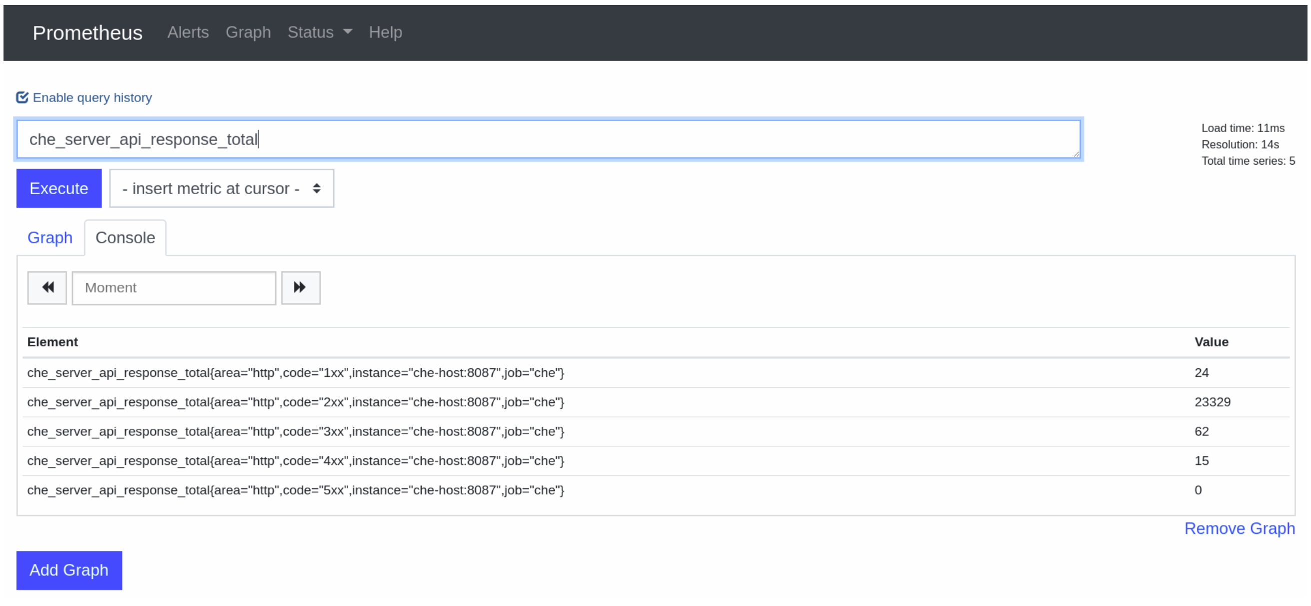
1. Set the **CHE_METRICS_ENABLED=true** environment variable
2. Expose the **8087** port as a service on the che-master host

3. Configure Prometheus to scrape metrics from the **8087** port
4. Configure a Prometheus data source on Grafana
5. Deploy CodeReady Workspaces-specific dashboards on Grafana

3.2. COLLECTING CODEREADY WORKSPACES METRICS WITH PROMETHEUS

Prometheus is a monitoring system that collects metrics in real time and stores them in a time series database.

Prometheus comes with a console accessible at the **9090** port of the application pod. By default, a template provides an existing service and aroute to access it. It can be used to query and view metrics.



che_server_api_response_total

Load time: 11ms
Resolution: 14s
Total time series: 5

Element	Value
che_server_api_response_total{area="http",code="1xx",instance="che-host:8087",job="che"}	24
che_server_api_response_total{area="http",code="2xx",instance="che-host:8087",job="che"}	23329
che_server_api_response_total{area="http",code="3xx",instance="che-host:8087",job="che"}	62
che_server_api_response_total{area="http",code="4xx",instance="che-host:8087",job="che"}	15
che_server_api_response_total{area="http",code="5xx",instance="che-host:8087",job="che"}	0

3.2.1. Prometheus terminology

Prometheus offers:

counter
: the simplest numerical type of metric whose value can be only increased. A typical example is counting the amount of HTTP requests that go through the system.

gauge
: numerical value that can be increased or decreased. Best suited for representing values of objects.

histogram
: a more complex metric that is suited for performing observations. Metrics are collected and grouped in configurable buckets, which allwos to present the results, for instance, in a form of a heatmap.

3.2.2. Configuring Prometheus

Prometheus configuration

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

1 rate, at which a target is scraped

2 rate, at which recording and alerting rules are re-checked (not used in our system at the moment)

3.3. VIEWING CODEREADY WORKSPACES METRICS ON GRAFANA DASHBOARDS

Grafana is used for informative representation of Prometheus metrics. Providing visibility for OpenShift, Grafana's deployment configuration and ConfigMaps are located in the **che-monitoring.yaml** configuration file.

3.3.1. Configuring and deploying Grafana

Grafana is run on port **3000** with a corresponding service and route.

Three ConfigMaps are used to configure Grafana:

- **grafana-datasources** – configuration for Grafana datasource, a Prometheus endpoint
- **grafana-dashboards** – configuration of Grafana dashboards and panels
- **grafana-dashboard-provider** – configuration of the Grafana dashboard provider API object, which tells Grafana where to look in the file system for pre-provisioned dashboards

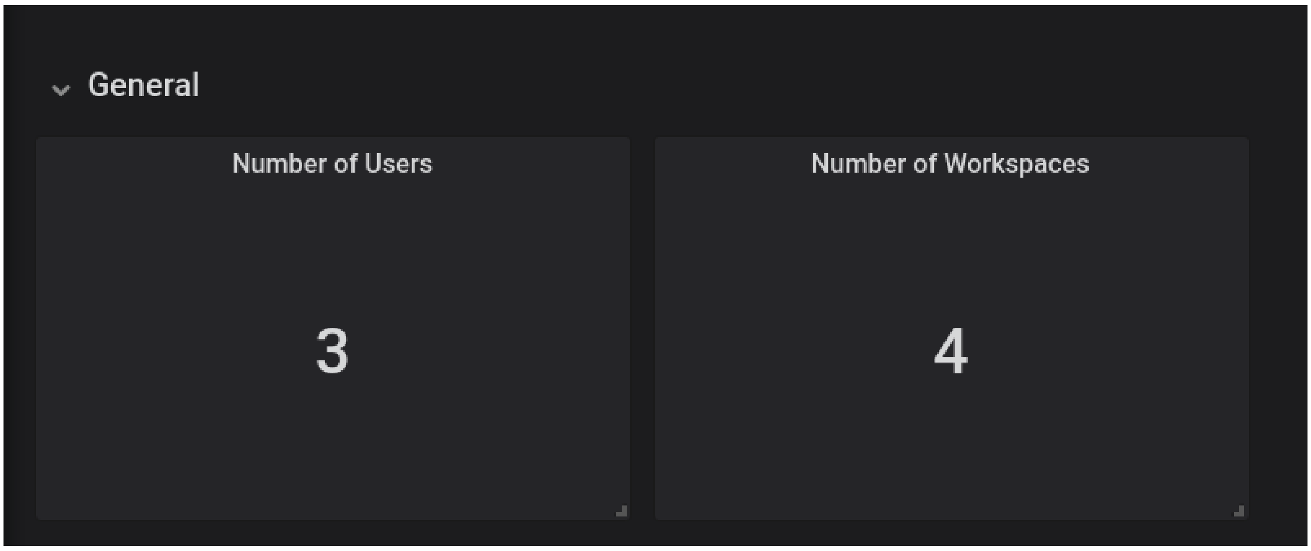
3.3.2. Grafana dashboards overview

CodeReady Workspaces provides several types of dashboards.

3.3.2.1. CodeReady Workspaces server dashboard

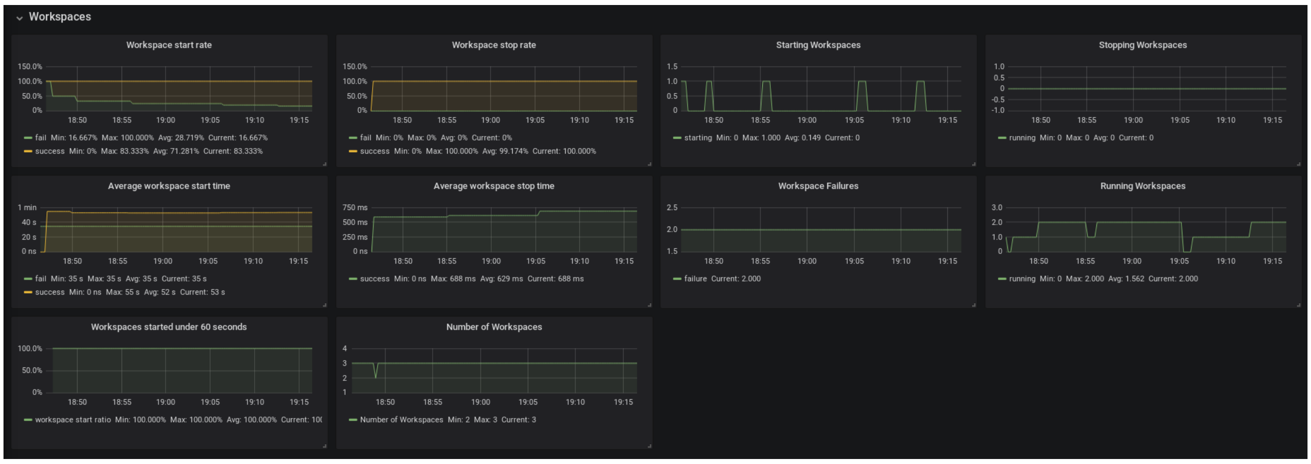
Use case: CodeReady Workspaces server-specific metrics related to CodeReady Workspaces components, such as workspaces or users.

Figure 3.2. The General panel



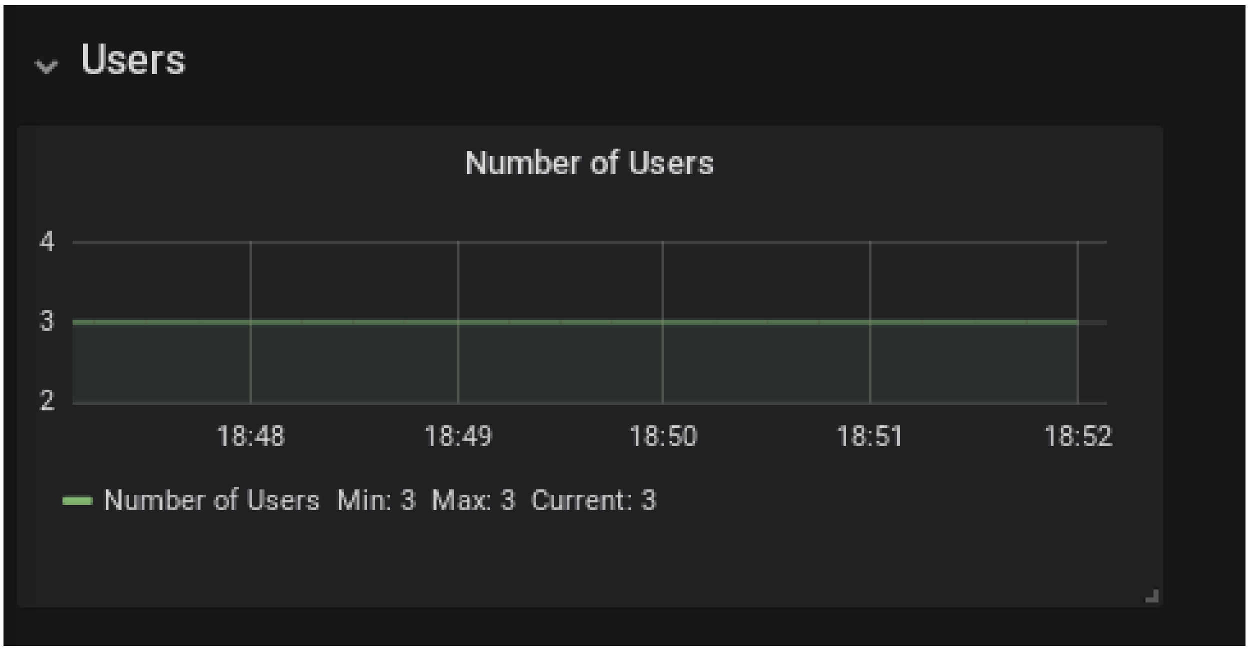
The General panel contains basic information, such as the total number of users and workspaces in the CodeReady Workspaces database.

Figure 3.3. The Workspaces panel


Workspaces
Workspace start rate
fail Min: 16.667% Max: 100.000% Avg: 28.719% Current: 16.667%
success Min: 0% Max: 83.333% Avg: 71.281% Current: 83.333%
Workspace stop rate
fail Min: 0% Max: 0% Avg: 0% Current: 0%
success Min: 0% Max: 100.000% Avg: 99.174% Current: 100.000%
Starting Workspaces
starting Min: 0 Max: 1.000 Avg: 0.149 Current: 0
Stopping Workspaces
running Min: 0 Max: 0 Avg: 0 Current: 0
Average workspace start time
fail Min: 35 s Max: 35 s Avg: 35 s Current: 35 s
success Min: 0 ns Max: 55 s Avg: 52 s Current: 53 s
Average workspace stop time
success Min: 0 ns Max: 688 ms Avg: 629 ms Current: 688 ms
Workspace Failures
failure Current: 2.000
Running Workspaces
running Min: 0 Max: 2.000 Avg: 1.562 Current: 2.000
Workspaces started under 60 seconds
workspace start ratio Min: 100.000% Max: 100.000% Avg: 100.000% Current: 10(
Number of Workspaces
Number of Workspaces Min: 2 Max: 3 Current: 3

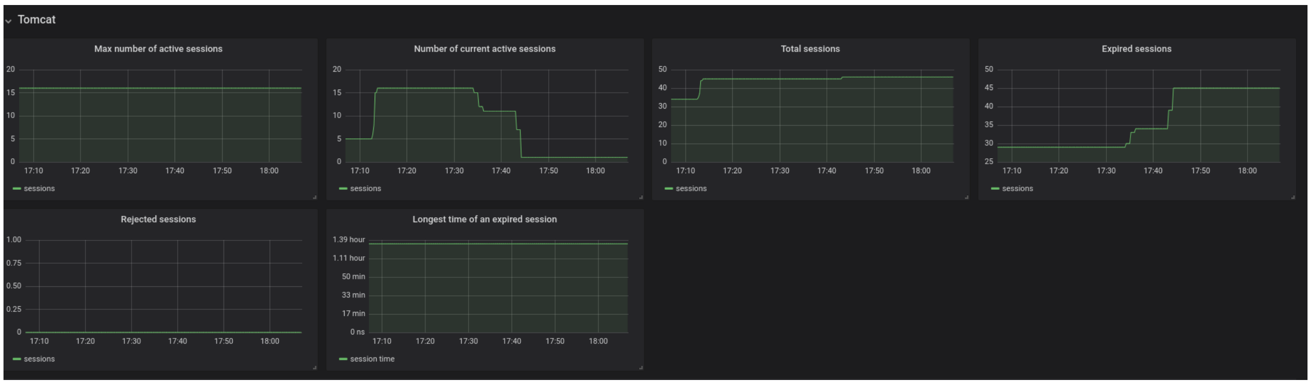
- Workspace start rate — the ratio between successful and failed started workspaces
- Workspace stop rate — the ratio between successful and failed stopped workspaces
- Workspace Failures — the number of workspace failures shown on the graph
- Starting Workspaces — the gauge that shows the number of currently starting workspaces
- Average Workspace Start Time — 1-hour average of workspace starts or fails
- Average Workspace Stop Time — 1-hour average of workspace stops
- Running Workspaces — the gauge that shows the number of currently running workspaces
- Stopping Workspaces — the gauge that shows the number of currently stopping workspaces
- Workspaces started under 60 seconds — the percentage of workspaces started under 60 seconds
- Number of Workspaces — the number of workspaces created over time

Figure 3.4. The Users panel



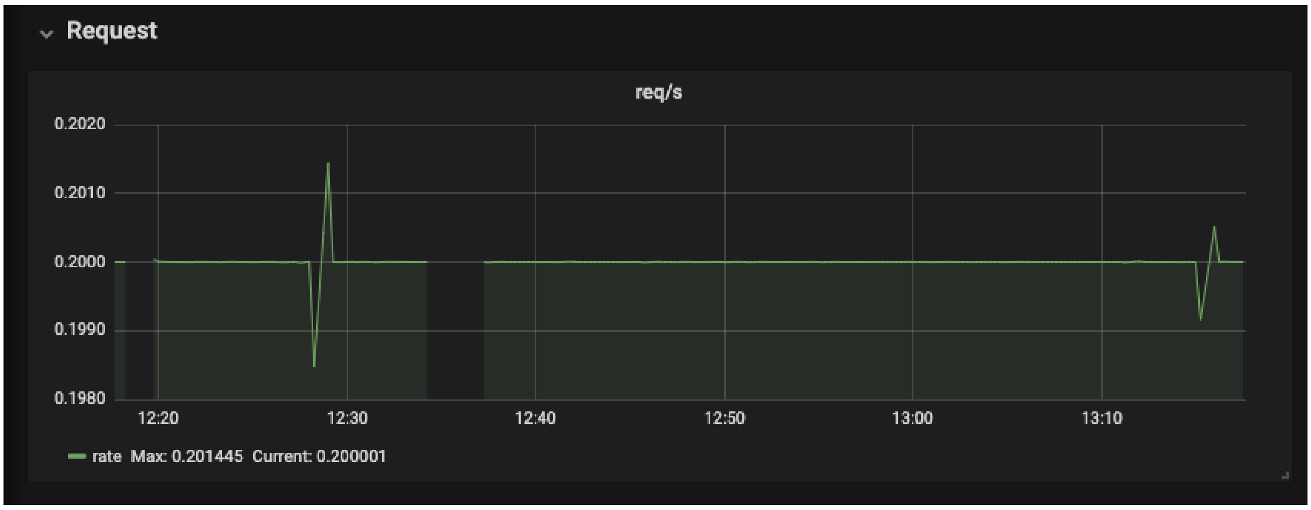
- Number of Users — the number of users known to CodeReady Workspaces over time

Figure 3.5. The Tomcat panel



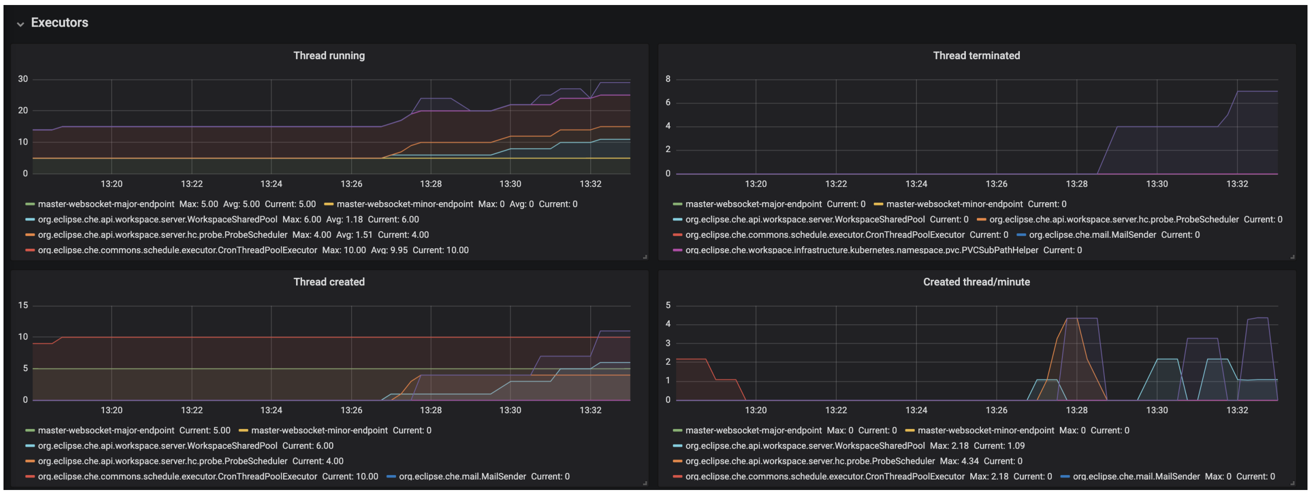
- Max number of active sessions — the max number of active sessions that have been active at the same time
- Number of current active sessions — the number of currently active sessions
- Total sessions — the total number of sessions
- Expired sessions — the number of sessions that have expired
- Rejected sessions — the number of sessions that were not created because the maximum number of active sessions was reached
- Longest time of an expired session — the longest time (in seconds) that an expired session had been alive

Figure 3.6. The Request panel



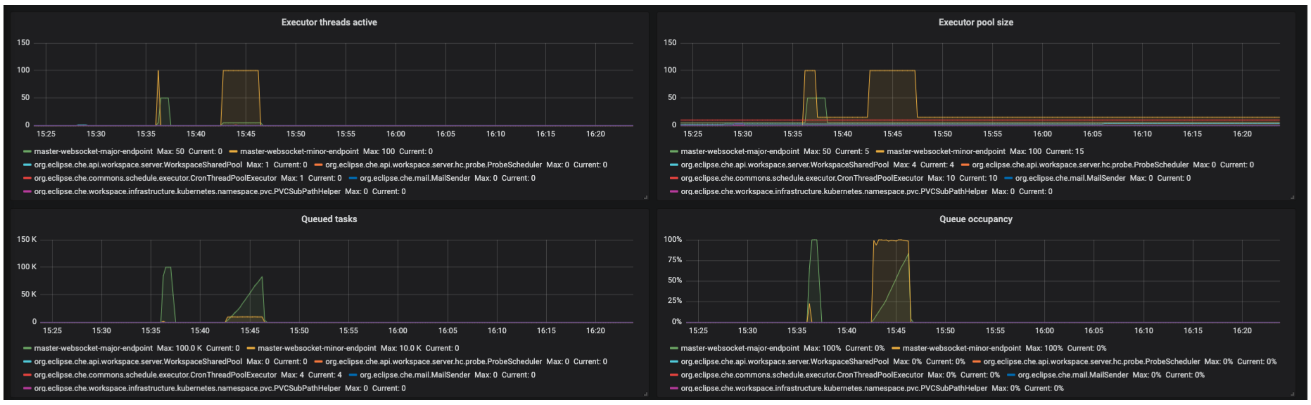
The Requests panel displays HTTP requests in a graph that shows the average number of requests per minute.

Figure 3.7. The Executors panel, part 1



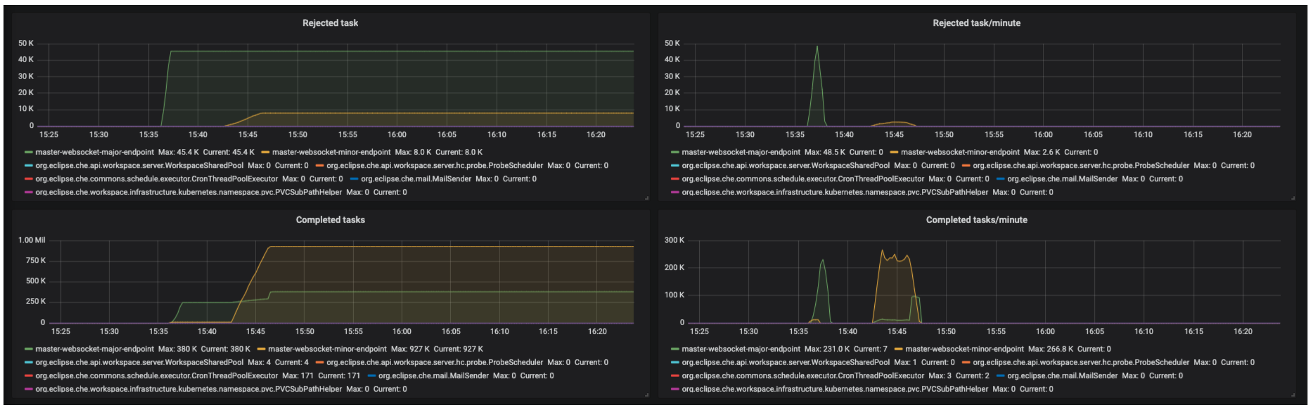
- Threads running - the number of threads that are not terminated aka alive. May include threads that are in a waiting or blocked state.
- Threads terminated - the number of threads that was finished its execution.
- Threads created - number of threads created by thread factory for given executor service.
- Created thread/minute - Speed of thread creating for the given executor service.

Figure 3.8. The Executors panel, part 2



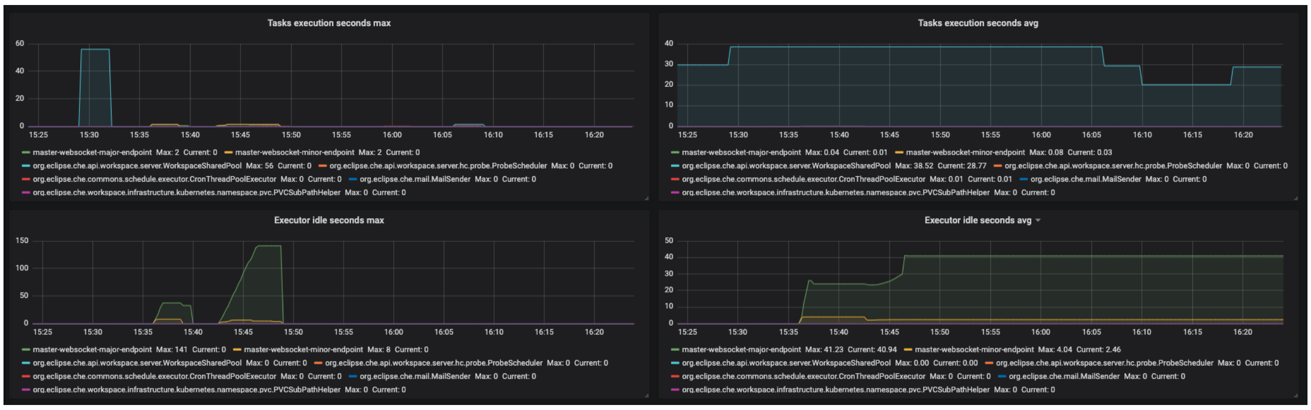
- Executor threads active - number of threads that actively execute tasks.
- Executor pool size - number of threads that actively execute tasks.
- Queued task - the approximate number of tasks that are queued for execution
- Queued occupancy - the percent of the queue used by the tasks that is waining for execution.

Figure 3.9. The Executors panel, part 3



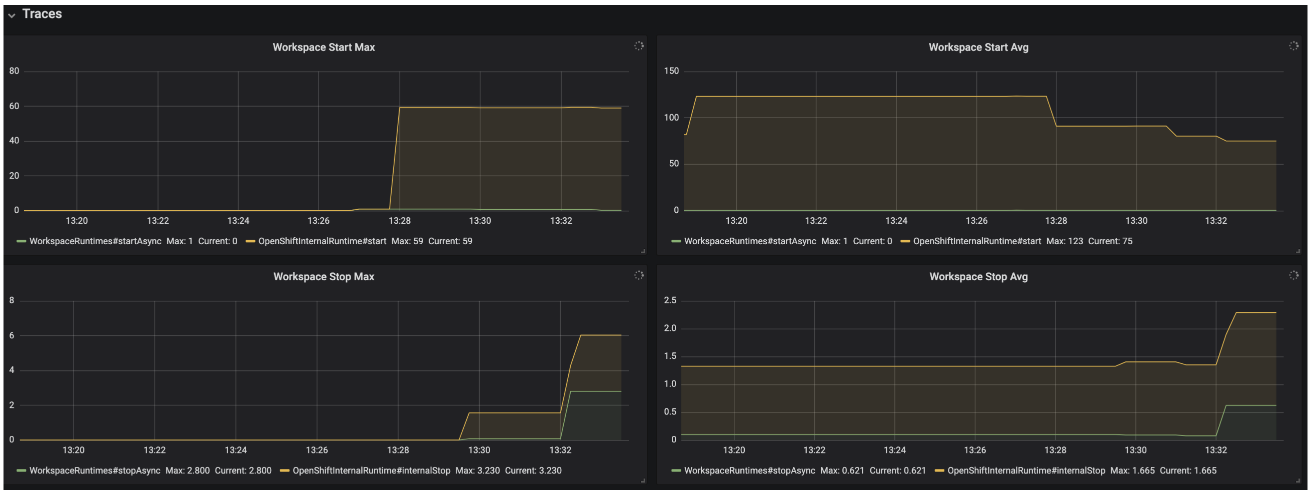
- Rejected task - the number of tasks that were rejected from execution.
- Rejected task/minute - the speed of task rejections
- Completed tasks - the number of completed tasks
- Completed tasks/minute - the speed of task execution

Figure 3.10. The Executors panel, part 4



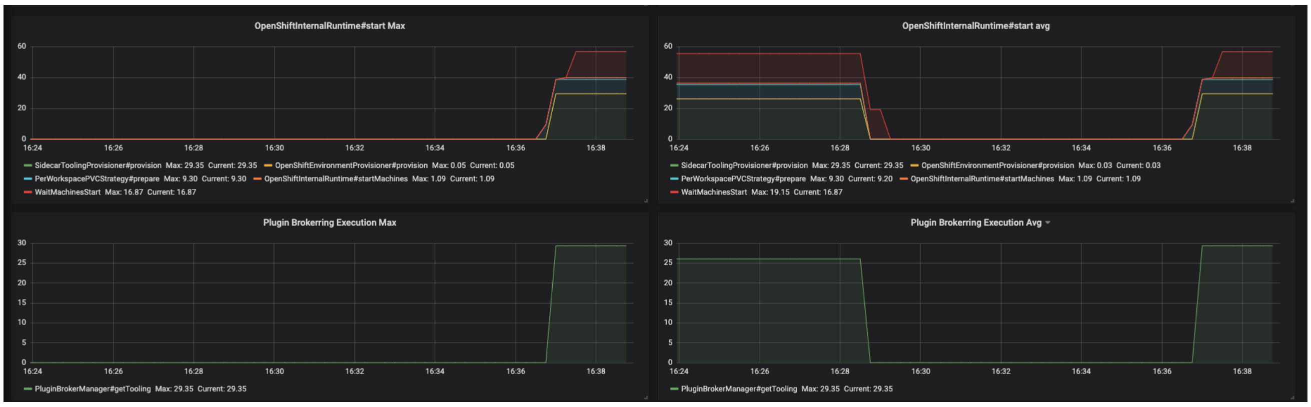
- Task execution seconds max - 5min moving maximum of task execution
- Tasks execution seconds avg - 1h moving average of task execution
- Executor idle seconds max - 5min moving maximum of executor idle state.
- Executor idle seconds avg - 1h moving average of executor idle state.

Figure 3.11. The Traces panel, part 1



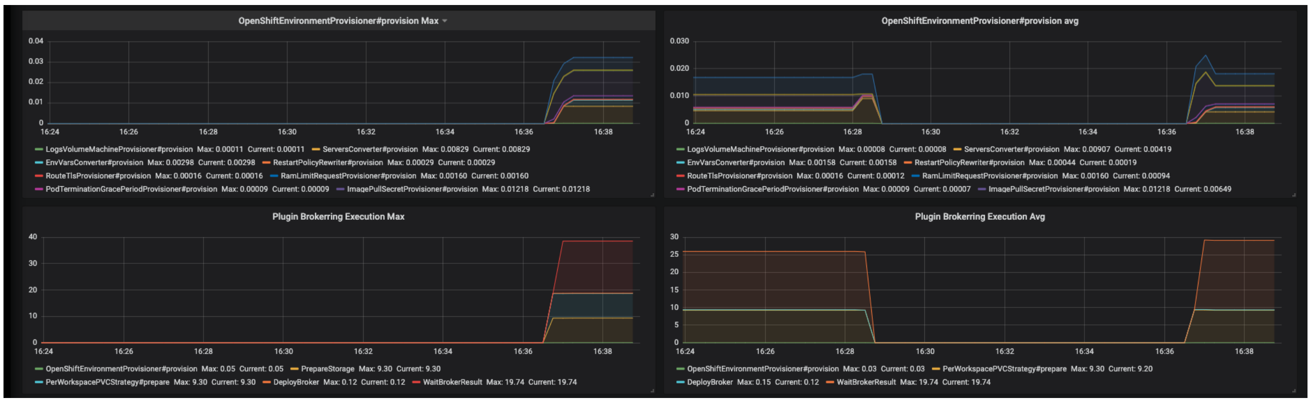
- Workspace start Max - maximum workspace start time
- Workspace start Avg - 1h moving average of the workspace start time components
- Workspace stop Max - maximum of workspace stop time
- Workspace stop Avg - 1h moving average of the workspace stop time components

Figure 3.12. The Traces panel, part 2



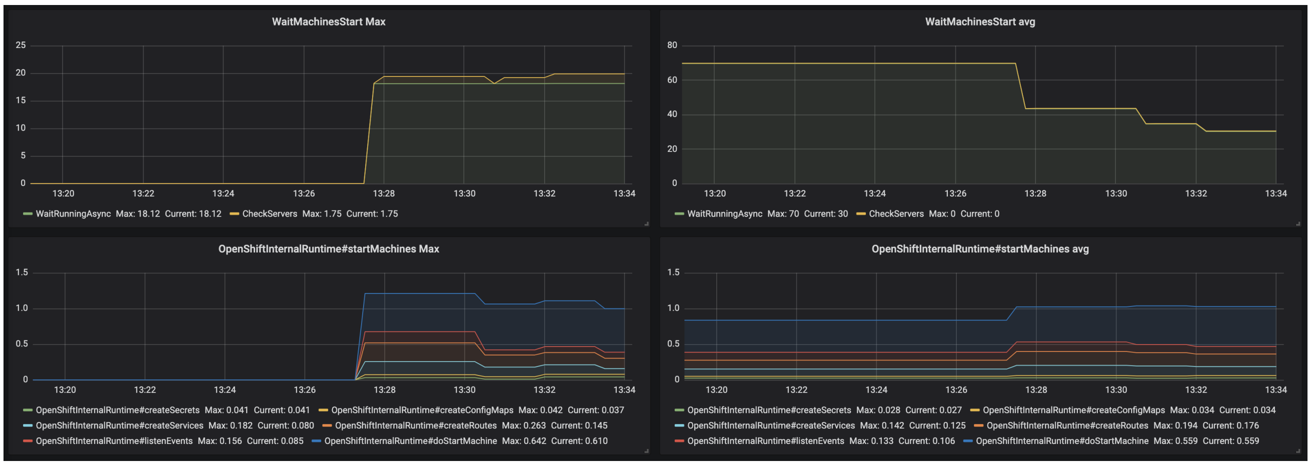
- OpenShiftInternalRuntime#start Max - maximum time of OpenShiftInternalRuntime#start operation
- OpenShiftInternalRuntime#start Avg - 1h moving average time of OpenShiftInternalRuntime#start operation
- Plugin Brokering Execution Max - maximum time of PluginBrokerManager#getTooling operation
- Plugin Brokering Execution Avg - 1h moving average of PluginBrokerManager#getTooling operation

Figure 3.13. The Traces panel, part 3



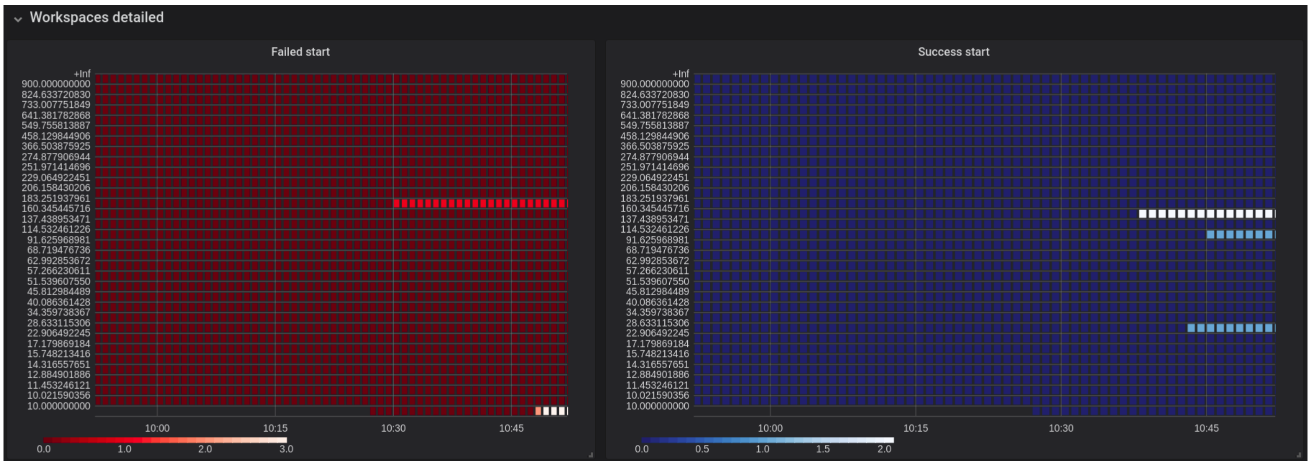
- OpenShiftEnvironmentProvisioner#provision Max - maximum time of OpenShiftEnvironmentProvisioner#provision operation
- OpenShiftEnvironmentProvisioner#provision Avg -1h moving average of OpenShiftEnvironmentProvisioner#provision operation
- Plugin Brokering Execution Max - maximum time of PluginBrokerManager#getTooling components execution time
- Plugin Brokering Execution Avg - 1h moving average of time of PluginBrokerManager#getTooling components execution time

Figure 3.14. The Traces panel, part 4



- WaitMachinesStart Max - maximim time of WaitMachinesStart operations
- WaitMachinesStart Avg - 1h moving average time of WaitMachinesStart operations
- OpenShiftInternalRuntime#startMachines Max - maximim time of OpenShiftInternalRuntime#startMachines operations
- OpenShiftInternalRuntime#startMachines Avg - 1h moving average of the time of OpenShiftInternalRuntime#startMachines operations

Figure 3.15. The Workspace detailed panel



The Workspace Detailed panel contains heat maps, which illustrate the average time of workspace starts or fails. The row shows some period of time.

3.3.2.2. CodeReady Workspaces server JVM dashboard

Use case: JVM metrics of the CodeReady Workspaces server, such as JVM memory or classloading.

Figure 3.16. CodeReady Workspaces server JVM dashboard

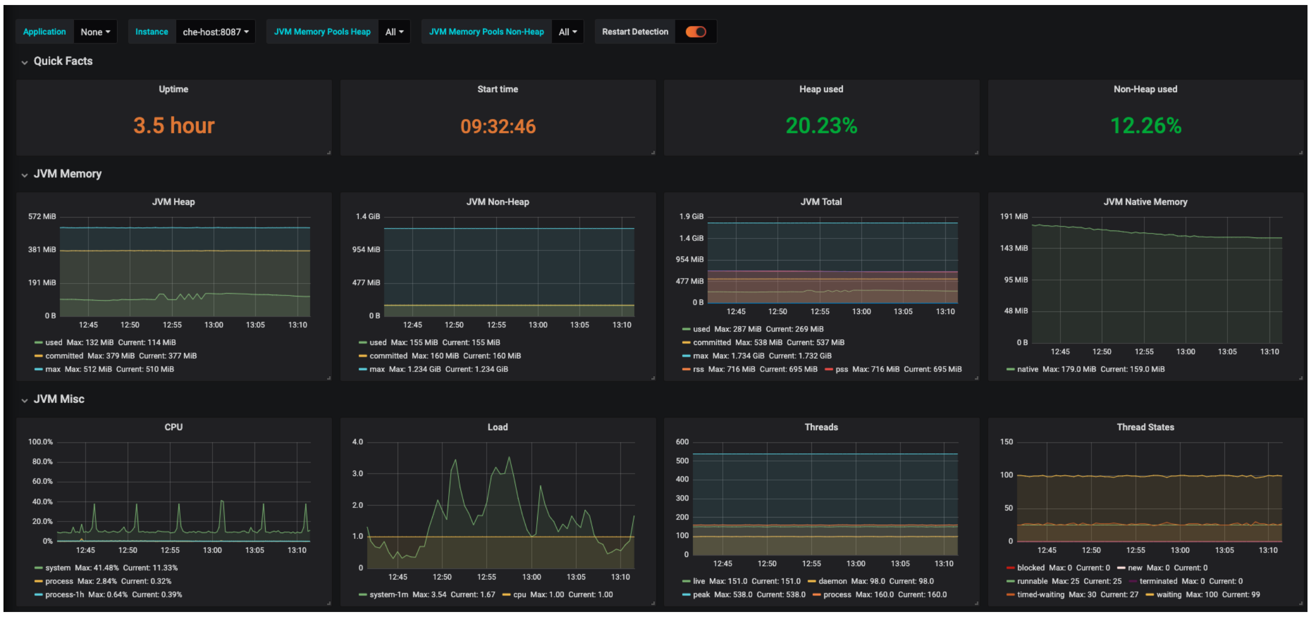

Application None
Instance che-host:8087
JVM Memory Pools Heap All
JVM Memory Pools Non-Heap All
Restart Detection
Quick Facts
Uptime 3.5 hour
Start time 09:32:46
Heap used 20.23%
Non-Heap used 12.26%
JVM Memory
JVM Heap
used Max: 132 MiB Current: 114 MiB
committed Max: 379 MiB Current: 377 MiB
max Max: 512 MiB Current: 510 MiB
JVM Non-Heap
used Max: 155 MiB Current: 155 MiB
committed Max: 160 MiB Current: 160 MiB
max Max: 1.234 GiB Current: 1.234 GiB
JVM Total
used Max: 287 MiB Current: 269 MiB
committed Max: 538 MiB Current: 537 MiB
max Max: 1.734 GiB Current: 1.732 GiB
rss Max: 716 MiB Current: 695 MiB
pss Max: 716 MiB Current: 695 MiB
JVM Native Memory
native Max: 179.0 MiB Current: 159.0 MiB
JVM Misc
CPU
system Max: 41.48% Current: 11.33%
process Max: 2.84% Current: 0.32%
process-1h Max: 0.64% Current: 0.39%
Load
system-1m Max: 3.54 Current: 1.67
cpu Max: 1.00 Current: 1.00
Threads
live Max: 151.0 Current: 151.0
daemon Max: 98.0 Current: 98.0
peak Max: 538.0 Current: 538.0
process Max: 160.0 Current: 160.0
Thread States
blocked Max: 0 Current: 0
new Max: 0 Current: 0
runnable Max: 25 Current: 25
terminated Max: 0 Current: 0
timed-waiting Max: 30 Current: 27
waiting Max: 100 Current: 99

Figure 3.17. Quick Facts

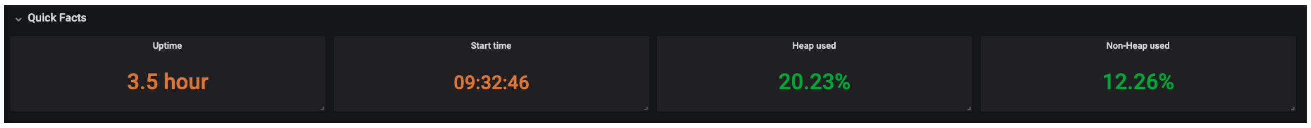


Figure 3.18. JVM Memory

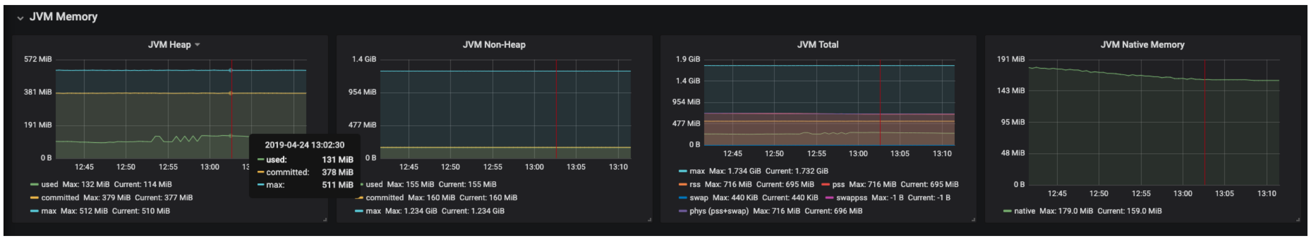

JVM Memory
JVM Heap
2019-04-24 13:02:30
used: 131 MiB
committed: 378 MiB
max: 511 MiB
used Max: 132 MiB Current: 114 MiB
committed Max: 379 MiB Current: 377 MiB
max Max: 512 MiB Current: 510 MiB
JVM Non-Heap
used Max: 155 MiB Current: 155 MiB
committed Max: 160 MiB Current: 160 MiB
max Max: 1.234 GiB Current: 1.234 GiB
JVM Total
max Max: 1.734 GiB Current: 1.732 GiB
rss Max: 716 MiB Current: 695 MiB
pss Max: 716 MiB Current: 695 MiB
swap Max: 440 KiB Current: 440 KiB
swappss Max: -1 B Current: -1 B
phys (pss+swap) Max: 716 MiB Current: 696 MiB
JVM Native Memory
native Max: 179.0 MiB Current: 159.0 MiB

Figure 3.19. JVM Misc

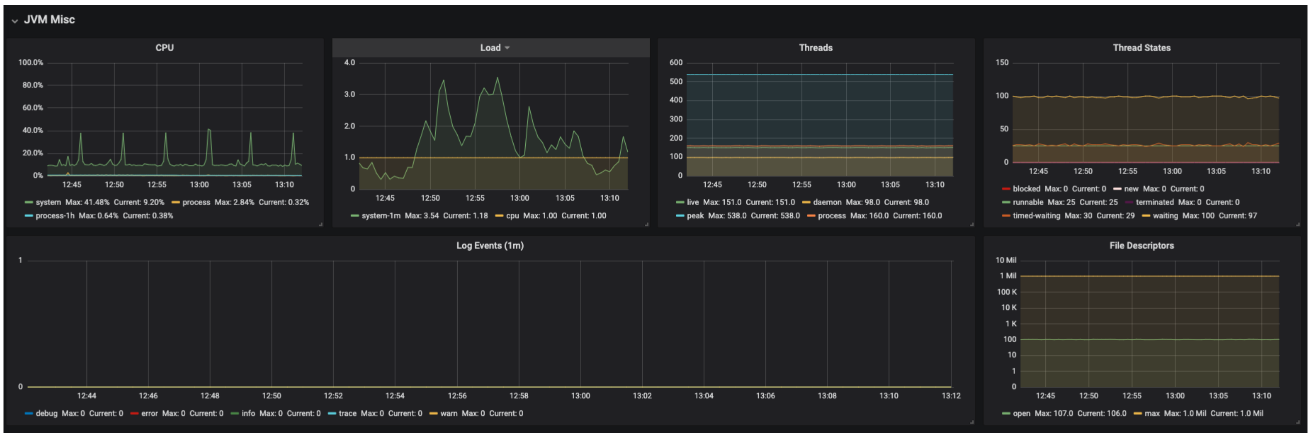

JVM Misc
CPU
system Max: 41.48% Current: 9.20%
process Max: 2.84% Current: 0.32%
process-1h Max: 0.64% Current: 0.38%
Load
system-1m Max: 3.54 Current: 1.18
cpu Max: 1.00 Current: 1.00
Threads
live Max: 151.0 Current: 151.0
daemon Max: 98.0 Current: 98.0
peak Max: 538.0 Current: 538.0
process Max: 160.0 Current: 160.0
Thread States
blocked Max: 0 Current: 0
new Max: 0 Current: 0
runnable Max: 25 Current: 25
terminated Max: 0 Current: 0
timed-waiting Max: 30 Current: 29
waiting Max: 100 Current: 97
Log Events (1m)
debug Max: 0 Current: 0
error Max: 0 Current: 0
info Max: 0 Current: 0
trace Max: 0 Current: 0
warn Max: 0 Current: 0
File Descriptors
open Max: 107.0 Current: 106.0
max Max: 1.0 Mil Current: 1.0 Mil

Figure 3.20. JVM Memory Pools (heap)

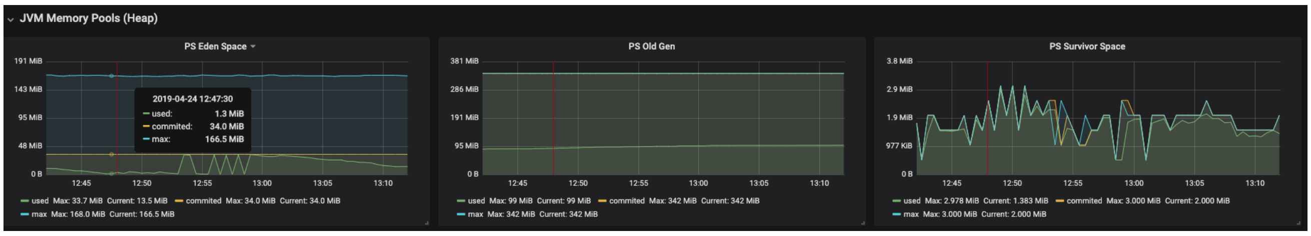


Figure 3.21. JVM Memory Pools (Non-Heap)

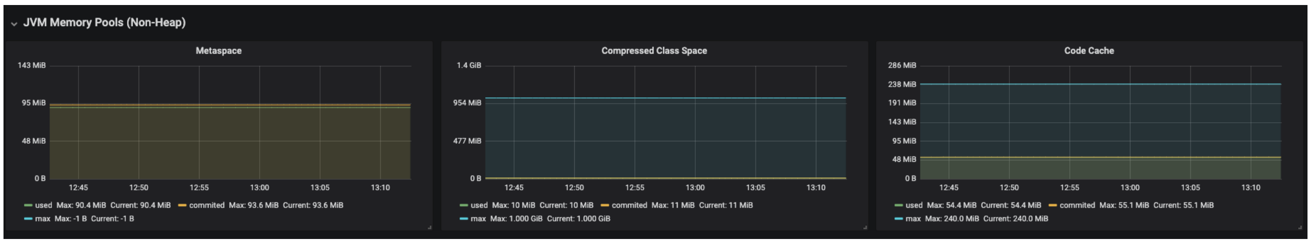


Figure 3.22. Garbage Collection

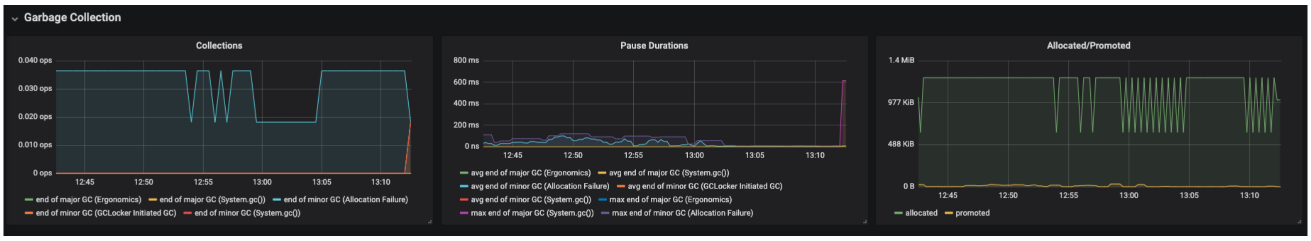


Figure 3.23. Classloading

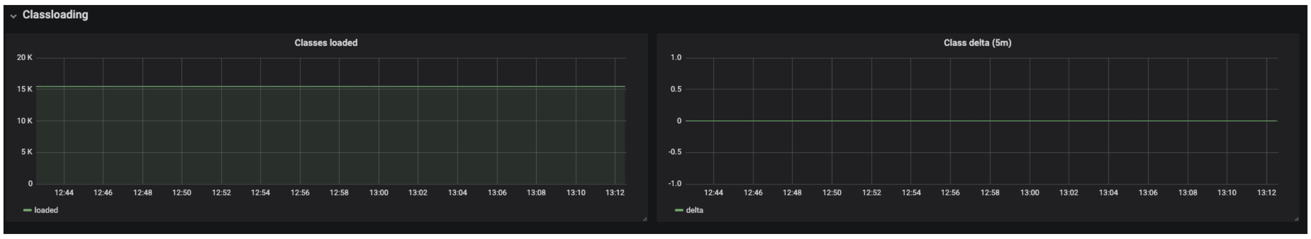
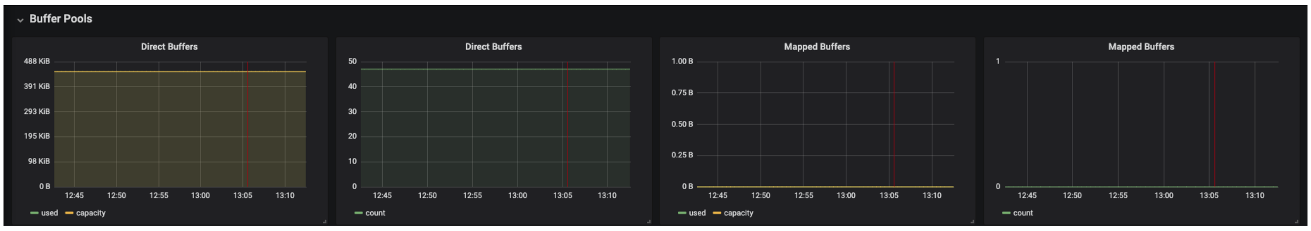


Figure 3.24. Buffer Pools



3.4. DEVELOPING GRAFANA DASHBOARDS

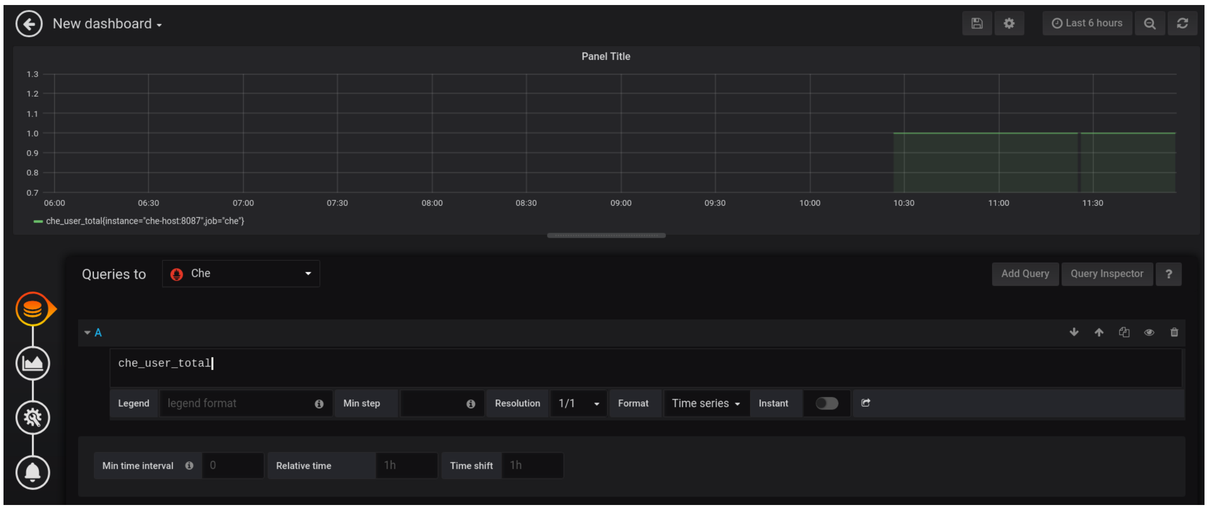
Grafana offers the possibility to add custom panels.

Procedure

To add a custom panel, use the New dashboard view.

1. In the first section, define Queries to. Use the *Prometheus Query Language* to construct a specific metric, as well as to modify it with various aggregation operators.

Figure 3.25. New Grafana dashboard: Queries to



che_user_total{instance="che-host:8087",job="che"}

Queries to: Che

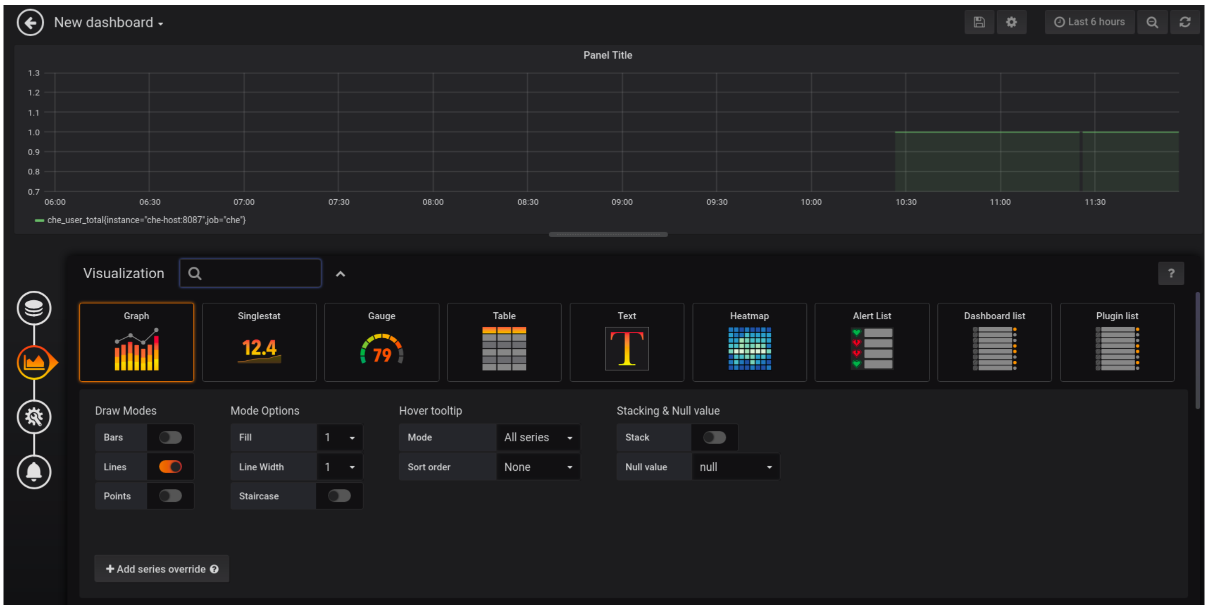
```
che_user_total
```

Legend: legend format | Min step | Resolution: 1/1 | Format: Time series | Instant

Min time interval: 0 | Relative time: 1h | Time shift: 1h

2. In the Visualisation section, choose a metric to be shown in the following visual in the form of a graph, gauge, heatmap, or others.

Figure 3.26. New Grafana dashboard: Visualization



che_user_total{instance="che-host:8087",job="che"}

Draw Modes: Bars | Lines | Points

Mode Options: Fill 1 | Line Width 1 | Staircase

Hover tooltip: Mode All series | Sort order None

Stacking & Null value: Stack | Null value null

3. Save changes to the dashboard by clicking the **Save** button, and copy and paste the JSON code to the deployment.

4. Load changes in the configuration of a running Grafana deployment. First remove the deployment:

```
$ oc process -f che-monitoring.yaml | oc delete -f -
```

Then redeploy your Grafana with the new configuration:

```
$ oc process -f che-monitoring.yaml | oc apply -f - | oc rollout latest grafana
```

3.5. EXTENDING CODEREADY WORKSPACES MONITORING METRICS

There are two major modules for metrics:

- **che-core-metrics-core** – contains core metrics module
- **che-core-api-metrics** – contains metrics that are dependent on core CodeReady Workspaces components, such as workspace or user managers

Procedure

To create a metric or a group of metrics, you need a class that extends the **MeterBinder** class. This allows to register the created metric in the overriden **bindTo(MeterRegistry registry)** method.

The following is an example of a metric that has a function that supplies the value for it:

Example metric

```
public class UserMeterBinder implements MeterBinder {

  private final UserManager userManager;

  @Inject
  public UserMeterBinder(UserManager userManager) {
    this.userManager = userManager;
  }

  @Override
  public void bindTo(MeterRegistry registry) {
    Gauge.builder("che.user.total", this::count)
        .description("Total amount of users")
        .register(registry);
  }

  private double count() {
    try {
      return userManager.getTotalCount();
    } catch (ServerException e) {
      return Double.NaN;
    }
  }
```

Alternatively, the metric can be stored with a reference and updated manually in some other place in the code.

Additional resources

For more information about the types of metrics and naming conventions, visit Prometheus documentation:

- Naming practices
- Metric types

CHAPTER 4. TRACING CODEREADY WORKSPACES

Tracing helps gather timing data to troubleshoot latency problems in microservice architectures and helps to understand a complete transaction or workflow as it propagates through a distributed system. Every transaction may reflect performance anomalies in an early phase when new services are being introduced by independent teams.

Tracing the CodeReady Workspaces application may help analyse the execution of various operations, such as workspace creations, workspace startup, breaking down the duration of sub-operations executions, helping finding bottlenecks and improve the overall state of the platform.

Tracers live in applications. They record timing and metadata about operations that take place. They often instrument libraries, so that their use is transparent to users. For example, an instrumented web server records when it received a request and when it sent a response. The trace data collected is called a span. A span has a context that contains information such as trace and span identifiers and other kinds of data that can be propagated down the line.

4.1. TRACING API

CodeReady Workspaces utilizes Opentracing API - a vendor-neutral framework for instrumentation. This means that if a developer wants to try a different tracing backend, then instead of repeating the whole instrumentation process for the new distributed tracing system, the developer can simply change the configuration of the tracer backend.

4.2. TRACING BACKEND

By default, CodeReady Workspaces uses Jaeger as the tracing backend. Jaeger was inspired by Dapper and OpenZipkin, and it is a distributed tracing system released as open source by Uber Technologies. Jaeger extends a more complex architecture for a larger scale of requests and performance.

4.3. ENABLING CODEREADY WORKSPACES METRICS COLLECTIONS

Prerequisites

- Installed Jaeger v1.12.0 or above. See instructions at Get up and running with Jaeger in your local environment.

Procedure

1. Enable the following environment variables for CodeReady Workspaces deployment:

```
# Activating {prod-short} tracing modules
CHE_TRACING_ENABLED=true

# Following variables are the basic Jaeger client library configuration.
JAEGER_ENDPOINT="http://jaeger-collector:14268/api/traces"

# Service name
JAEGER_SERVICE_NAME="che-server"

# URL to remote sampler
JAEGER_SAMPLER_MANAGER_HOST_PORT="jaeger:5778"
```

```
# Type and param of sampler (constant sampler for all traces)
JAEGER_SAMPLER_TYPE="const"
JAEGER_SAMPLER_PARAM="1"

# Maximum queue size of reporter
JAEGER_REPORTER_MAX_QUEUE_SIZE="10000"
```

Additional resources

- List of all Jaeger client environment variables

4.4. VIEWING CODEREADY WORKSPACES TRACES IN JAEGER UI

This section demonstrates how to utilize the Jaeger UI to overview traces of CodeReady Workspaces operations.

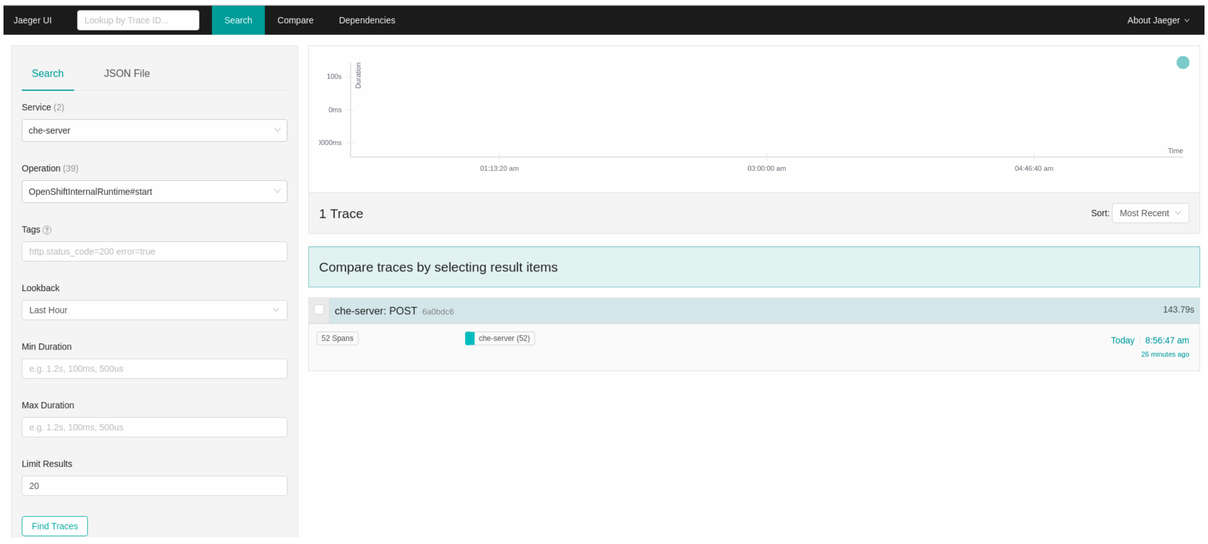
Procedure

In this example, the CodeReady Workspaces instance has been running for some time and one workspace start has occured.

To inspect the trace of the workspace start:

1. In the Search panel on the left, filter spans by the operation name (span name), tags, or time and duration.

   Figure 4.1. Using Jaeger UI to trace CodeReady Workspaces



Service (2): che-server

Operation (39): OpenShiftInternalRuntime#start

Tags: http.status_code=200 error=true

Lookback: Last Hour

Min Duration: e.g. 1.2s, 100ms, 500us

Max Duration: e.g. 1.2s, 100ms, 500us

Limit Results: 20

1 Trace   Sort: Most Recent

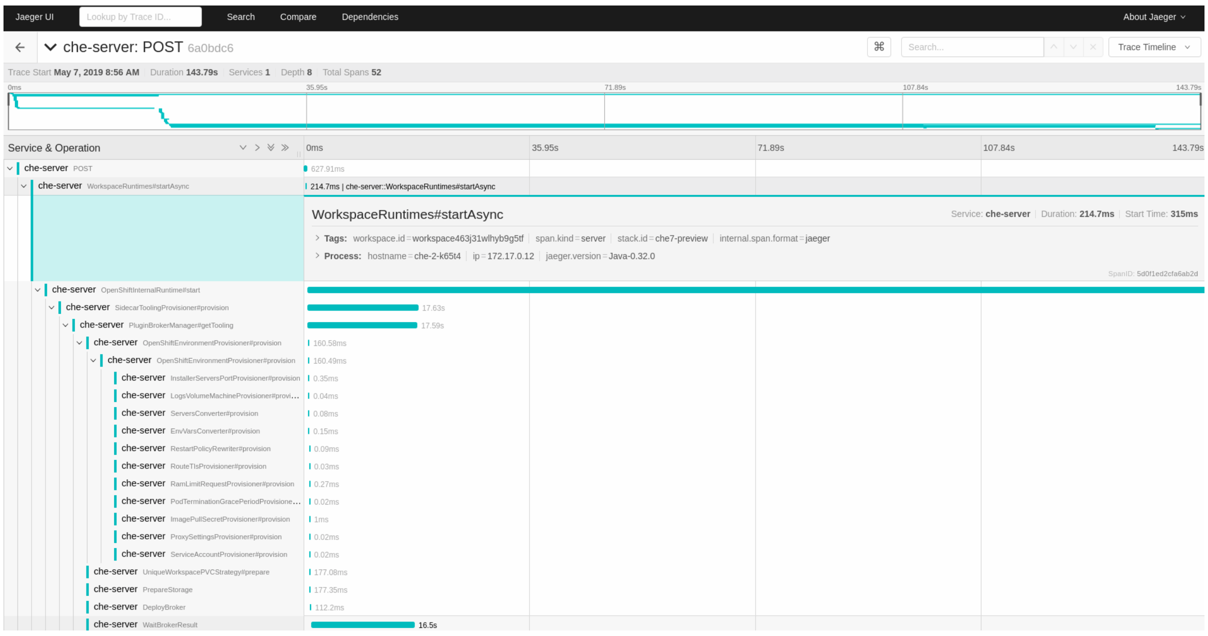
Compare traces by selecting result items

che-server: POST 6a0bdc6   143.79s

52 Spans   che-server (52)   Today | 8:56:47 am
26 minutes ago

2. Select the trace to expand it and show the tree of nested spans, as well as additional information about the highlighted span, such as tags or durations.

Figure 4.2. Expanded tracing tree



```
che-server: POST 6a0bdc6
Trace Start May 7, 2019 8:56 AM   Duration 143.79s   Services 1   Depth 8   Total Spans 52

Service & Operation                                   0ms    35.95s    71.89s    107.84s    143.79s
che-server POST                                       627.91ms
  che-server WorkspaceRuntimes#startAsync             214.7ms | che-server::WorkspaceRuntimes#startAsync

      WorkspaceRuntimes#startAsync        Service: che-server | Duration: 214.7ms | Start Time: 315ms
      > Tags: workspace.id=workspace463j31wlhyb9g5tf | span.kind=server | stack.id=che7-preview | internal.span.format=jaeger
      > Process: hostname=che-2-k65t4 | ip=172.17.0.12 | jaeger.version=Java-0.32.0
                                                                              SpanID: 5d0f1ed2cfa6ab2d

    che-server OpenShiftInternalRuntime#start
      che-server SidecarToolingProvisioner#provision              17.63s
        che-server PluginBrokerManager#getTooling                 17.59s
          che-server OpenShiftEnvironmentProvisioner#provision    160.58ms
            che-server OpenShiftEnvironmentProvisioner#provision  160.49ms
              che-server InstallerServersPortProvisioner#provision    0.35ms
              che-server LogsVolumeMachineProvisioner#provi...        0.04ms
              che-server ServersConverter#provision                   0.08ms
              che-server EnvVarsConverter#provision                   0.15ms
              che-server RestartPolicyRewriter#provision              0.09ms
              che-server RouteTlsProvisioner#provision                0.03ms
              che-server RamLimitRequestProvisioner#provision         0.27ms
              che-server PodTerminationGracePeriodProvisione...       0.02ms
              che-server ImagePullSecretProvisioner#provision         1ms
              che-server ProxySettingsProvisioner#provision           0.02ms
              che-server ServiceAccountProvisioner#provision          0.02ms
          che-server UniqueWorkspacePVCStrategy#prepare           177.08ms
          che-server PrepareStorage                               177.35ms
          che-server DeployBroker                                 112.2ms
          che-server WaitBrokerResult                             16.5s
```

4.5. CODEREADY WORKSPACES TRACING CODEBASE OVERVIEW AND EXTENSION GUIDE

The core of the tracing implementation for CodeReady Workspaces is in the **che-core-tracing-core** and **che-core-tracing-web** modules.

All HTTP requests to the tracing API have their own trace. This is done by **TracingFilter** from the OpenTracing library, which is bound for the whole server application. Adding a **@Traced** annotation to methods causes the **TracingInterceptor** to add tracing spans for them.

4.5.1. Tagging

Spans may contain standard tags, such as operation name, span origin, error, as well as others that may help users with querying and filtering spans. Workspace-related operations (such as starting or stopping workspaces) have additional tags, including **userId**, **workspaceId**, and **stackId**. Spans created by **TracingFilter** also have an HTTP status code tag.

Declaring tags in a traced method is done statically by setting fields from the **TracingTags** class:

```
TracingTags.WORKSPACE_ID.set(workspace.getId());
```

TracingTags is a class where all commonly used tags are declared, as respective**AnnotationAware** tag implementations.

Additional resources

For more information about how to use Jaeger UI, visit Jaeger documentation: Jaeger Getting Started Guide.

CHAPTER 5. SECURING CODEREADY WORKSPACES

This section describes all aspects of user authentication, types of authentication, and permissions models on the CodeReady Workspaces server and its workspaces.

5.1. AUTHENTICATING USERS

This document covers all aspects of user authentication in Red Hat CodeReady Workspaces, both on the CodeReady Workspaces server and in workspaces. This includes securing all REST API endpoints, WebSocket or JSON RPC connections, and some web resources.

All authentication types use the JWT open standard as a container for transferring user identity information. In addition, CodeReady Workspaces server authentication is based on the OpenID Connect protocol implementation, which is provided by default byKeycloak.

Authentication in workspaces implies the issuance of self-signed per-workspace JWT tokens and their verification on a dedicated service based on JWTProxy.

5.1.1. Authenticating to the CodeReady Workspaces server

5.1.1.1. Authenticating to the CodeReady Workspaces server using OpenID

OpenID authentication on the CodeReady Workspaces server implies the presence of an external OpenID Connect provider and has the following main steps:

- Authenticate the user through a JWT token that is retrieved from an HTTP request or, in case of a missing or invalid token, redirect the user to the Keycloak login page.
- Send authentication tokens in an Authorization header. In limited cases, when it is not possible to use the Authorization header, the token can be sent in the token query parameter. Example: OAuth authentication initialization.
- Compose an internal **subject** object that represents the current user inside the CodeReady Workspaces server code.

NOTE

The only supported and tested OpenID provider is Keycloak.

Procedure

To authenticate to the CodeReady Workspaces server using OpenID authentication:

1. Request the OpenID 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**, or **client_id** returned in the JSON format.
2. The service URL is ***<che.host>*:*<che.port>*/api/keycloak/settings**, and it is only available in the CodeReady Workspaces multi-user mode. The presence of the service in the URL confirms that the authentication is enabled in the current deployment.
   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"
}
```

The service allows downloading the JavaScript client library to interact with the provider using the **<che.host>:<che.port>/api/keycloak/OIDCKeycloak.js** URL.

3. Redirect the user to the appropriate provider's login page with all the necessary parameters, including **client_id** and the return redirection path. This can be done with any client library (JS or Java).

4. When the user is logged in to the provider, the client side-code is obtained, and the JWT token has validated the token, the creation of the **subject** begins.

The verification of the token signature occurs in two main steps:

1. Authentication: The token is extracted from the Authorization header or from the**token** query parameter and is parsed using the public key retrieved from the provider. In case of expired, invalid, or malformed tokens, a **403** error is sent to the user. The use of the query parameter should be minimised because its support may be limited or removed in future versions.
   If the validation is successful, the parsed form of the token is passed to the environment initialization step:

2. Environment initialization: The filter extracts data from the JWT token claims, creates the user in the local database if it is not yet present, and constructs the **subject** object and sets it into the per-request EnvironmentContext object, which is statically accessible everywhere.
   If the request was made using only a machine token, the following single authentication filter is used:

   org.eclipse.che.multiuser.machine.authentication.server.MachineLoginFilter: The filter finds the user that the **userId** token belongs to, retrieves the user instance, and sets the principal to the session. The CodeReady Workspaces server-to-server requests are performed using a dedicated request factory that signs every request with the current subject token obtained from the **EnvironmentContext** object.

NOTE

Providing user-specific data

Since Keycloak may store user-specific information (first and last name, phone number, job title), there is a special implementation of the ProfileDao that can provide this data to consumers. The implementation is read-only, so users cannot perform create and update operations.

5.1.1.1.1. Obtaining the token from credentials through Keycloak

Clients that cannot run JavaScript or other clients (such as command-line clients or Selenium tests) must request the authorization token directly from Keycloak.

To obtain the token, send a request to the token endpoint with the username and password credentials. This request can be schematically described as the 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
```

The CodeReady Workspaces dashboard uses a customized Keycloak login page and an authentication mechanism based on **grant_type=authorization_code**. It is a two-step authentication process:

1. Logging in and obtaining the authorization code.
2. Obtaining the token using this authorization code.

5.1.1.1.2. Obtaining the token from the OpenShift token through Keycloak

When CodeReady Workspaces has been installed on OpenShift using the Operator, and the OpenShift OAuth integration has been enabled (it is by default), then the CodeReady Workspaces authentication token of a user can also be retrieved from the user OpenShift token.

To do this, send a request to the token endpoint that can be schematically described as the following cURL request:

```
$ curl -X POST -d "client_id=<client_name>" \
--data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
-d "subject_token=<user_openshift_token>" \
 -d "subject_issuer=<openshift_identity_provider_name>" \
 --data-urlencode "subject_token_type=urn:ietf:params:oauth:token-type:access_token" \
 http://<keyckloak_host>:5050/auth/realms/<realm_name>/protocol/openid-connect/token
```

The default values for ***<openshift_identity_provider_name>*** are:

- On OpenShift 3.11: **openshift-v3**
- On OpenShift 4.x: **openshift-v4**

<user_openshift_token> is the token retrieved by the end-user with the command:

```
$ oc whoami --show-token
```

WARNING

Before using this token exchange feature for an end user, the end user should have logged in at least once interactively to the CodeReady Workspaces Dashboard through the OpenShift login page. This is required to properly link OpenShift and Keycloak user accounts, and set the required user profile information (email, first and last names).

5.1.1.2. Authenticating to the CodeReady Workspaces server using other authentication implementations

This procedure describes how to use an OIDC authentication implementation other than Keycloak.

Procedure

1. Update the authentication configuration parameters that are stored in the **multiuser.properties** file (such as client ID, authentication URL, realm name).
2. Write a single filter or a chain of filters to validate tokens, create the user in the CodeReady Workspaces dashboard, and compose the **subject** object.
3. If the new authorization provider supports the OpenID protocol, use the OIDC JS client library available at the settings endpoint because it is decoupled from specific implementations.
4. If the selected provider stores additional data about the user (first and last name, job title), it is recommended to write a provider-specific ProfileDao implementation that provides this information.

5.1.1.3. Authenticating to the CodeReady Workspaces server using OAuth

For easy user interaction with third-party services, the CodeReady Workspaces server supports OAuth authentication. OAuth tokens are also used for GitHub-related plug-ins.

OAuth authentication has two main flows: internal and external, which is based on the Keycloak brokering mechanism. The following are the two main OAuth API implementations:

- org.eclipse.che.security.oauth.EmbeddedOAuthAPI, for the internal flow.
- org.eclipse.che.multiuser.keycloak.server.oauth2.DelegatedOAuthAPI, for the external flow.

Use the **che.oauth.service_mode=*<embedded|delegated>*** configuration property to switch between the two implementations. The main REST endpoint in the OAuth API is org.eclipse.che.security.oauth.OAuthAuthenticationService, which contains:

- An authenticate method to start the OAuth authentication flow
- A callback method to process callbacks from the provider
- A token to retrieve the current user’s OAuth token

These methods refer to the currently activated embedded or delegated OAuthAPI that does the underlying operations, including finding the appropriate authenticator, initializing the login process, and user forwarding.

5.1.1.4. Using Swagger or REST clients to execute queries

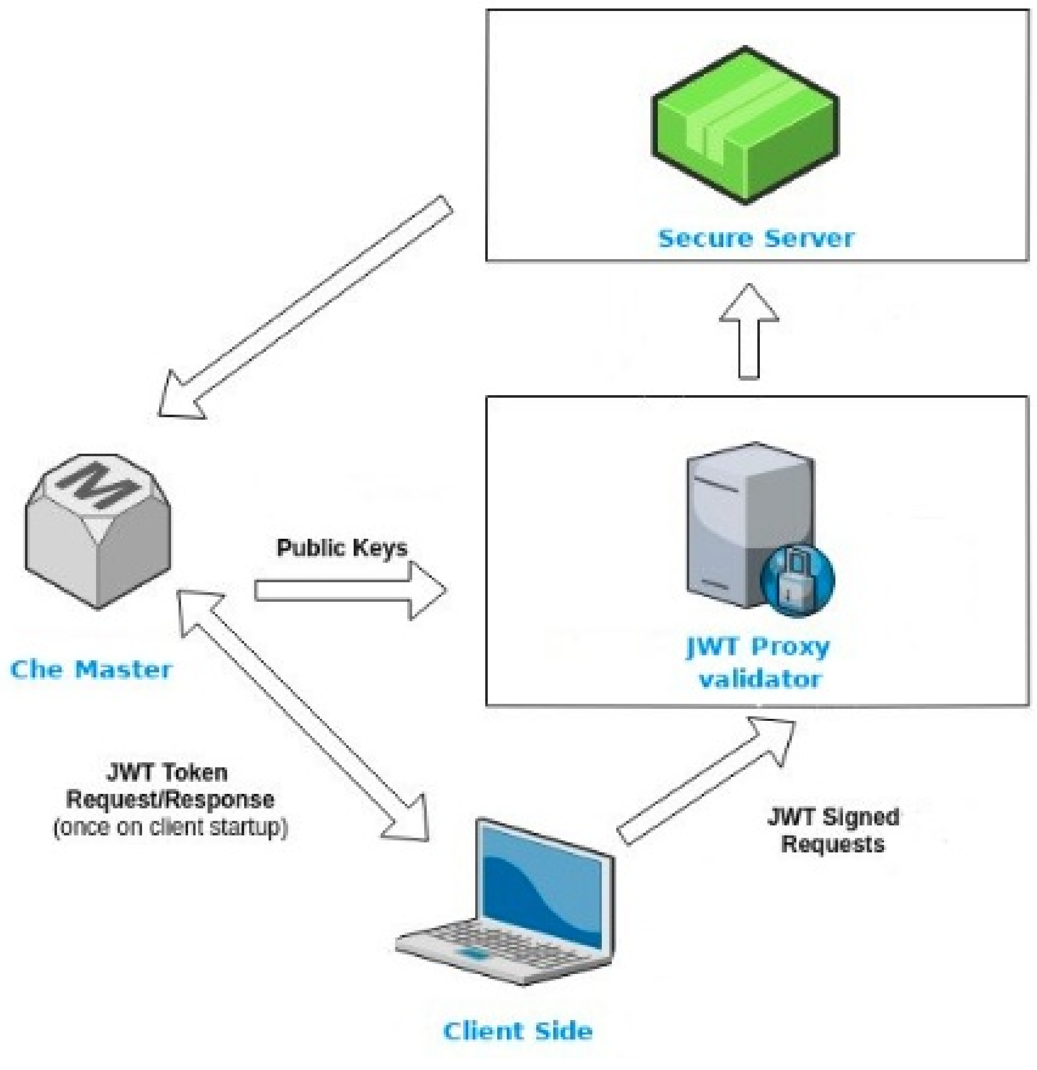
The user's Keycloak token is used to execute queries to the secured API on the user's behalf through REST clients. A valid token must be attached as the Request header or the **?token=$token** query parameter.

Access the CodeReady Workspaces Swagger interface at http://che_host:8080/swagger. The user must be signed in through Keycloak, so that the access token is included in the Request header.

5.1.2. Authenticating in a workspace

Workspace containers may contain services that must be protected with authentication. Such protected services are called secure. For this purpose, a machine authentication mechanism should be used. Machine tokens avoid the need to pass Keycloak tokens to workspace containers (which can be insecure). Also, Keycloak tokens may have a relatively shorter lifetime and require periodic renewals or refreshes, which is difficult to manage and keep in sync with the same user session tokens on clients.

Figure 5.1. Authentication inside a workspace



5.1.2.1. Creating secure servers

To create secure servers in workspaces, set the **secure** attribute of the endpoint to**true** in the **dockerimage** type component in the devfile.

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
components:
  - type: dockerimage
    image: eclipse/maven-jdk8:latest
    volumes:
      - name: maven-repo
        containerPath: /root/.m2
        env:
      - name: ENV_VAR
        value: value
        endpoints:
      - name: maven-server
        port: 3101
        attributes:
        protocol: http
        secure: 'true'
        public: 'true'
        discoverable: 'false'
        memoryLimit: 1536M
```

5.1.2.2. Workspace JWT token

Workspace tokens are JSON web tokens (JWT) that contain the following information in their claims:

- **uid**: The ID of the user who owns this token
- **uname**: The name of the user who owns this token
- **wsid**: The ID of a workspace which can be queried with this token

Every user is provided with a unique personal token for each workspace. The structure of a token and the signature are different than they are in Keycloak. The following is an example token 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": "RSASHA256(base64UrlEncode(header) + . +  base64UrlEncode(payload))"
}
```

The SHA-256 cipher with the RSA algorithm is used for signing machine tokens. It is not configurable. Also, there is no public service that distributes the public part of the key pair with which the token is signed.

5.1.2.3. Machine token validation

The validation of machine tokens is performed using a dedicated per-workspace service with **JWTProxy** running on it in a separate pod. When the workspace starts, this service receives the public part of the SHA key from the CodeReady Workspaces server. A separate verification endpoint is created for each secure server. When traffic comes to that endpoint, **JWTProxy** tries to extract the token from the cookies or headers and validates it using the public-key part.

To query the CodeReady Workspaces server, a workspace server can use the machine token provided in the **CHE_MACHINE_TOKEN** environment variable. This token is the user's who starts the workspace. The scope of such requests is restricted to the current workspace only. The list of allowed operations is also strictly limited.

5.2. AUTHORIZING USERS

User authorization in CodeReady Workspaces is based on the permissions model. Permissions are used to control the allowed actions of users and establish a security model. Every request is verified for the presence of the required permission in the current user subject after it passes authentication. You can control resources managed by CodeReady Workspaces and allow certain actions by assigning permissions to users.

Permissions can be applied to the following entities:

- Workspace
- Organization
- System

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

5.2.1. workspace permissions

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

The following permissions are associated with workspaces:

Table 5.1. workspace permissions

Permission	Description
read	Allows reading the workspace configuration.

Permission	Description
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.

5.2.2. CodeReady Workspaces organization permissions

An CodeReady Workspaces organization is a named set of users. The following permissions are applicable to organizations:

Table 5.2. CodeReady Workspaces organization permissions

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.

5.2.3. CodeReady Workspaces system permissions

CodeReady Workspaces system permissions control aspects of the whole CodeReady Workspaces installation. The following permissions are applicable to the system:

Table 5.3. CodeReady Workspaces system permission

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 accessing endpoints used for monitoring the state of the server.

All system permissions are granted to the administrative user who is configured in the **CHE_SYSTEM_ADMIN__NAME** property (the default is**admin**). The system permissions are granted when the CodeReady Workspaces server starts. If the user is not present in the CodeReady Workspaces user database, it happens after the first user's login.

5.2.4. 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 the given account.
/resource/free/{accountId}	POST	Edit free resource limit for the given account.
/resource/free/{accountId}	DELETE	Remove free resource limit for the 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 the CodeReady Workspaces server.

Path	HTTP Method	Description
/logger/{name}	GET	Get configurations of logger by its name in the CodeReady Workspaces server.
/logger/{name}	PUT	Create logger in the CodeReady Workspaces server.
/logger/{name}	POST	Edit logger in the CodeReady Workspaces server.
/resource/{accountId}/details	GET	Get detailed information about resources for the given account.
/system/stop	POST	Shutdown all system services, prepare CodeReady Workspaces to stop.

5.2.5. monitorSystem permission

Users with the monitorSystem permission have access to the following services.

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

5.2.6. Listing CodeReady Workspaces permissions

To list CodeReady Workspaces permissions that apply to a specific resource, perform the **GET /permissions** request.

To list the permissions that apply to a user, perform the **GET /permissions/{domain}** request.

To list the permissions that apply to all users, perform the **GET /permissions/{domain}/all** request. The user must have manageSystem permissions to see this information.

The suitable domain values are:

- system
- organization
- workspace

NOTE

The domain is optional. If no domain is specified, the API returns all possible permissions for all the domains.

5.2.7. Assigning CodeReady Workspaces permissions

To assign permissions to a resource, perform the **POST /permissions** request. The suitable domain values are:

- system
- organization
- workspace

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

Requesting CodeReady Workspaces user permissions

```
{
  "actions": [
    "read",
    "use",
    "run",
    "configure",
    "setPermissions"
  ],
  "userId": "userID",          (1)
  "domainId": "workspace",
  "instanceId": "workspaceID"  (2)
}
```

(1) The userId parameter is the ID of the user that has been granted certain permissions.

(2) The instanceId parameter is the ID of the resource that retrieves the permission for all users.

5.2.8. Sharing CodeReady Workspaces permissions

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

Procedure

To share workspace permissions:

1. Select a workspace in the user dashboard.
2. Navigate to the Share tab and enter the email IDs of the users. Use commas or spaces as separators for multiple emails.

CHAPTER 6. REMOVING USER DATA

To remove all user data, see the CodeReady Workspaces 2.0 Installation Guide