



# **Red Hat Developer Studio 12.9**

## **Getting Started with Container and Cloud-based Development**

Starting Development of Container and Cloud-based Applications Using Red Hat Developer Studio



# Red Hat Developer Studio 12.9 Getting Started with Container and Cloud-based Development

---

Starting Development of Container and Cloud-based Applications Using Red Hat Developer Studio

Supriya Takkhi  
sbharadw@redhat.com

## Legal Notice

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

This compilation of topics contains information on how to start developing containerized applications and applications for cloud deployment.

## Table of Contents

<b>CHAPTER 1. DEVELOPING USING CONTAINERS AND THE CLOUD</b> .....	<b>4</b>
1.1. USING RED HAT CONTAINER DEVELOPMENT KIT TOOLING IN DEVELOPER STUDIO	4
1.1.1. Installing CDK in DevStudio	4
1.1.1.1. Downloading and installing CDK in the IDE	4
1.1.2. Using the Docker tooling	8
1.1.2.1. Using Docker for Container-based development	8
1.1.2.2. Building the Docker image Using the Container Development Environment	9
1.1.2.3. Next steps for the Docker tooling	10
1.1.3. Using the OpenShift Container Platform tooling	10
1.1.3.1. Next steps for the OpenShift tooling	11
<b>CHAPTER 2. DEVELOPING FOR THE CLOUD WITH OPENSIFT 3</b> .....	<b>12</b>
2.1. CREATING AN OPENSIFT CONTAINER PLATFORM APPLICATION IN DEVELOPER STUDIO	12
2.1.1. Creating a new OpenShift Container Platform connection	12
2.1.2. Creating a new OpenShift Container Platform project	13
2.1.3. Creating a new OpenShift Container Platform application	14
2.1.4. Importing an existing OpenShift Container Platform application into the IDE	17
2.1.5. Deploying an application using the server adapter	18
2.1.6. Deleting an OpenShift Container Platform project	18
2.2. SETTING UP AND REMOTELY MONITORING AN OPENSIFT CONTAINER PLATFORM APPLICATION	19
2.2.1. Setting up OpenShift Client Binaries	19
2.2.2. Setting up Port Forwarding	19
2.2.3. Streaming Pod Logs	20
2.2.4. Streaming Build Logs	21
2.3. BUILDING AND DEPLOYING DOCKER-FORMATTED CONTAINER IMAGE TO CONTAINER DEVELOPMENT KIT OPENSIFT REGISTRY	21
2.3.1. Installing the javascript modules	22
2.3.2. Building the frontend microservice	23
2.3.2.1. Deploying the frontend microservice	23
2.3.3. Connecting the frontend and bonjour microservices	24
2.3.3.1. Deploying the bonjour microservice	24
2.3.3.2. Scaling the pod	25
2.3.4. Editing the bonjour microservice	26
2.3.4.1. Viewing the edited bonjour microservice on the frontend microservice	26
2.4. DEPLOYING THE OPENSIFT CONTAINER PLATFORM 3 RESOURCE	27
2.4.1. Deploying the s2i-spring-boot-cfx-jaxrs template	27
2.4.2. Viewing the s2i-spring-boot-cfx-jaxrs application in the Web Console	28
2.4.3. Defining services and routes using a JSON file	29
<b>CHAPTER 3. DEVELOPING WITH DOCKER</b> .....	<b>32</b>
3.1. USING DOCKER TOOLING IN DEVELOPER STUDIO	32
3.1.1. Connecting to Docker Daemon	32
Installing Docker	32
Setting up an account in the Docker tooling	32
3.1.1.1. Testing an existing Docker connection	34
3.1.1.2. Editing a Docker connection	35
3.1.2. Managing Docker images	36
3.1.2.1. Pulling the jboss/wildfly:11.0.0.Final image	36
3.1.2.2. Pushing images	37
3.1.2.3. Running Image Launch Configuration	38
3.1.2.4. Building images with Dockerfile	40

3.1.2.5. Working with image tags	41
3.1.2.5.1. Adding tags to images	41
3.1.2.5.2. Removing tags for images	41
3.1.3. Managing Docker Containers	42
3.1.4. Working with docker-compose.yml files	42
3.1.4.1. Creating the docker-compose project	42
3.1.4.2. Creating the required files in the docker-compose project	43
3.1.5. Troubleshooting	45



# CHAPTER 1. DEVELOPING USING CONTAINERS AND THE CLOUD

## 1.1. USING RED HAT CONTAINER DEVELOPMENT KIT TOOLING IN DEVELOPER STUDIO

Red Hat Container Development Kit (CDK) is a pre-built container development environment based on Red Hat Enterprise Linux (RHEL). CDK helps you get started with developing container-based applications quickly. You can easily set up CDK and then use toolings, such as, OpenShift Container Platform and Docker, through Developer Studio (DevStudio), without spending additional time in setting up and configuring the supplementary tooling.

After it is installed, you can use the installed components with the Docker tooling.

### 1.1.1. Installing CDK in DevStudio

You can download and install CDK from within DevStudio. This option requires some additional configuration steps before the two products can be used together.

#### Prerequisites

- Ensure that hardware virtualization is enabled on your system.
- Ensure that the following are installed on your system:
  - Hypervisor such as VirtualBox, Linux KVM/libvirt, xhyve (macOS) or hyper-V (Windows) is installed and configured
  - DevStudio 12.0
- Ensure that you have a Red Hat Developer account. For a new account, visit <https://developers.redhat.com/>.

For details about installing these prerequisites, see the [Red Hat Container Development Kit Installation Guide](#).

#### 1.1.1.1. Downloading and installing CDK in the IDE

##### Procedure

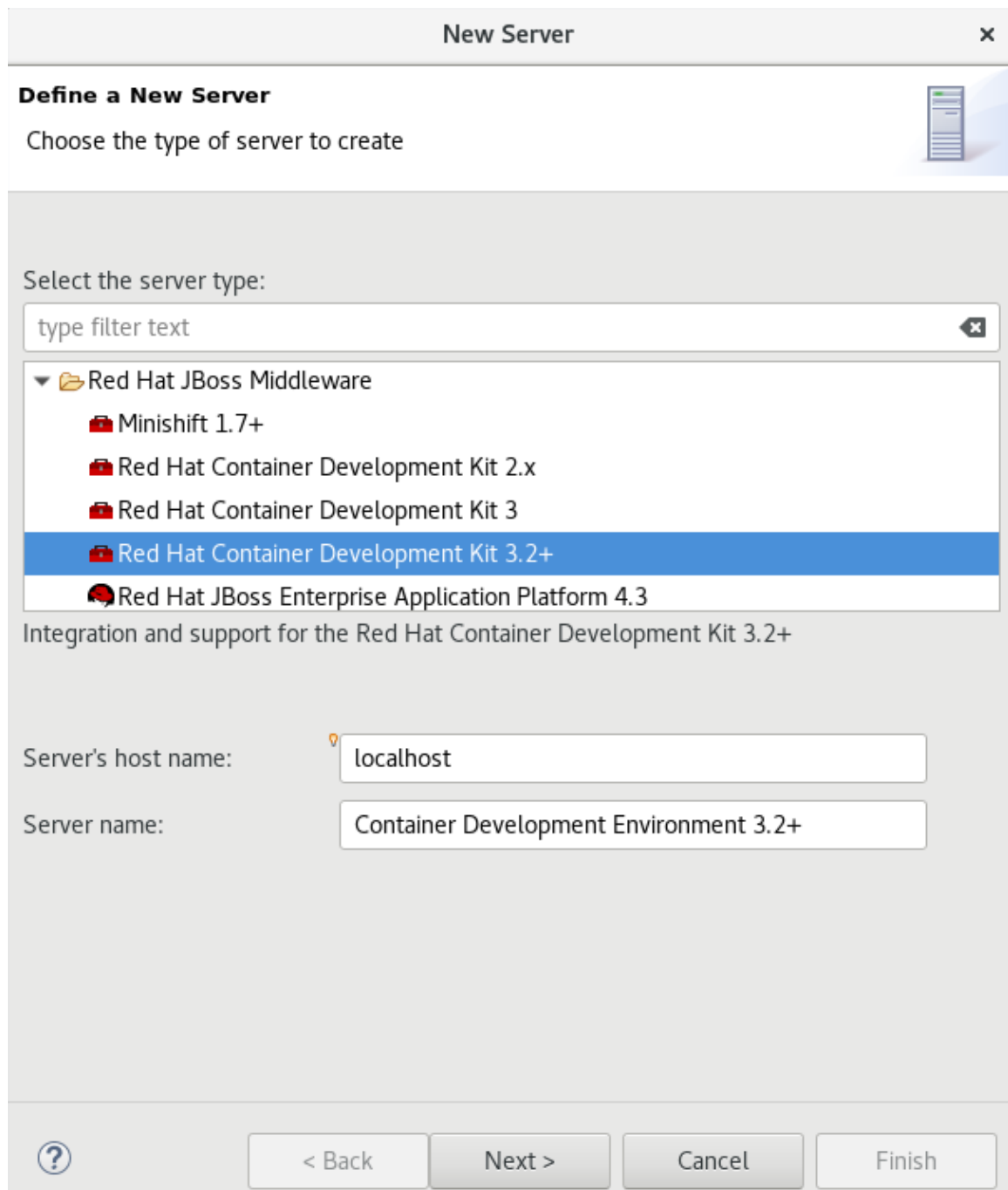
To download and install CDK in the IDE, take the following steps:

1. Start the IDE.
2. To open the **Servers** view, click **Windows > Show View > Servers**.
3. Click the **No servers are available. Click this link to create a new server** link (or right-click an existing server and click **New > Server**.)
4. In the **New Server** window:
  - a. Expand **Red Hat JBoss Middleware** and click **Red Hat Container Development Kit 3.2+**.
  - b. Let the **Server's host name** field be as is because it is not applicable to CDK.



- c. In the **Server Name** field, if desired, type a different server name (**Container Development Environment 3.2+** is the default server name).
- d. Click **Next**.

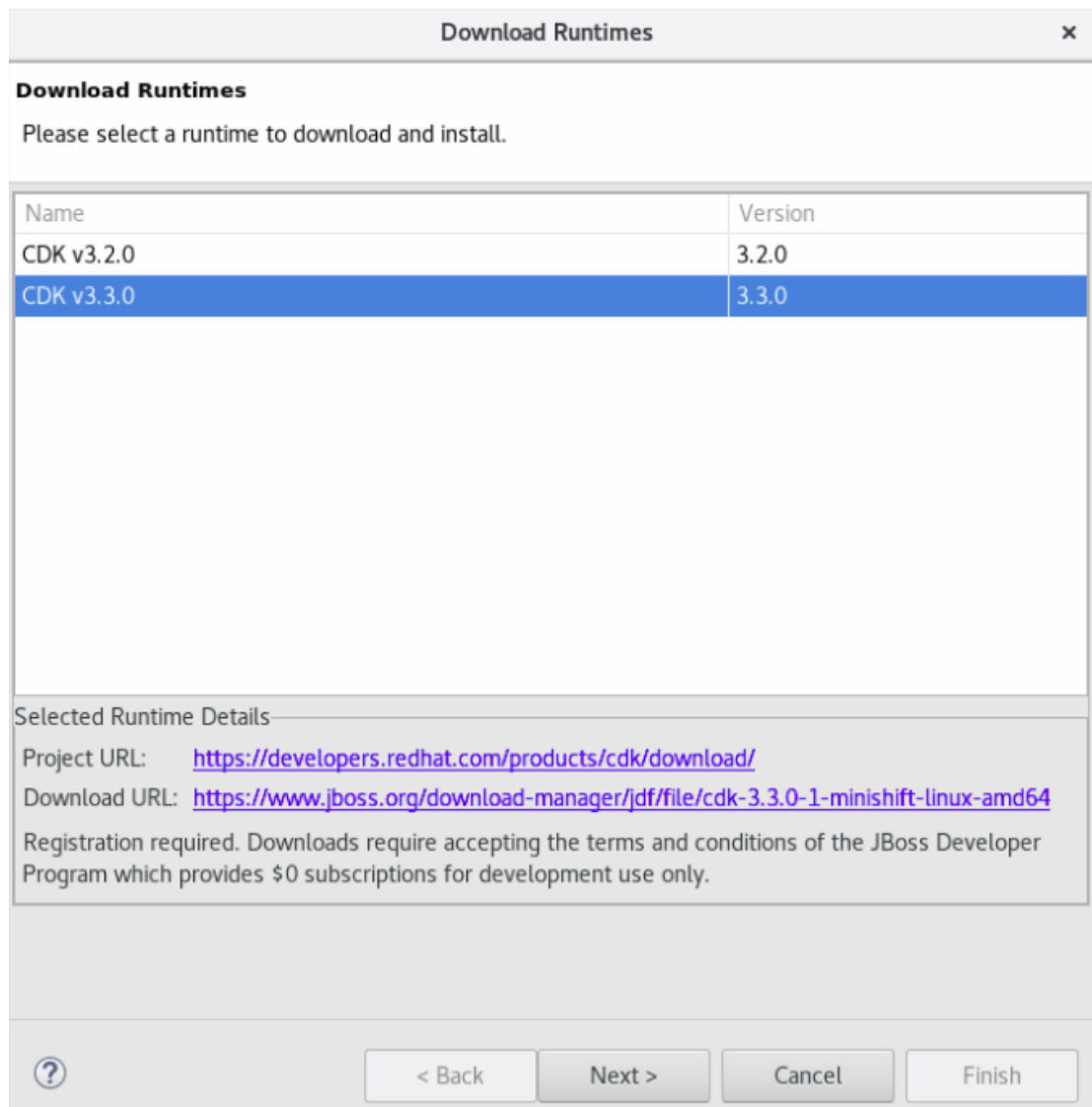
**Figure 1.1. Selecting Red Hat Container Development Kit 3.2+**



5. In the **New Server - Red Hat Container Development Environment** window:
  - a. In the **Username** list, click your username. If you do not have the credentials set up, add the credentials for the `access.redhat.com` domain. If you do not have your credentials, sign up on `developers.redhat.com`.
  - b. In the **Hypervisor** field, ensure that the relevant virtualization system for your operating system appears.
  - c. In the **Minishift Binary** field, click **Browse** to locate the minishift binary. If you do not have the CDK binary yet, follow the steps to download and install the CDK runtime:

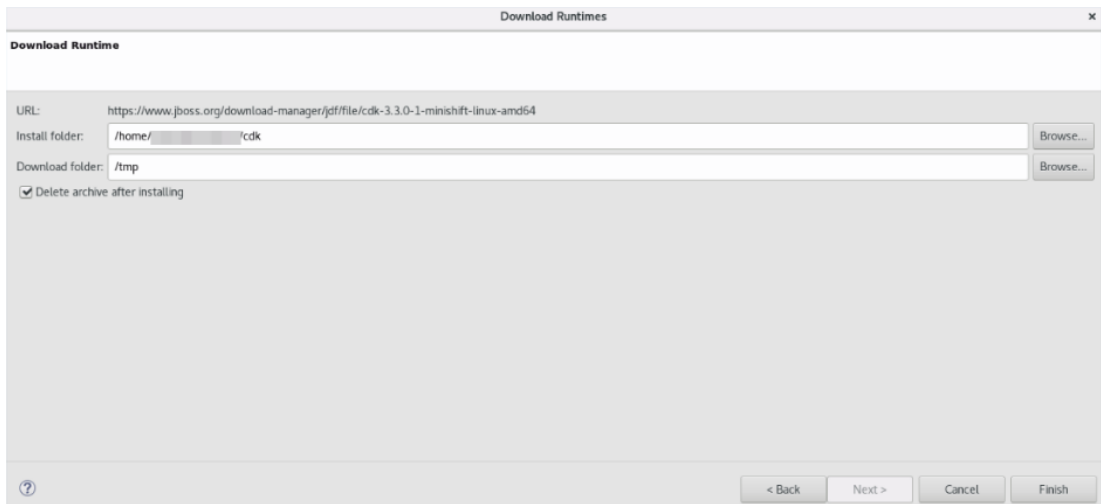
- i. Click the **Download and install runtime** link to open the **Download Runtimes** window.
- ii. From the available Runtimes, click the CDK version that you want to download and click **Next**.

**Figure 1.2. Selecting the CDK Version for Download**



- iii. In the **JBoss.org Credentials** window, in the **Username** list, click your username and in the **Password** field, enter your password. Click **Next**.
- iv. In the next window, click **I accept the terms of the license agreement** and click **Next**.
- v. In the **Download Runtime** window, ensure that the **Install folder** and the **Download folder** fields, show the desired location (or, if desired, change the locations).
- vi. Click **Finish**.

Figure 1.3. Downloading the CDK Runtime



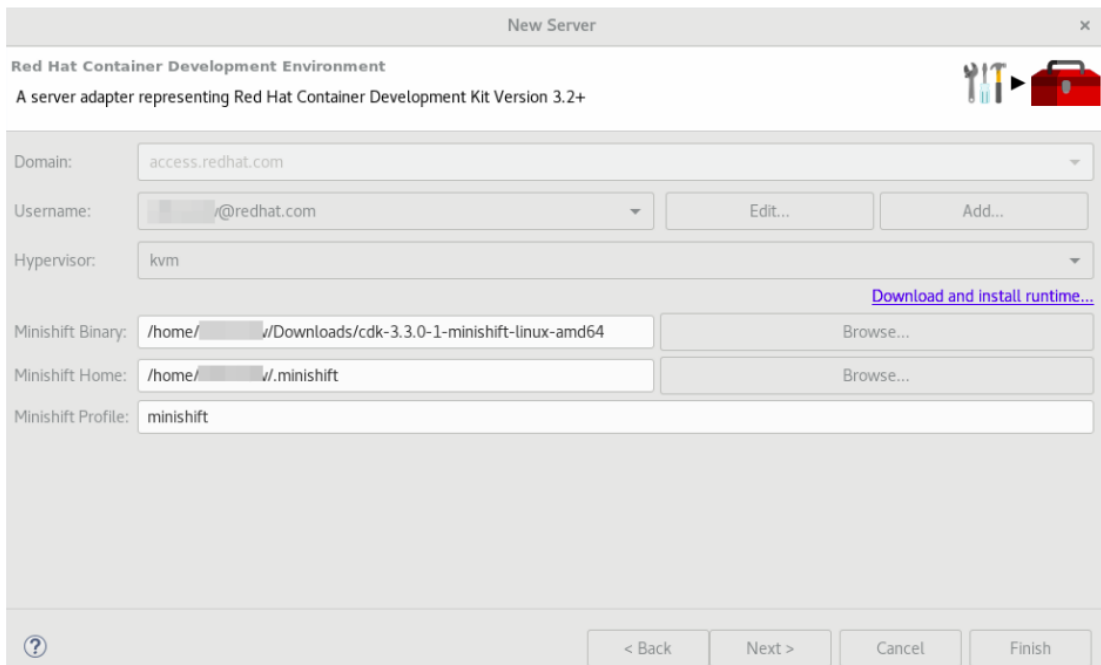
The **New Server** window appears showing the progress of the download. The download takes a few minutes to complete.



### IMPORTANT

Do not close the **New Server** window because if you do so the runtime will be downloaded but will not be configured. Once the download is complete, the **Minishift Binary** field shows the path to the downloaded binary.

Figure 1.4. The Minishift Binary Field showing the Path to the Downloaded Binary



- d. The **Minishift Home** field contains path to the folder with the CDK configuration files; default is `.minishift` in the user's home directory; you can change this to the desired location.
- e. The **Minishift Profile** field is set to `minishift` by default; you can change this to the desired value.
- f. Click **Finish**.

- The **Servers** view will include a new server adapter: **Container Development Environment 3.2+**. Right-click it and click **Start**. The **Console** view shows the progress of starting CDK.



## NOTE

In case you did not setup CDK prior to starting the server adapter, you will see a warning dialog: **Warning: CDK has not been properly initialized!**. Confirm the dialog and continue with starting CDK. Note that the Setup CDK command is called in the background. This command will set up files in your Minishift Home directory. If there is any content in this directory you may need to confirm overwriting it. Check the directory content before confirming to avoid losing important data.

**Figure 1.5. CDK has not been properly initialized Message**



- When the server adapter is starting, you may be asked to enter your credentials for developers.redhat.com (associated with access.redhat.com). If so, enter the credentials and continue.
- At the end of starting CDK, if it appears, in the **Untrusted SSL Certificate** dialog box, click **Yes**. The **OpenShift Explorer** view opens showing the IP address and the port of the OpenShift Container Platform that you have connected to: **developer {connection\_IP}** (example, **developer https://10.1.2.2:8443**). Expand the connection to see the sample projects.
- You can also open the **Docker Explorer** view to view the Container Development Environment 3.2+ connection and expand the connection to see the Containers and Images.

## 1.1.2. Using the Docker tooling

After starting the CDK server in the IDE, you can follow one of the two container development workflows to use the Docker tooling.

### 1.1.2.1. Using Docker for Container-based development

#### Procedure

Use Docker for Container-based Development as follows:

- Create a new project with your Dockerfile.
  - Click **File > New > Project**.
  - Type *java* in the search field and from the results, select **Java Project** and click **Next** to continue.
  - In the **Project name** field, type a name for the new project and click **Finish**. The **Project Explorer** view shows the project that you just created.

- d. Click **File > New > File**.
- e. In the **New File** window:
  - i. In the **Enter or select the parent folder** field, click the project that you created.
  - ii. In the **File name** field, type *Dockerfile* and click **Finish**.
- f. Edit the Dockerfile as desired and then save it. For example, copy and paste the following content in the dockerfile and then save the file:

```

    # Use latest jboss/base-jdk:8 image as the base
FROM jboss/base-jdk:8

# Set the WILDFLY_VERSION env variable
ENV WILDFLY_VERSION 10.1.0.Final
ENV WILDFLY_SHA1 9ee3c0255e2e6007d502223916cefad2a1a5e333
ENV JBOSS_HOME /opt/jboss/wildfly

USER root

# Add the WildFly distribution to /opt, and make wildfly the
owner of the extracted tar content
# Make sure the distribution is available from a well-known place
RUN cd $HOME \
    && curl -O
https://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-
$WILDFLY_VERSION.tar.gz \
    && sha1sum wildfly-$WILDFLY_VERSION.tar.gz | grep
$WILDFLY_SHA1 \
    && tar xf wildfly-$WILDFLY_VERSION.tar.gz \
    && mv $HOME/wildfly-$WILDFLY_VERSION $JBOSS_HOME \
    && rm wildfly-$WILDFLY_VERSION.tar.gz \
    && chown -R jboss:0 ${JBOSS_HOME} \
    && chmod -R g+rw ${JBOSS_HOME}

# Ensure signals are forwarded to the JVM process correctly
for graceful shutdown
ENV LAUNCH_JBOSS_IN_BACKGROUND true

USER jboss

# Expose the ports we're interested in
EXPOSE 8080

# Set the default command to run on boot
# This will boot WildFly in the standalone mode and bind to
all interface
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0"]
).
```

For additional information about the Dockerfile, see <https://docs.docker.com/engine/reference/builder>.

### 1.1.2.2. Building the Docker image Using the Container Development Environment

## Procedure

To do a Docker image build using the Container Development Environment, take the following steps:

1. In the **Project Explorer** view, expand the project and right-click the Dockerfile and select **Run As > Docker Image Build**.
2. In the **Docker Image Build Configuration** dialog box:
  - a. In the **Connection** field, select your Container Development Environment server adapter.
  - b. In the **Image Name** field, enter the desired name for the docker image and click **OK**. After the build is done, a new image with the given name is listed in the **Docker Explorer** view under CDK Docker connection under images and in the **Docker Images** view. Also, the **Console** view shows **Successfully built <Docker\_image\_ID>** message.
3. Run a Docker image using the Container Development Environment:
  - a. Open the **Docker Explorer** view by typing Ctrl+3 in the quick access menu or using the **Window > Perspective > Open Perspective > Docker Tooling** menu option.
  - b. Navigate to the **Images** node under the Docker connection.
  - c. Right-click your image and click **Run**.
  - d. In the **Run a Docker Image** window, fill in the necessary details and click **Finish** to run your image. The **Console** view shows the progress of execution of the Docker image. Optionally, give the container a name. This name helps locate the specific container in a list of containers in the future.
  - e. In the **Docker Explorer** view, select the container that you named in the preceding step and expand its node and select the 8080 port and click **Show In > Web Browser** to access the application deployed in the Docker container. The application opens in the default web browser.

### 1.1.2.3. Next steps for the Docker tooling

For further information about the basics of Docker Tooling, see [Using Docker Tooling in Developer Studio](#).

### 1.1.3. Using the OpenShift Container Platform tooling

#### Procedure

Use OpenShift Container Platform for Container-based development as follows:

1. Create a new OpenShift Container Platform project. These projects are like namespaces for OpenShift applications. They are different from how Eclipse projects relate to Eclipse applications. Additionally, Eclipse projects can be mapped to OpenShift applications.
  - a. In the **OpenShift Explorer** view, right-click the connection and click **New > Project** to create a new OpenShift Container Platform project.



#### NOTE

The CDK server adapter creates the OpenShift Container Platform connection when you start the CDK server adapter in the preceding sections.

- b. Add the name and other relevant details for the new project and click **Finish**.
2. Create an application in your OpenShift Container Platform project using the templates:
    - a. Right-click your new project name and click **New > Application**.
    - b. In the **New OpenShift Application** window, search box, type the application type required. For example, for a **node.js** application, type *nodejs* and from the displayed list, select the relevant nodejs template and click **Finish**.
    - c. Click **OK** to accept the results of the application creation process.
    - d. In the **Import OpenShift Application** window, select a **Git Clone Location** and click **Finish**.

For additional tasks that you can do with the OpenShift Container Platform projects and application, refer to [Creating an OpenShift Container Platform Application in Developer Studio](#).

### 1.1.3.1. Next steps for the OpenShift tooling

For additional tasks to be performed using the OpenShift Container Platform tooling, see [Developing for the Cloud with OpenShift 3](#).

## CHAPTER 2. DEVELOPING FOR THE CLOUD WITH OPENSIFT 3

### 2.1. CREATING AN OPENSIFT CONTAINER PLATFORM APPLICATION IN DEVELOPER STUDIO

Using the OpenShift Container Platform tooling you can create, import, and modify OpenShift Container Platform applications.

#### 2.1.1. Creating a new OpenShift Container Platform connection

You must create an OpenShift connection in the **OpenShift Explorer** view in Developer Studio to use the OpenShift tooling in the IDE. An OpenShift connection connects your IDE to an OpenShift instance (based on CDK, OpenShift Online, Kubernetes, minishift). The connection is listed in the **OpenShift Explorer** view of the IDE and is in the format: `username@example.com {OpenShift_console_URL}`. You can have more than one OpenShift connection configured in the IDE.

#### Procedure

1. In the IDE, click **Window** → **Show View** → **Other**
2. Search for **OpenShift Explorer**, select it, and click **OK**
3. In the **OpenShift Explorer** view, click **New Connection Wizard**.
4. In the **Connection** list, click **<New Connection>**.
5. In the **Server type** list, click **OpenShift 3**.
6. In the **Server** field, type the URL for an OpenShift Container Platform server.
7. In the **Authentication** section, in the **Protocol** list, click **OAuth** to authenticate using the token or click **Basic** to authenticate using login credentials.
8. Click **Finish**.




Figure 2.1. Set up a new OpenShift Container Platform connection

New OpenShift Connection ✕

**Sign in to OpenShift**

Please sign in to your OpenShift server.



**OPENSIFT**

New to OpenShift OpenShift 3? Explore the [getting started documentation](#).

Connection: <New Connection> ▼

Server type: OpenShift 3 ▼

Use default server

Server: https://openshift3serverURL.com ▼

**Authentication**

Protocol: OAuth ▼

Enter a token or [retrieve](#) a new one.

Token \* ZEp9Cy-TlCm2S3BkGi9QD45Y-RX3wWMTjW38EmxzH8Q

Save token (could trigger secure storage login)

Advanced >>

?

Cancel
Finish

### 2.1.2. Creating a new OpenShift Container Platform project

You must create a project, which essentially is a namespace with additional annotations, to centrally manage the access to resources for regular users.

#### Prerequisites

- An OpenShift Container Platform connection exists.

## Procedure

1. In the **OpenShift Explorer** view, right-click the connection and click **New > Project**. The **Create OpenShift Project** window appears.
2. In the **Project Name** field, type a name for the project. Project names must be alphanumeric and can contain the character “-” but must not begin or end with this character.
3. In the **Display Name** field, type a display name for the project. This name is used as the display name for your project in the **OpenShift Explorer** view and on the OpenShift Container Platform web console after the project is created.
4. In the **Description** field, type a description of the project.
5. Click **Finish**. The project is listed in the **OpenShift Explorer** view, under the relevant connection.

### 2.1.3. Creating a new OpenShift Container Platform application

Use the **New OpenShift Application** wizard in the IDE to create OpenShift Container Platform applications from default or custom templates. Using a template to create an application is helpful because you can use the same template to create multiple similar applications with different or identical configurations for each of them.

#### Prerequisites

- An OpenShift Container Platform project exists.

#### Procedure

1. In the **OpenShift Explorer** view of the IDE, right-click the connection and click **New** → **Application**.
2. If required, in the **New OpenShift Application** wizard, sign in to your OpenShift Container Platform server using the **Basic** protocol (username and password) or the **OAuth** protocol (token) and click **Next**.
3. In the **Select Template** window, click the **Server application source** tab.

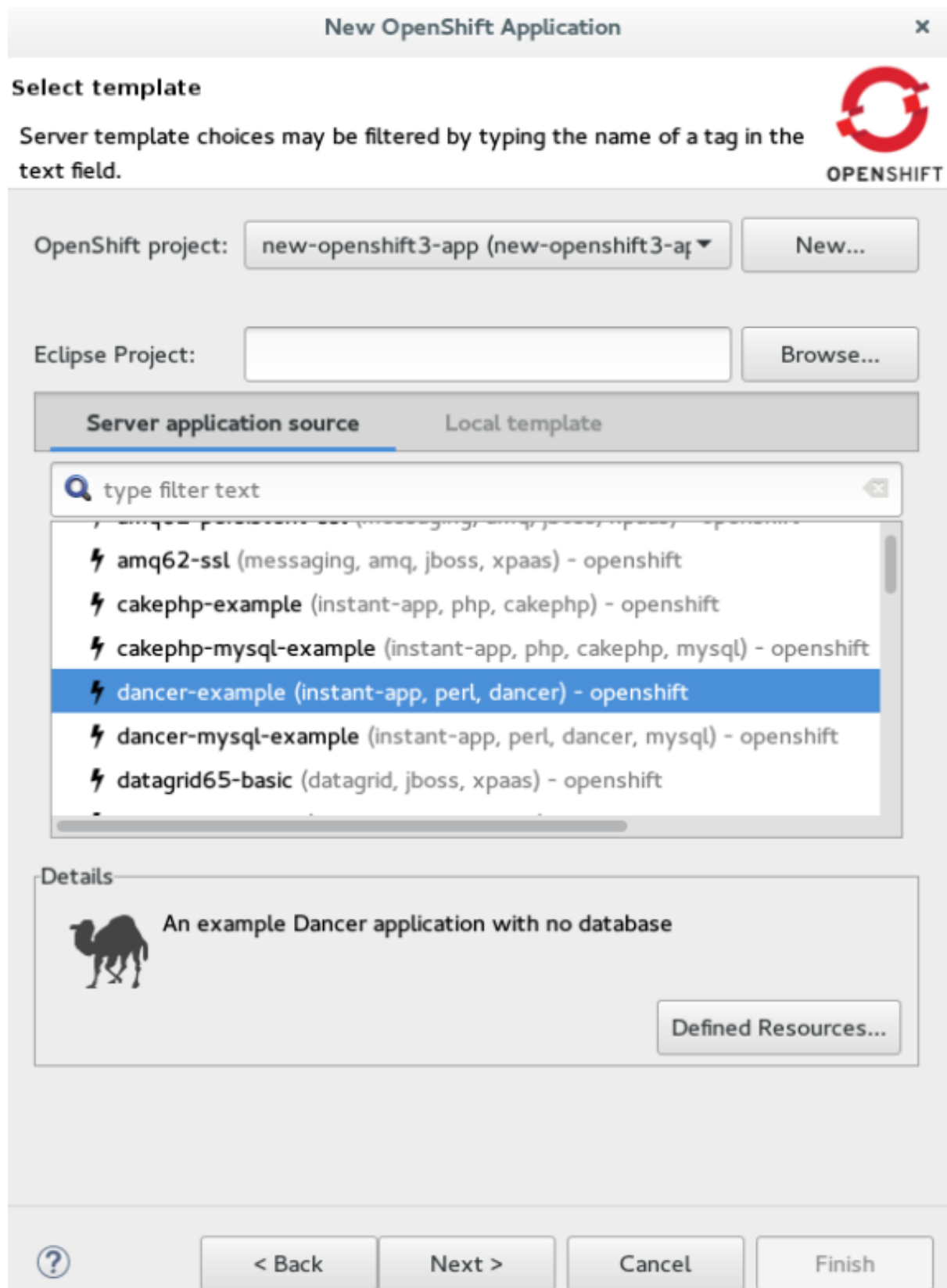


#### NOTE

To create an application from a local template, click the **Local template** tab and then click **Browse File System** or **Browse Workspace** to locate the template that you want to base the project on.

4. From the list, click the template that you want to base your project on. You can also use the **type filter text** field to search for specific templates.
5. Click **Next**.

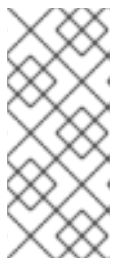
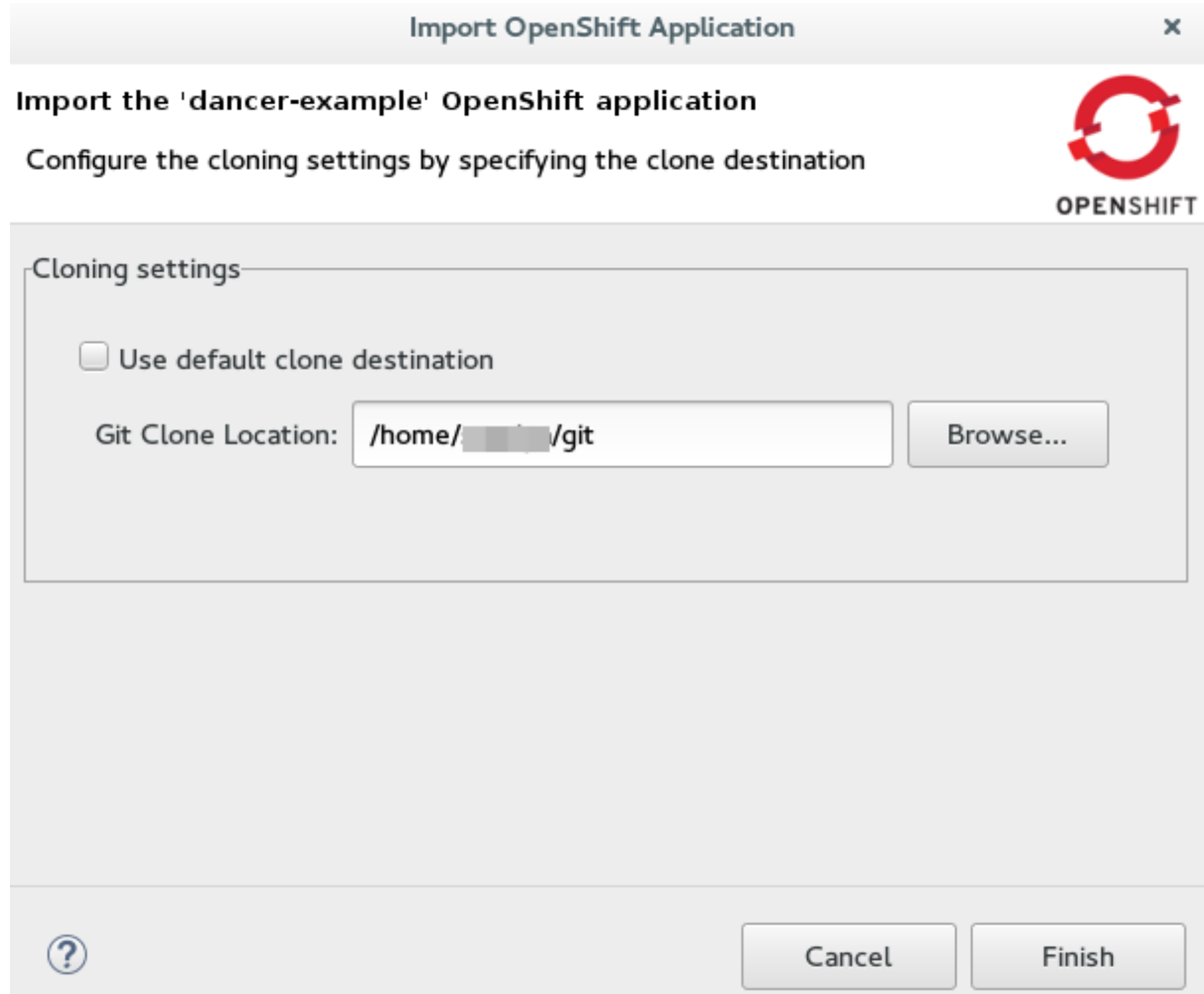
Figure 2.2. Select a template for project creation



6. In the **Template Parameters** window, confirm the parameter values and click **Next**.
7. In the **Resource Labels** window, confirm the labels that you want to add to each resource. You can also click **Add** or **Edit** to add labels or edit the existing ones.
8. Click **Finish**.

- In the **Results of creating the resources from the `{template_name}`** window, review the details and click **OK**.
- In the **Import Application** window, click **Use default clone destination** to clone the application at the default location or in the **Git Clone Location** field, type or browse for the location where you want to clone the application, and click **Finish**.

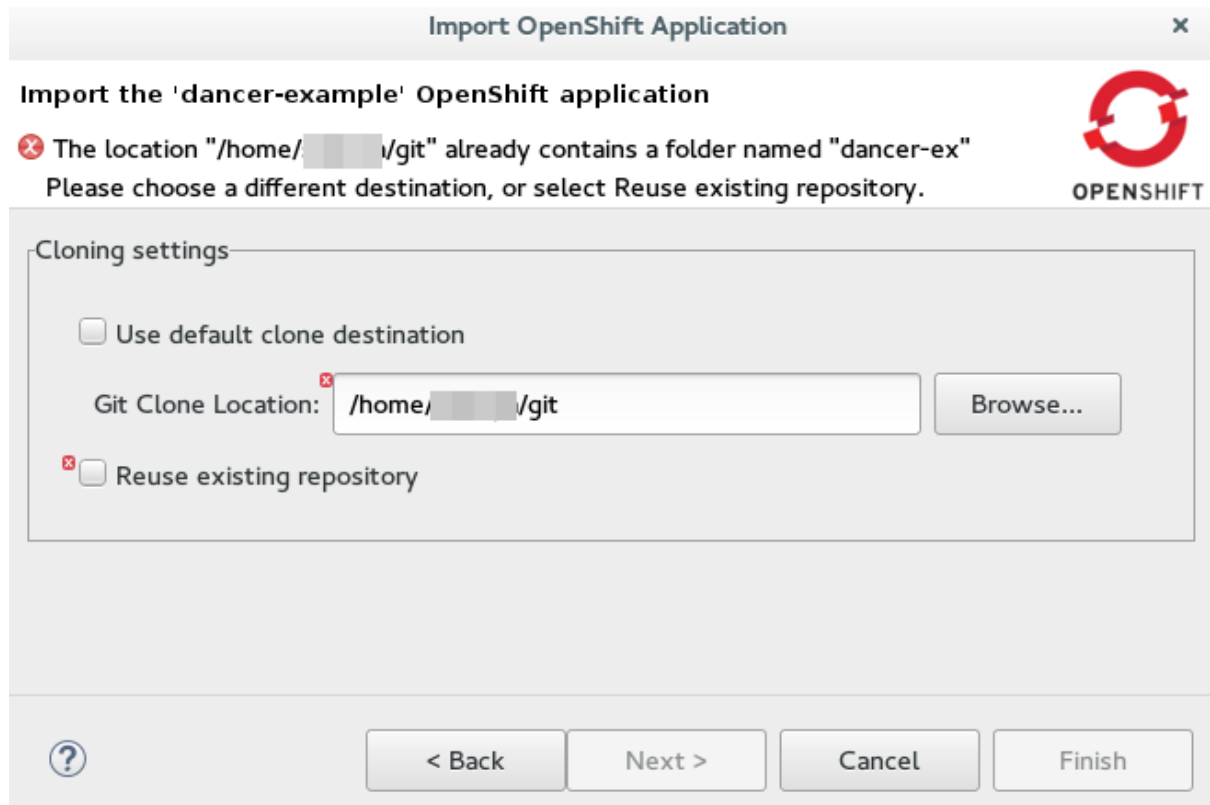
**Figure 2.3. Selecting a Git clone location**



#### NOTE

If the Git location chosen to clone the application already contains a folder with the application name that you are trying to import, you must select a new location for the Git clone. If you do not select a new location, the existing repository will be reused with the changes you made being retained but not reflected on the OpenShift Container Platform console.

Figure 2.4. Git clone location reuse



**Result:** The application appears in the **Project Explorer** view.

### Additional Resources

- To learn more about using and creating templates with OpenShift Container Platform, see [Templates](#).

### 2.1.4. Importing an existing OpenShift Container Platform application into the IDE

The **OpenShift Explorer** view in the IDE lists applications associated with your OpenShift Container Platform accounts. You can import the source code for these applications individually into the IDE using the **Import OpenShift Application** wizard. After the application is imported, you can easily modify the application source code, as required, build the application, and view it in a web browser.

### Prerequisites

- The application that you are importing in the IDE has its source specified in the **build config** file.

### Procedure

If required, sign in to your OpenShift Container Platform server using the **Basic** protocol or the **OAuth** protocol. In the **OpenShift Explorer** view of the IDE, expand the connection to locate the application to import. Depending on the type of application you want to import, take the following steps: If a project has a single application under it and you want to import this application, right-click the *{project name}* and click **Import Application**. If a project has multiple applications under it and you want to import a specific application, right-click the service and then click **Import Application**. In the **Import OpenShift Application** wizard, **Existing Build Configs** list, click the application that you want to import and click **Next**. Ensure the location in the **Git Clone Destination** field corresponds to where you want to make a local copy of the application Git repository. Click **Finish**. The application is listed in the **Project Explorer** view.

### 2.1.5. Deploying an application using the server adapter

The server adapter enables you to publish the changes that you made in your workspace project to the running OpenShift application on the OpenShift instance. It enables incremental deployment of applications directly into the deployed pods on OpenShift. You can use the server adapter to push changes in your application directly to the running OpenShift application without committing the source code to the Git repository.

#### Procedure

1. In the **OpenShift Explorer** view of the IDE, expand the connection, the project, and then the application.
2. Right-click the *{application name}* and click **Server Adapter**.
3. In the **Server Settings** window **Resources** section, select the service.



#### NOTE

The OpenShift service has a build configuration with a Git URL matching the Git remote URL of one of the workspace projects. A workspace project is selected automatically.

4. Click **Finish**. The **Servers** view is the view in focus with the server showing **[Started, Publishing...]** followed by the **Console** view showing the progress of application publishing.
5. To view the application, in the **OpenShift Explorer** view of the IDE, right-click the application, and click **Show In** → **Web browser**. The application displays in the built-in web browser.

### 2.1.6. Deleting an OpenShift Container Platform project

You may choose to delete a project from the workspace to make a fresh start in project development or after you have concluded development in a project. When you delete a project, all the resources associated with the project are deleted.

#### Prerequisites

- An OpenShift Container Platform project exists.

#### Procedure

1. In the **OpenShift Explorer** view of the IDE, expand the connection to locate the project you want to delete.
2. Right-click the *{project name}* and click **Delete Project**.
3. In the **OpenShift resource deletion** window, click **OK**.



#### NOTE

To delete more than one project (and the containing applications), in the **OpenShift Explorer** view, click the project to select it and while holding the Control key select another project that you want to delete and then press **Delete**.

## 2.2. SETTING UP AND REMOTELY MONITORING AN OPENSIFT CONTAINER PLATFORM APPLICATION

In some scenarios, the user already has a remote instance of OpenShift Container Platform running with various applications on it and may want to monitor it. The IDE allows users to set up a connection to a remote instance of OpenShift Container Platform and then use logs (application logs and build logs) to troubleshoot and monitor running applications.

### 2.2.1. Setting up OpenShift Client Binaries

#### Prerequisites

Before setting up port forwarding or streaming application and build logs, it is mandatory to set up OpenShift Client Binaries.

#### Procedure

To set up the OpenShift Client Binaries, take the following steps:

1. In the IDE, navigate to **Window** → **Preferences** → **JBoss Tools** → **OpenShift 3**.
2. Click the **here** link.
3. In the **Download from GitHub** section, click the **Release page** link.
4. Scroll to the **Assets** section for the relevant version of OpenShift Origin. Click the appropriate link to begin the client tools download for the binary for your operating system.
5. After the download is complete, extract the contents of the file.
6. Navigate to **Window** → **Preferences** → **JBoss Tools** → **OpenShift 3**
7. Click **Browse** and select the location of the OpenShift Client executable file.
8. Click **Apply** and **Close**. OpenShift Client Binaries are now set up for your IDE.

### 2.2.2. Setting up Port Forwarding

Using the **Application Port Forwarding** window, you can connect the local ports to their remote counterparts to access data or debug the application. Port forwarding automatically stops due to any one of the following reasons:

- The OpenShift Container Platform connection terminates
- The IDE shuts down
- The workspace is changed

Port forwarding must be enabled each time to connect to OpenShift Container Platform from the IDE.

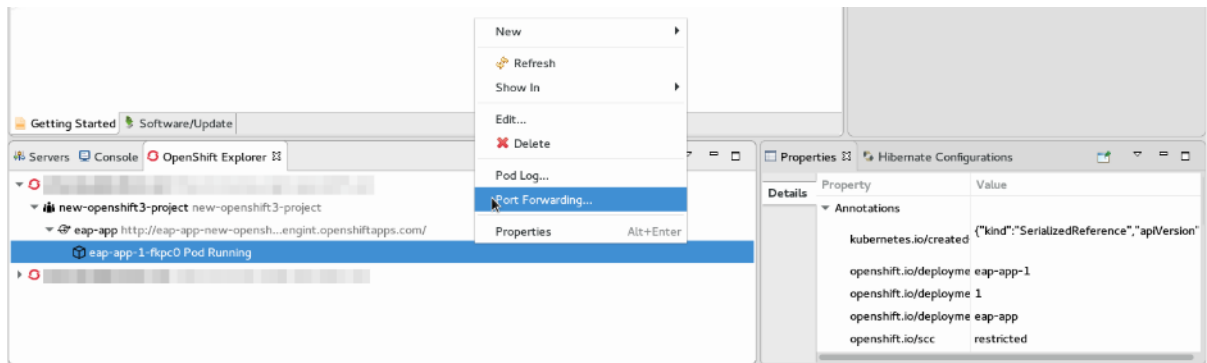
#### Procedure

To set up port forwarding, take the following steps:

1. In the **OpenShift Explorer** view, expand the connection, the project, the services, and then the Pods.

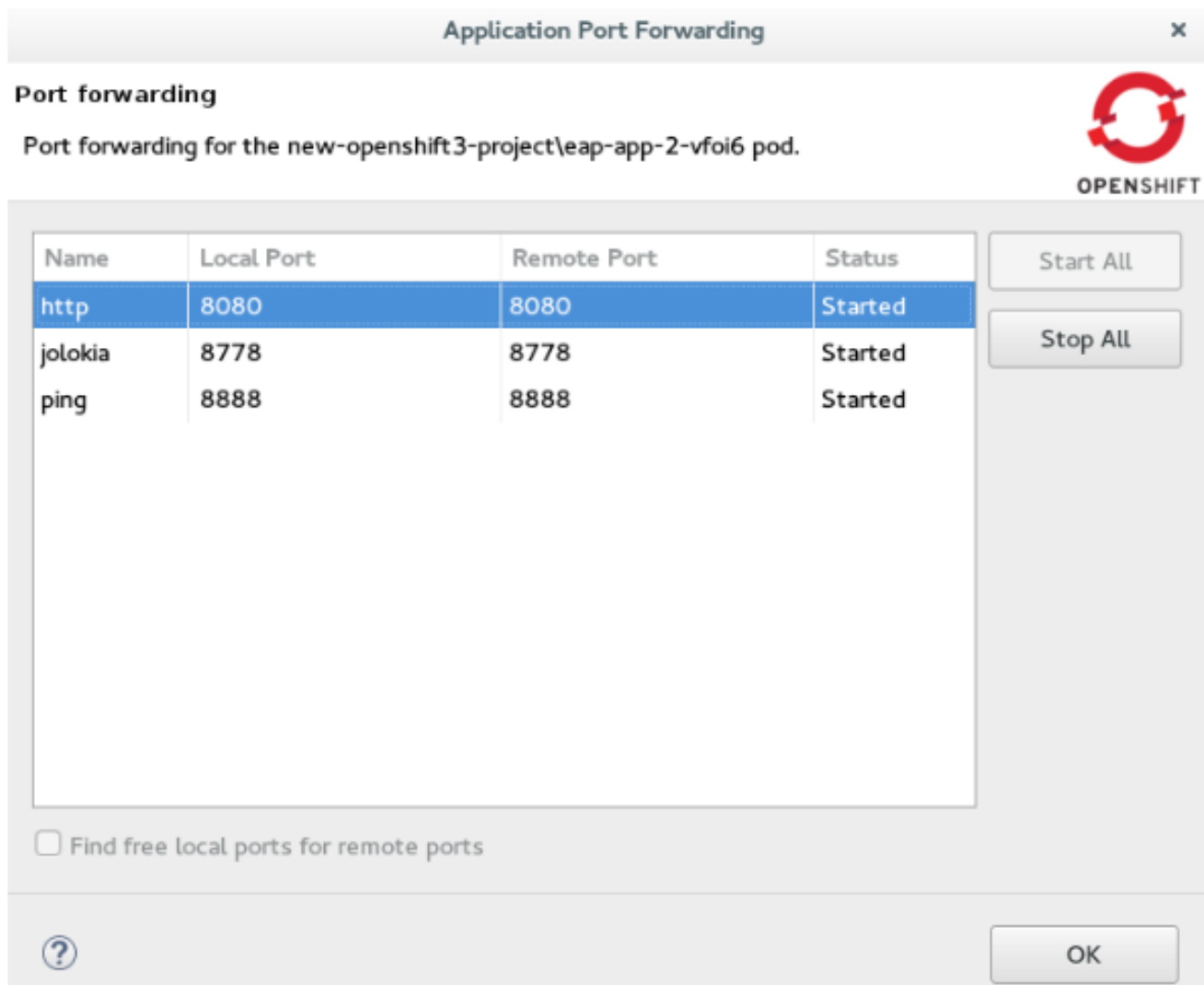
- Right-click the relevant pod and then click **Port Forwarding**.

**Figure 2.5. Setting up Port Forwarding**



- In the **Application Port Forwarding** window, click the **Find free local ports for remote ports** check box and then click **Start All**. The **Status** column shows **Started**, indicating that port forwarding is now active. Additionally, the **Console** view shows the status of port forwarding for the particular service.

**Figure 2.6. Start Port Forwarding**



### 2.2.3. Streaming Pod Logs



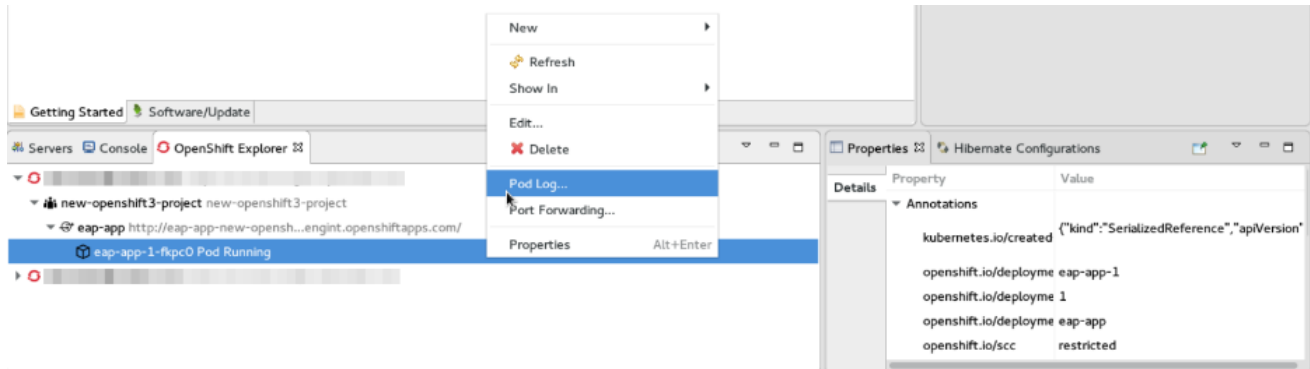
Pod logs are general logs for an application running on a remote OpenShift Container Platform instance. The streaming application logs feature in the IDE is used to monitor applications and use the previous pod log to troubleshoot if the application fails or returns errors.

## Procedure

To stream the application logs, take the following steps:

1. In the **OpenShift Explorer** view, expand the project, the services, and then the Pods.
2. Right-click the relevant Pod and then click **Pod Log**.

**Figure 2.7. Streaming Pod Log**



The **Console** view displays the Pod log.

## 2.2.4. Streaming Build Logs

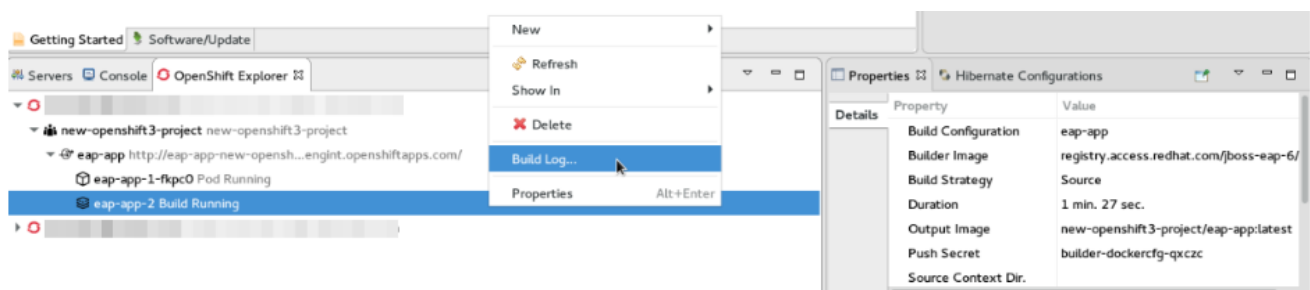
Build logs are logs that document changes to applications running on a remote OpenShift Container Platform instance. The streaming build logs feature in the IDE is used to view the progress of the application build process and to debug the application.

## Procedure

To stream build logs, take the following steps:

1. In the **OpenShift Explorer** view, expand the project, the services, and then the build.
2. Right-click the relevant build instance and click **Build Log**.

**Figure 2.8. Streaming Build Log**



The **Console** view is now the view in focus showing the build log.

## 2.3. BUILDING AND DEPLOYING DOCKER-FORMATTED CONTAINER IMAGE TO CONTAINER DEVELOPMENT KIT OPENSHIFT REGISTRY

In this article we deploy the Docker based microservices, **frontend** and **bonjour**, into an OpenShift Container Platform instance running on Red Hat Container Development Kit, in Developer Studio. We use the Helloworld-MSA tutorial available in GitHub at: <https://github.com/redhat-helloworld-msa/helloworld-msa>.

The article shows how you can easily build a local Docker image, not present on Docker Hub, to Container Development Environment and then deploy that image to an OpenShift Container Platform instance, using Developer Studio. **frontend** and **bonjour** microservices, used here, are examples of such private images that are not present in Docker Hub.

## Prerequisites

1. Install npm: Before running Developer Studio, install npm on your system. See the npm documentation for instructions for various platforms: <https://docs.npmjs.com/getting-started/what-is-npm>.
2. Download and install JDK 8.
3. Install Developer Studio and Red Hat Container Development Kit. To install Developer Studio, see: [https://access.redhat.com/documentation/en-us/red\\_hat\\_jboss\\_developer\\_studio/12.0/html/installation\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_jboss_developer_studio/12.0/html/installation_guide/) and to install Red Hat Container Development Kit, see [https://access.redhat.com/documentation/en-us/red\\_hat\\_container\\_development\\_kit/3.4/html/getting\\_started\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_container_development_kit/3.4/html/getting_started_guide/).
4. Clone the following projects and then import them into Developer Studio using the **Import** wizard (from **File > Open Projects from File System**).
  - a. **bonjour** project from: <https://github.com/redhat-helloworld-msa/bonjour>
  - b. **frontend** project from: <https://github.com/redhat-helloworld-msa/frontend>
5. Set up the oc client binaries in the IDE from **Window > Preferences**, expand **JBoss Tools**, and then click **OpenShift 3**.

### 2.3.1. Installing the javascript modules

After you complete this section, a new **node\_modules** folder is listed under the project in the **Project Explorer** view. This new folder indicates the download and installation of the required javascript modules.

## Prerequisites

- Install npm: Before running Developer Studio, install npm on your system. See the npm documentation for instructions for various platforms: <https://docs.npmjs.com/getting-started/what-is-npm>.
- Download and install JDK 8.
- Install Developer Studio and Red Hat Container Development Kit. To install Developer Studio, see: [https://access.redhat.com/documentation/en-us/red\\_hat\\_developer\\_studio/12.0/html/installation\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_developer_studio/12.0/html/installation_guide/) and to install Red Hat Container Development Kit, see [https://access.redhat.com/documentation/en-us/red\\_hat\\_container\\_development\\_kit/3.5/html/getting\\_started\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_container_development_kit/3.5/html/getting_started_guide/).
- Set up the oc client binaries in the IDE from **Window → Preferences**, expand **JBoss Tools**, and then click **OpenShift 3**.

## Procedure

1. In the **Project Explorer** view of the IDE, expand **frontend** and right-click **package.json**.
2. Click **Run As** → **npm Install** to download and install the required javascript modules in the project.

### 2.3.2. Building the frontend microservice

**frontend** is a Docker based microservice that is not present on Docker Hub. It is the landing page for the application that you are building. The **frontend** microservice then calls other microservices (**bonjour**, in this case) and displays the results from these calls. This section walks you through steps to build the **bonjour** private image.

#### Prerequisites

- A cloned and imported copy of the **bonjour** project from <https://github.com/redhat-helloworld-msa/bonjour> is available.

#### Procedure

1. In the **Project Explorer** view, expand **frontend**, right-click **Dockerfile**, and then click **Run As** → **Docker Image Build**.
2. Set the configuration in the **Docker Image Build Configuration** window:
  - a. In the **Connection** list, select **Container Development Environment**.
  - b. In the **Repository Name** field, type *demo/frontend*.
3. Click **OK**. The Docker-formatted Container image starts building against the Docker Daemon running in the Container Development Environment.

#### 2.3.2.1. Deploying the frontend microservice

You can deploy the **frontend** microservice into an OpenShift Container Platform instance running on Red Hat Container Development Kit, in Developer Studio. After you build the **frontend** microservice, the Docker-formatted container image **demo/frontend** is available in the **Docker Explorer** view under the **Container Development Environment** option.

#### Prerequisites

- A cloned and imported copy of the **frontend** project from <https://github.com/redhat-helloworld-msa/frontend> is available.

#### Procedure

1. In the **Docker Explorer** view of the IDE, select **Container Development Environment** → **Images**, right-click **demo/frontend**, and click **Deploy to OpenShift**.
2. In the **Deploy an Image** window, click **New**.
3. Create a new OpenShift project using the **Create OpenShift Project** window:
  - a. In the **Project Name** field, type the name of the new project, **demo**.

- b. Optionally, complete the **Display Name** and **Description** fields.
  - c. Click **OK**.
4. In the **Deploy an Image** window, select the **Push Image to Registry** check box and click **Next**.
5. In the **Deployment Configuration & Scalability** window, change the **OS\_PROJECT**. Click **OS\_PROJECT** to open the **Environment Variable** window and in the **Value** field, type *demo* (from step 5) and click **OK**.
6. In the **Deployment Configuration & Scalability** window, click **Next** and then click **Finish**. After the Docker-formatted Container image is pushed to the Docker Registry on OpenShift Container Platform, the Eclipse plugin generates all the required OpenShift Container Platform resources for the application to run.
7. In the **Deploy Image to OpenShift** window, review the details of deploying the image and click **OK**.
8. In the **OpenShift Explorer** view, expand the connection → *{project name}* → **Service** → **Pod** to see the Pod running. Right-click the pod and click **Pod Log**. The **Console** view shows the **frontend** service running. In the **OpenShift Explorer** view, expand the application, right-click the service, and click **Show In** → **Web Browser**. The **frontend** microservice in the Bonjour Service shows: **Error getting value from service <microservice>** This message confirms that the **bonjour** microservice is connected.

### 2.3.3. Connecting the frontend and bonjour microservices

The **bonjour** microservice is a node.js application that returns the string **bonjour-de-<pod\_ID>**. You can build the **bonjour** microservice and then view it on the **frontend** microservice to validate it.

#### Prerequisites

- A built and deployed **frontend** microservice is available.
- A cloned and imported copy of the bonjour project from <https://github.com/redhat-helloworld-msa/bonjour> is available.

#### Procedure

1. In the **Project Explorer** view, expand **bonjour** and right-click **package.json**.
2. Click **Run As > npm Install**.
3. In the **Project Explorer** view, expand **bonjour** and right-click **Dockerfile**.
4. Click **Run As > Docker Image Build**.
5. Set the configuration in the **Docker Image Build Configuration** window:
  - a. In the **Connection** list, select **Container Development Environment**.
  - b. In the **Repository Name** field, type *demo/bonjour*.
6. Click **OK**.

#### 2.3.3.1. Deploying the bonjour microservice

You can deploy the Docker-formatted Container image either from the **Docker Explorer** view (as done in step 3 of the **Building a Docker-formatted Container Image** section) or, as done in this section, from the **OpenShift Explorer** view of the IDE.

### Prerequisites

- A cloned and imported copy of the `bonjour` project from <https://github.com/redhat-helloworld-msa/bonjour> is available.

### Procedure

1. In the **OpenShift Explorer** view of the IDE, right-click the project (**demo**), and click **Deploy Docker Image**.
2. To deploy the image using the **Deploy an Image** window:
  - a. In the **Docker Connection** list, click the Docker connection.
  - b. In the **Image Name** field, type `demo/bonjour`.
  - c. Click the **Push Image to Registry** check box.
3. Click **Next**.
4. In the **Deployment Configuration & Scalability** window, click **Next**.
5. In the **Services and Routing Settings** window, click **Finish**.
6. In the **Deploy Image to OpenShift** window, click **OK**.

#### 2.3.3.2. Scaling the pod

Scaling pods in OpenShift enables you to use resources effectively. This section walks you through steps to increase the number of pods by scaling them up.

### Prerequisites

- A pod that is up and running.

### Procedure

1. In the **OpenShift Explorer** view of the IDE, expand the application name (**demo**).
2. Right-click the pod and click **Pod Log** to check if the pod is running.
3. Navigate to the browser where you have the OpenShift application running and click **Refresh Results**. You will see a greeting from the `bonjour` service with a hostname that matches the pod name in the **OpenShift Explorer** view.
4. In the **OpenShift Explorer** view, right-click the service and click **Scale** → **Up**. You now have two Pods running on OpenShift Container Platform.
5. Navigate to the browser and click **Refresh Results** to see the service balancing between the two Pods.

### 2.3.4. Editing the bonjour microservice

The **bonjour** microservice is a node.js application that returns the string **bonjour-de-<pod\_ID>**. This section walks you through editing this microservice.

#### Prerequisites

- The bonjour microservice is available.

#### Procedure

1. In the **Project Explorer** view, expand **bonjour**, and double-click **bonjour.js** to open it in the default editor.
2. Find the following function:

```
function say_bonjour(){  
    Return "Bonjour de " + os.hostname();
```

3. Change the function as shown in the following example:

```
function say_bonjour(){  
    Return "Salut de " + os.hostname();
```

4. Save the file.

#### 2.3.4.1. Viewing the edited bonjour microservice on the frontend microservice

You can use private images to build a local Docker image, not present on Docker Hub, on the Container Development Environment. The **frontend** and **bonjour** microservices are examples of such private images that are not present in Docker Hub. Then, you can deploy the private image to an OpenShift Container Platform instance, using Developer Studio.

#### Prerequisites

- The bonjour microservice is available.

#### Procedure

1. In the **Project Explorer** view, expand **bonjour** and right-click **Dockerfile**.
2. Click **Run As** → **Docker Image Build**.
3. Optional:, To edit the configuration, open the **Run Configuration** window.
4. Wait for the **Console** view to show that the Docker-formatted container image is successfully pushed to the Docker daemon.
5. In the **Docker Explorer** view, expand **Container Development Environment > Images**.
6. Right-click the image and click **Deploy to OpenShift**.
7. In the **Deploy an Image** window, click **Push Image to Registry** and then click **Next**.

8. In the **Deployment Configuration & Scalability** window, click **Finish**. In the **OpenShift Explorer** view under **bonjour**, you can see that the pods are added running.
9. Navigate to the browser and click **Refresh Results**. The new greeting displays.

## 2.4. DEPLOYING THE OPENSIFT CONTAINER PLATFORM 3 RESOURCE

In this article, you use the **s2i-spring-boot-cfx-jaxrs** template in OpenShift Container Platform as an example to define resources for your OpenShift Container Platform application. Use similar steps to define resources for any other OpenShift Container Platform application.

### Prerequisites

1. Install Red Hat Container Development Kit (CDK) 3. For detailed instructions to install CDK 3, see [https://access.redhat.com/documentation/en-us/red\\_hat\\_container\\_development\\_kit/3.0/html/installation\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_container_development_kit/3.0/html/installation_guide/).

### 2.4.1. Deploying the s2i-spring-boot-cfx-jaxrs template

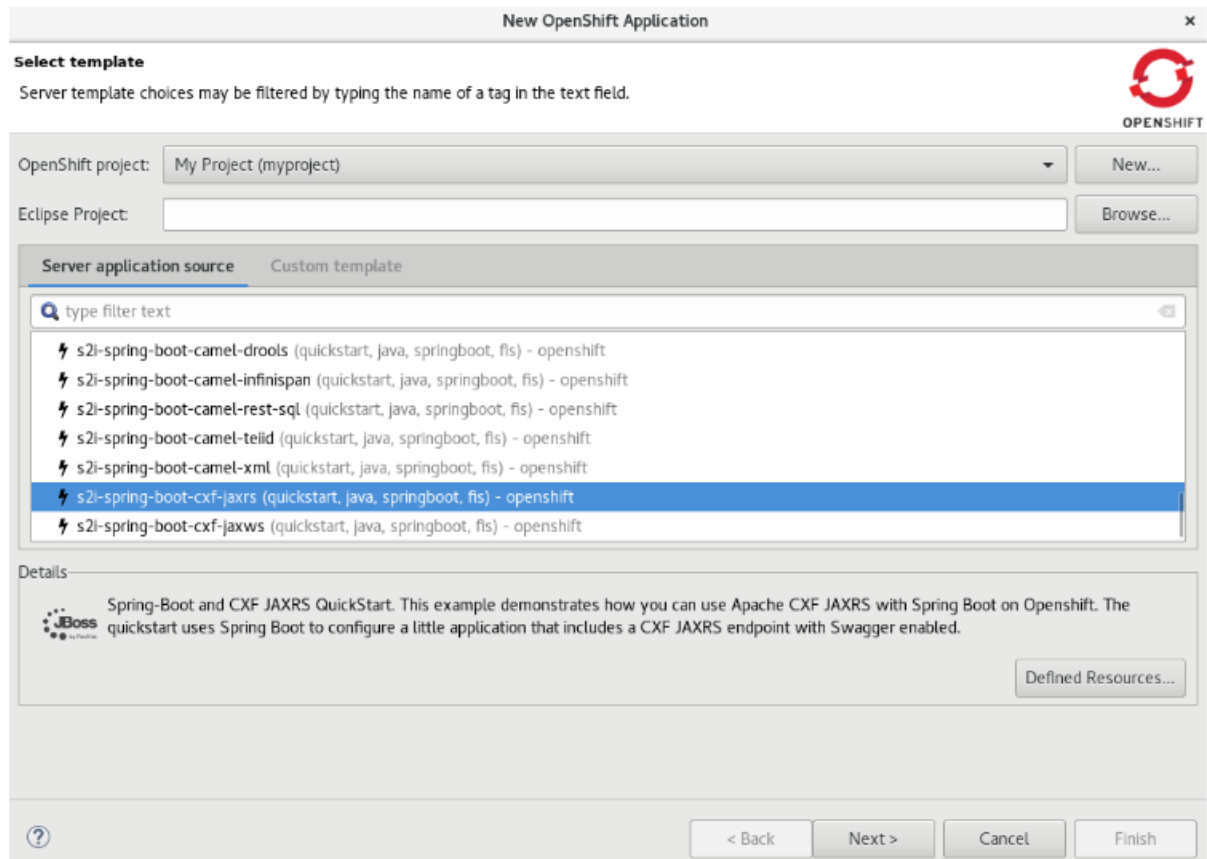
Set up the IDE to work with CDK 3 as described in [Using Container Development Kit Tooling in Red Hat Developer Studio](#). The new connection for OpenShift Container Platform is listed in the **OpenShift Explorer** view, making the **s2i-spring-boot-cfx-jaxrs** template available for use.

### Procedure

To deploy the template, take the following steps:

1. In the **OpenShift Explorer** view, expand the connection and right-click the *{project name}* and click **New > Application**.
2. In the **New OpenShift Application** window:
  - a. In the **OpenShift project** field, click the project that you want to create the new application in.
  - b. In the **Server application source** tab, scroll through the list and locate and click **s2i-spring-boot-cfx-jaxrs (quickstart, java, springboot, fis) - openshift**.
3. Click **Finish**.
4. In the **Create Application Summary** window, click **OK**.
5. In the **Import OpenShift Application** window in the **Git Clone Location** field, enter the location where you want to clone the template and click **Finish**.

Figure 2.9. Selecting the s2i-spring-boot-cfx-jaxrs Template



- In the **OpenShift Explorer** view, expand the project, expand the **s2i-spring-boot-cfx-jaxrs** application and then right-click the **s2i-spring-boot-cfx-jaxrs-1** build and click **Build Log**. The **Console** view shows the progress of the build. The **Console** view shows the **latest: digest: sha256:{checksum} size: 9033 Push successful** message.

## 2.4.2. Viewing the s2i-spring-boot-cfx-jaxrs application in the Web Console

This section is required optionally if you want to see the application running on the terminal. In absence of a service or route, you can view the application through the **Pod** tab in the web console.

### Procedure

To view the application in the Pod:

- In the web console, click **Applications > Pod** and then click the **s2i-spring-boot-cfx-jaxrs-1** pod.
- Click the **Logs** tab.
- In the logs, locate **Jolokia: Agent started with URL <https://172.17.0.6:8778/jolokia/>** and copy the IP address (*172.17.0.6*, in this example).
- Click the **Terminal** tab.
- In the terminal, type the following command: **curl [http://{IP\\_address}:8080/services/helloservice/sayHello](http://{IP_address}:8080/services/helloservice/sayHello)**.  
Example: **curl <http://172.17.0.6:8080/services/helloservice/sayHello>**
- Press Enter.



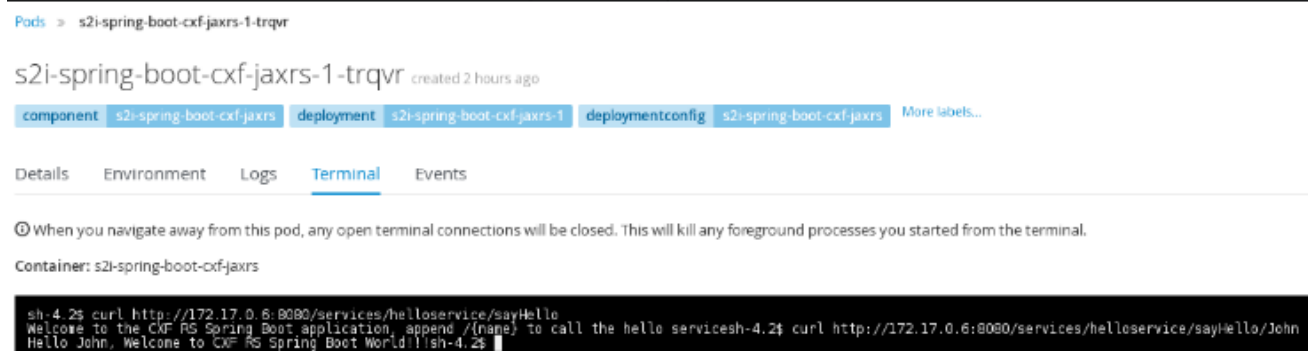
7. Append the next line with: **curl**

[http://172.17.0.6:8080/services/helloservice/sayHello/{your\\_name}](http://172.17.0.6:8080/services/helloservice/sayHello/{your_name}) and press Enter.

Example: **curl** <http://172.17.0.6:8080/services/helloservice/sayHello/John>

The **Hello John, Welcome to CXF RS Spring Boot World!!!** message appears, showing that application is up and running.

**Figure 2.10. s2i-spring-boot-cxf-jaxrs Application in the Web Console**



### 2.4.3. Defining services and routes using a JSON file

Use the `services-route.json` file to create the service for the `s2i-spring-boot-cxf-jaxrs` application and then create a route for the service. In this case the target port is 8080 where the route sends the request to the application.

#### Procedure

To define the resources, take the following steps:

1. Copy the following content and paste it in a file, name the file `services-routes.json`, and save it.

```
{
  "apiVersion": "v1",
  "kind": "List",
  "metadata": {},
  "items": [
    {
      "apiVersion": "v1",
      "kind": "Service",
      "metadata": {
        "name": "s2i-spring-boot-cxf-jaxrs"
      },
      "spec": {
        "ports": [
          {
            "name": "8080-tcp",
            "protocol": "TCP",
            "port": 8080,
            "targetPort": 8080
          },
          {
            "name": "8778-tcp",
            "protocol": "TCP",

```

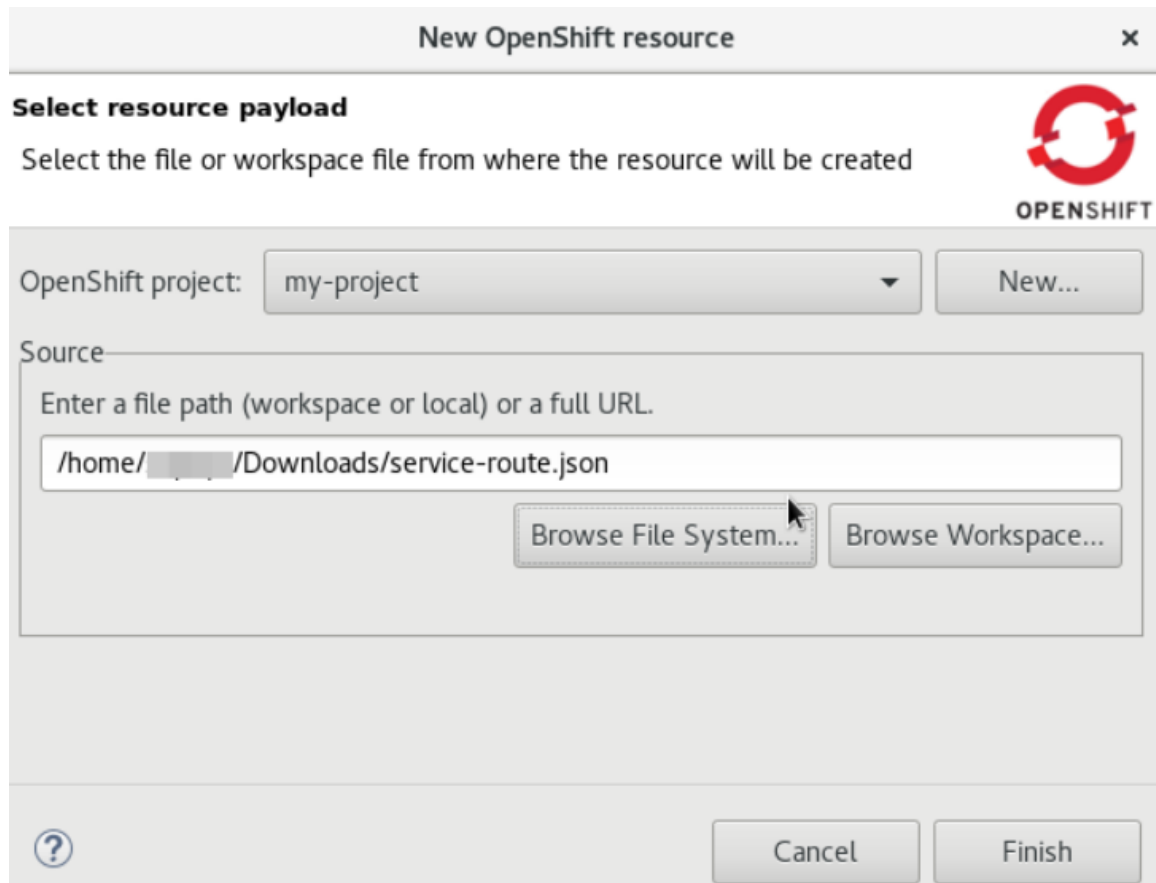
```

        "port": 8778,
        "targetPort": 8778
      }
    ],
    "selector": {
      "deploymentconfig": "s2i-spring-boot-cxf-jaxrs"
    }
  }
},
{
  "apiVersion": "v1",
  "kind": "Route",
  "metadata": {
    "name": "s2i-spring-boot-cxf-jaxrs"
  },
  "spec": {
    "to": {
      "kind": "Service",
      "name": "s2i-spring-boot-cxf-jaxrs",
      "weight": 100
    },
    "port": {
      "targetPort": "8080-tcp"
    },
    "wildcardPolicy": "None"
  }
}
]
}

```

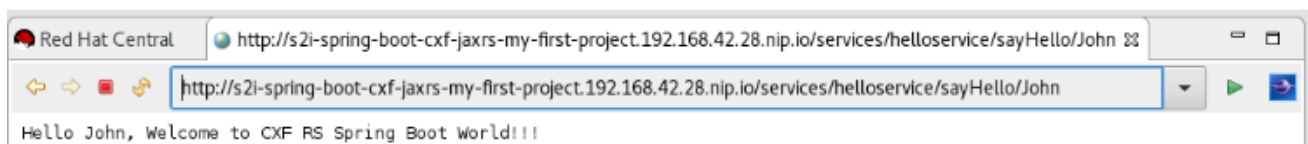
2. In the **OpenShift Explorer** view, right-click the project and click **New > Resource**.
3. In the **New OpenShift Resource** window:
  - a. In the **OpenShift project** list, click the project that you deployed the application to.
  - b. In the **Source** pane, click **Browse File System** and locate and select the **services-routes.json** file.
  - c. Click **Finish**.

Figure 2.11. Selecting the service-routes.json File



4. The **Create Resource Summary** window shows the details of the created service and route. Click **OK**.
5. In the **OpenShift Explorer** view, right-click the project and click **Show in > Web browser**. The **Whitelabel Error Page** shows that the application has no explicit mapping.
6. In the address bar, append the URL with **services/hello-service/sayHello/**. The URL should now look like: [http://s2i-spring-boot-cxf-jaxrs-\*{IP\\_address}\*.nip.io/services/hello-service/sayHello/](http://s2i-spring-boot-cxf-jaxrs-<i>{IP_address}</i>.nip.io/services/hello-service/sayHello/).
7. Press **Enter**. The web browser shows the **Welcome to the CXF RS Spring Boot application, append *{name}* to call the hello service** message.
8. At the end of the URL, append a name, for example: [http://s2i-spring-boot-cxf-jaxrs-\*{IP\\_address}\*.nip.io/services/hello-service/sayHello/John](http://s2i-spring-boot-cxf-jaxrs-<i>{IP_address}</i>.nip.io/services/hello-service/sayHello/John). The page displays the message: **Hello John, Welcome to CXF RS Spring Boot World!!!**

Figure 2.12. Viewing the s2i-spring-boot-cxf-jaxrs Application in the Web Browser



## CHAPTER 3. DEVELOPING WITH DOCKER

### 3.1. USING DOCKER TOOLING IN DEVELOPER STUDIO

The DevStudio Docker tooling allows you to manage Docker Images and Containers. The Docker tooling functionality is the same as running the docker commands on the CLI. Docker tooling is the GUI-based interface of the Docker commands on the CLI.

#### 3.1.1. Connecting to Docker Daemon

You must have a connection to a Docker daemon before you can work with Docker Images or Containers.

##### Installing Docker

1. Install Docker on your system. To install it on different platforms:
  - On Windows, use: <https://docs.docker.com/docker-for-windows/install/>
  - On Fedora, use: <https://fedoraproject.org/wiki/Docker> or <https://docs.docker.com/engine/installation/>.
  - On MacOS, use: <https://docs.docker.com/docker-for-mac/install/>
2. After you have installed Docker, grant the user permission to work with Docker by running the following commands as the root user:

```
# groupadd docker && sudo gpasswd -a ${USER} docker && sudo  
systemctl restart docker  
# usermod -aG docker ${USER}
```

Where, {USER} is your username.

3. Either restart the system or log out and log into the system again for the changes to take effect.

##### Setting up an account in the Docker tooling

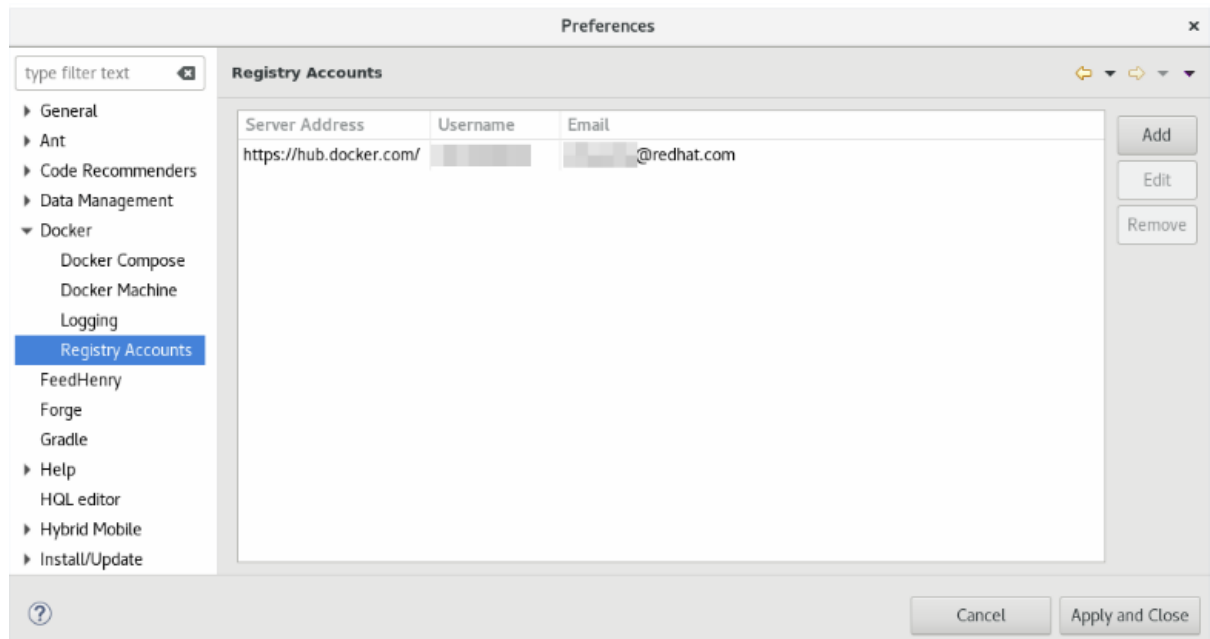


#### NOTE

You must set up the account in Docker tooling only if you want to push the images to the remote registry. If not, you may skip this section.

You must have an account set up in Docker Tooling. To do that:

1. In the IDE, click **Window > Preferences > Docker > Registry Accounts**.

**Figure 3.1. Creating a New Registry Account**

2. Click **Add** to display the **New Registry Account** window.
3. In **New Registry Account** window:
  - a. In the **Server Address** field, type <https://hub.docker.com/> (in this example, we are using <https://hub.docker.com/> as the Docker Registry).
  - b. In the **Username** field, type the username that you use on [hub.docker.com](https://hub.docker.com/) (if you don't already have a Docker ID as yet, navigate to <https://hub.docker.com/> and create one).
  - c. In the **Email** field, type the email address that you used on <https://hub.docker.com/>.
  - d. In the **Password** field, type the password for <https://hub.docker.com/>.
  - e. Click **OK**.

**Figure 3.2. Entering Details for the New Registry Account**

**New Registry Account** [X]

Add/Edit a Registry Account

Server Address:

Username:

Email:

Password:

Cancel OK

f. The **Preferences** window shows the details that you just entered. Click **Apply** and **Close**.

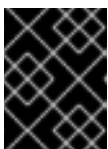
### 3.1.1.1. Testing an existing Docker connection

You must be connected to a Docker daemon before you can manage Docker images or containers.

#### Procedure

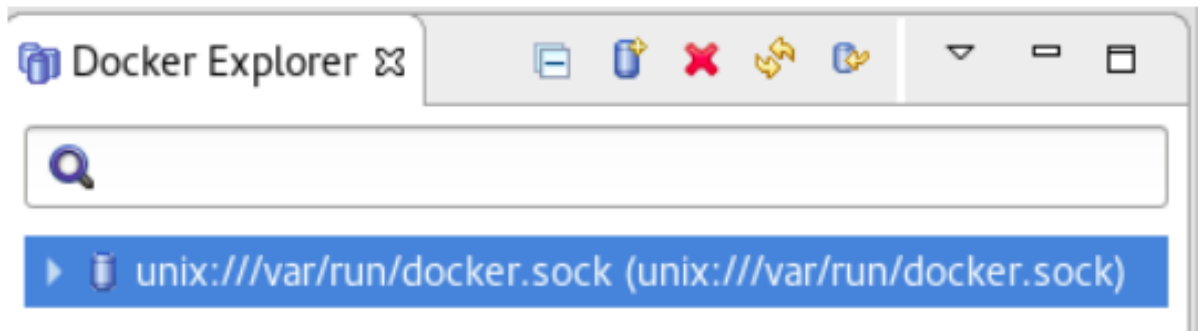
To display the **Docker Explorer** view, take the following steps:

1. Click **Window > Show View > Other** (or click **Window > Perspective > Open Perspective > Other** and then click **Docker Tooling**).
2. In the **type filter text** field, type **Docker** and from the results click **Docker Explorer**. If you already have Docker installed on your system, the Unix socket location displays in the **Docker Explorer** view.



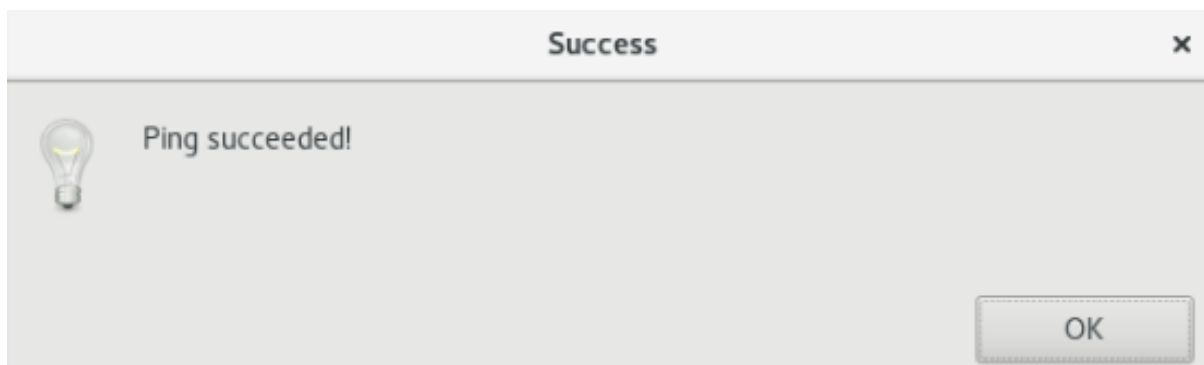
#### **IMPORTANT**

In case the Unix socket does not appear, refer to the note in the **Installing Docker** section to grant user the permissions required to work with Docker.

**Figure 3.3. Unix Socket Location Displayed in the Docker Explorer View**

To test the connection:

3. In the **Docker Explorer** view, right-click `unix:///var/run/docker.sock` and click **Edit** to open the **Edit Docker Connection** window.
4. Click **Test Connection**. The **Ping succeeded!** message confirms the connection.

**Figure 3.4. The Ping succeeded! Message Confirming the Connection****NOTE**

In case your connection to the Docker daemon is lost, use the following two commands to restart the Docker daemon:

```
sudo systemctl start docker
sudo systemctl enable docker
```

**3.1.1.2. Editing a Docker connection****Procedure**

To edit a Docker connection, atke teh following steps:

1. In the **Docker Explorer** view, right-click the connection and click **Edit**. The **Edit Docker Connection** window opens.
2. Click **Browse** next to the **Unix socket Location** field to locate a new location of the Unix socket (or, check the TCP connection and add the URI).
3. After you have selected the new location, click **OK** and then click **Finish**.  
To filter the **Docker Explorer** view:

4. In the **Docker Explorer** view, click the **View Menu** arrow and then click **Filters and Customization**.
5. Select the appropriate options to filter out the required images:
  - a. Click the **Dangling (untagged) images** checkbox, to filter out images that are no longer referred to.
  - b. Click the **Intermediate images** checkbox, to filter out the images that have no repo tags that are parents of named images.
  - c. Click the **Stopped containers** checkbox, to filter out containers that are stopped but not paused.
  - d. Click the **Top-level images** checkbox, to show the first repo tag for an Image with multiple tags. You have performed various management operations on your Docker container.

### 3.1.2. Managing Docker images

In this section you will work with the `jboss/wildfly:11.0.0.Final` image.

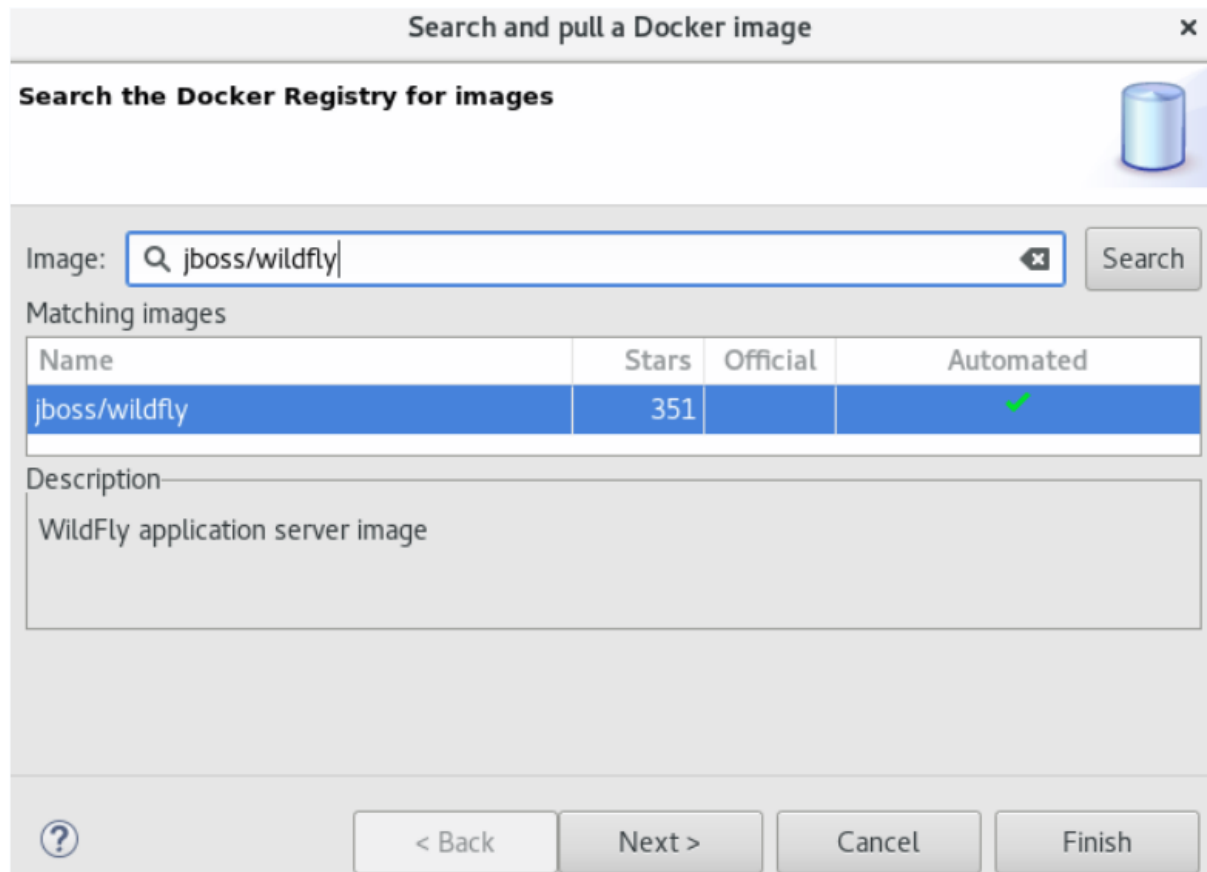
#### 3.1.2.1. Pulling the `jboss/wildfly:11.0.0.Final` image

##### Procedure

To pull the `jboss/wildfly:11.0.0.Final` image, take the following steps:

1. In the **Docker Explorer** view, expand the docker URL > **Images**.
2. Right-click the **Images** folder and click **Pull**. The **Pull Image** window opens.
3. In the **Pull Image** window, the Docker Hub registry is used by default. To specify an additional private registry, click the **Add a registry account** link.
4. Click **Search** to display the **Search and pull a Docker Image** window.
5. In the **Image** field, type `jboss/wildfly` and press **Enter**



**Figure 3.5. Searching for the jboss/wildfly Image**

6. Select the *jboss/wildfly* image and click **Next**.
7. In the **Choose a tag for the selected image** window, locate and click **11.0.0.Final**.
8. Click **Finish**. The **Pull Image** window, **Image name** field shows **jboss/wildfly:11.0.0.Final**.
9. Click **Finish**. The notification area in the IDE shows the progress of the image being pulled. Wait for the pull task to complete. This may take time because an Image may use several intermediate Images each of which may be several bytes.

In the **Docker Explorer** view, expand the docker URL > **Images**. The **docker.io/jboss/wildfly:11.0.0.Final** image is listed.

### 3.1.2.2. Pushing images

After you push an image to the Docker Registry or to your private registry, the image becomes available in the Docker Cloud. This image is then available for use by other developers.

#### Prerequisites

- Before you try to push, an Image it is important that you tag your Image.

To tag the Image, take the following steps:

1. In the **Docker Explorer** view, expand the docker URL > **Images**.
2. Right-click the image name that you want to add a tag to. Click **Add Tag**.
3. In the **Tag Image** window:

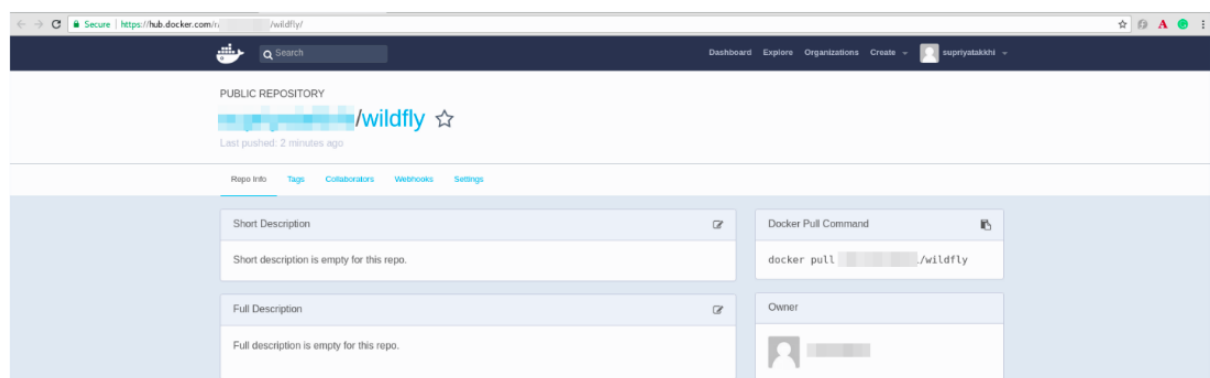
- a. In the **New Tag** field, type `docker_hub/<your_username_on_docker_hub>/<image tag>:<version>` (here: `docker.io/<your_username_on_docker_hub>/wildfly:11.0.0.Final`).
4. Click **Finish**. The new image with the tag appears in the **Images** folder: `docker.io/<your_username_on_docker_hub>/wildfly:11.0.0.Final`.

## Procedure

To push an Image, atke teh following steps:

1. In the **Docker Explorer** view, expand the docker URL > **Images**.
2. Right-click the image name that you want to Push. Click **Push**. The **Push Image** window opens.
3. In the **Registry account** list, by default, the `docker.io` account appears. Change this to select the account that shows `<your_dockerhub_username>@<dockerhub URL>`.
4. Ensure that the **Image name** list displays the name of the image that you want to push. If not, select the image name from the list.
5. Click **Finish**. The notification area in the IDE shows the progress of the image being pushed. Wait for the push task to complete. After it is complete, the pushed Image is available in your public repository at <https://hub.docker.com/>.

**Figure 3.6. Pushed Image in the Public Repository**



### 3.1.2.3. Running Image Launch Configuration

In this section you will use the **Run Image** wizard to create a Container based on an Image.

## Procedure

To run an Image, take the following steps:

1. In the **Docker Explorer** view, expand the docker URL > **Images**.
2. Right-click the image name that you want to run (`docker.io/<your_username_on_docker_hub>/wildfly:11.0.0.Final`, in this case). Click **Run**.
3. In the **Run a Docker Image** window:
  - a. The **Image** field, by default, shows the name of the image you are running. To run an Image that is not currently loaded, type the Image name in this field.
  - b. In the **Container Name** field, type a name for the container.
  - c. Clear the **Publish all exposed ports to random ports on the host interfaces** check box.

d. Click *Finish*.

**Figure 3.7. Running the Image Launch Configuration**

Run a Docker Image ×


**Docker Container settings** 

Image:  Search...

[Pull this image...](#)

Container Name:

Entrypoint:

Command:

Publish all exposed ports to random ports on the host interfaces

Only publish the selected container ports below to the host:

Container Port	Type	Host Address	Host Port
<input checked="" type="checkbox"/> 8080	/tcp		8080

Add...  
Edit...  
Remove

Links to other containers:

Container Name	Alias

Add...  
Edit...  
Remove

Keep STDIN open to Console even if not attached (-i)

Allocate pseudo-TTY from Console (-t)

Automatically remove the container when it exits (--rm)

Give extended privileges to this container (--privileged)

Use unconfined seccomp profile (--securityOpt seccomp=unconfined)

Add basic security (--readonly --tmpfs /run --tmpfs /tmp --cap-drop=all)

?
< Back
Next >
Cancel
Finish

The **Console** view is the view in focus showing the progress of the task. The **WildFly Full 11.0.0.Final (WildFly Core 3.0.8.Final)** started in 3188ms - Started 292 of 553 services

*(347 services are lazy, passive or on-demand)* message indicates that the image is started.

4. In the web browser navigate to <http://localhost:8080/> to see the Image running.

**Figure 3.8. Image Running on Localhost**



### 3.1.2.4. Building images with Dockerfile

You can build an Image or create an Image by modifying an existing image. Typically, this involves installing new packages. The specification of the new Docker Image is done via a special file which must be named **Dockerfile**

#### Prerequisites

- You must have a Dockerfile created on your local machine.

Following is an example of a Dockerfile. You may use this sample file or have your own Dockerfile.

```
# Use latest jboss/base-jdk:8 image as the base
FROM jboss/base-jdk:8
# Set the WILDFLY_VERSION env variable
ENV WILDFLY_VERSION 11.0.0.Final
ENV WILDFLY_SHA1 0e89fe0860a87bfd6b09379ee38d743642edfcfb
ENV JBOSS_HOME /opt/jboss/wildfly
USER root
# Add the WildFly distribution to /opt, and make wildfly the owner of the
extracted tar content
# Make sure the distribution is available from a well-known place
RUN cd $HOME \
  && curl -O https://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-
$WILDFLY_VERSION.tar.gz \
  && sha1sum wildfly-$WILDFLY_VERSION.tar.gz | grep $WILDFLY_SHA1 \
  && tar xf wildfly-$WILDFLY_VERSION.tar.gz \
  && mv $HOME/wildfly-$WILDFLY_VERSION $JBOSS_HOME \
  && rm wildfly-$WILDFLY_VERSION.tar.gz \
  && chown -R jboss:0 ${JBOSS_HOME} \
  && chmod -R g+rw ${JBOSS_HOME}
# Ensure signals are forwarded to the JVM process correctly for graceful
shutdown
```

```
ENV LAUNCH_JBOSS_IN_BACKGROUND true
USER jboss
# Expose the ports we're interested in
EXPOSE 8080
# Set the default command to run on boot
# This will boot WildFly in the standalone mode and bind to all interface
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0"]
```

## Procedure

To build an image, take the following steps:

1. Open the **Docker Images** view by clicking **Window** → **Show View** → **Docker** → **Docker Images**.
2. Click the **Build Image** icon (`image::docker_build_image_icon.png[width=25px]`).
3. In the **Build a Docker Image** window:
  - a. In the **Image Name** field, type a new name for the Image in the **repo/name:version** format (**mydockerrepo/wildfly:11.0.0.Final**, in this case).
4. Click **Browse** next to the **Directory** field to locate and select the Dockerfile.
5. Click **Finish**. The **Console** view is the view in focus showing the progress of the image being built. The **Successfully built <image\_ID>** message shows that the image has been built successfully.

### 3.1.2.5. Working with image tags

Tags are additional names for images. They are usually in the following format:  
**docker\_hub/<your\_username\_on\_docker\_hub>/<image\_tag>:<version>**.

#### 3.1.2.5.1. Adding tags to images

## Procedure

To add a tag to an image, take the following steps:

1. In the **Docker Explorer** view, right-click the image name that you want to add a tag to and, from the context menu, click **Add Tag**.
2. In the **Tag Image** window:
  - a. In the **New Tag** field, type `docker_hub/<your_username_on_docker_hub>/<image_tag>:<version>` (in this case, **docker.io/<your\_username\_on\_docker\_hub>/wildfly:11.0.0.Final**).
  - b. Click **Finish**.

#### 3.1.2.5.2. Removing tags for images

## Procedure

To remove a tag, take the following steps:

1. In the **Docker Explorer** view, right-click the image name that you want to remove the tag for.

2. From the context menu, click **Remove Tag**.
3. In the **Remove Image Tag** window:
  - a. Ensure that the **Tag** field, shows the name of the image that you want to remove the tag for. If not, select the appropriate image from the list.
  - b. Click **Finish**. The image is no longer listed in the **Docker Explorer** view.

### 3.1.3. Managing Docker Containers

Docker containers are isolated processes that are based on Docker Images. Once created, users can stop, start, pause, unpause, kill, or remove the containers, or read their logs. To manage the Docker Containers:

#### Procedure

1. Click **Window** → **Show View** → **Other**.
2. In the **filter text field**, type Docker to view Docker-related options in the list.
3. Expand **Docker** and double-click **Docker Containers**. The **Docker Containers** view appears displaying a list of all containers running on the Docker host.
4. Click the desired container to select it. You can now manage your containers using the buttons in the **Docker Container** view header:
  - a. To pause the container, click **Pause**.
  - b. To start the container, click **Start**.
  - c. To view the container logs, right-click the container name and click **Display Log**.
  - d. To view a list of all containers, click on the right-most icon in the list of icons in the view, which displays a drop-down option to view all containers. Click this option to view all available containers.

### 3.1.4. Working with docker-compose.yml files

You can use a **docker-compose.yml** file and start Docker Compose from the context menu of a project in the **Project Explorer** view. The **docker-compose.yml** file contains the configuration that is applied to each container started for a service. Docker Compose is a tool for running applications composed from containers.

#### Prerequisites

1. Install Docker Compose: For instruction, see <https://docs.docker.com/compose/install/>.
2. Set up Docker Compose in DevStudio: Navigate to **Window** → **Preferences** → **Docker** → **Docker Compose**. In the **Docker Compose** field, select the location where Docker Compose is installed (usually, `/usr/local/bin`). Click **Apply and Close**.

#### 3.1.4.1. Creating the docker-compose project

#### Procedure

To create the docker-compose project, take the following steps:

1. In the IDE, click **File** → **New** → **Project**.
2. In the **New Project** wizard, click **General** > **Project**. Click **Next**.
3. In the **Project name** field, type `docker-compose` and click **Finish**. The new docker-compose project is listed in the **Project Explorer** view.

### 3.1.4.2. Creating the required files in the docker-compose project

In this section you will create the files required to run the docker-compose build.

#### Procedure

To create the files in the docker-compose project, take the following steps:

1. In the **Project Explorer** view, right-click **docker-compose** and click **New** > **File**.
2. In the **New File** window, **File name** field, type `Dockerfile` and click **Finish**. The `Dockerfile` opens in the default editor.
3. In the `Dockerfile`, copy and paste the following content and save the file.

```
1FROM python:3.4-alpine
2ADD . /code
3WORKDIR /code
4RUN pip install -r requirements.txt
5CMD ["python", "app.py"]
```

4. In the **Project Explorer** view, right-click **docker-compose** and click **New** > **File**.
5. In the **New File** window, **File name** field, type `app.py` and click **Finish**. The `app.py` file opens in the default editor.
6. In the `app.py` file, copy and paste the following content and save the file.

```
from flask import Flask
from redis import Redis

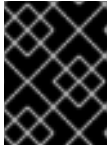
app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

7. In the **Project Explorer** view, right-click **docker-compose** and click **New** > **File**.
8. In the **New File** window, **File name** field, type `docker-compose.yml` and click **Finish**. The `docker-compose.yml` file opens in the default editor.

- In the `docker-compose.yml` file, copy and paste the following content and save the file.



### IMPORTANT

The indentation for the following file must be retained as shown here. If not retained, the `docker-compose.yml` file will have errors and will not build.

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

- In the **Project Explorer** view, right-click `docker-compose` and click **New > File**.
- In the **New File** window, **File name** field, type `requirements.txt` and click **Finish**. The `requirements.txt` file opens in the default editor.
- In the `requirements.txt` file, copy and paste the following content and save the file.

```
1 | flask
2 | redis
```

= Building the docker-compose project

## Procedure

To build the docker-compose project:

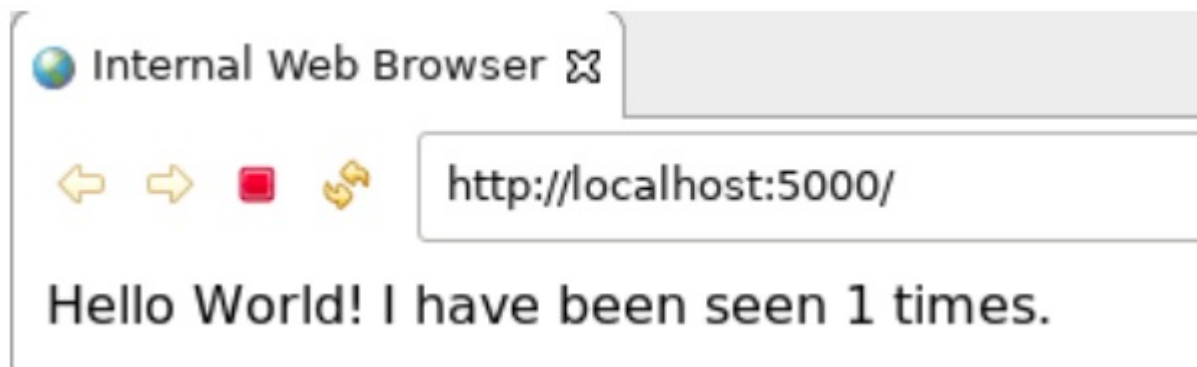
- In the **Project Explorer** view, expand the `docker-compose` project.
- Right-click the `docker-compose.yml` file, click **Run As > Docker Compose**. The **Console** view is the view in focus showing the progress of the of the image being pulled.

### Figure 3.9. Console View Showing Progress of the Image Being Pulled

```
Status: Downloaded newer image for docker.io/redis:alpine
Creating dockercompose_web_1 ...
Creating dockercompose_redis_1 ...
Creating dockercompose_web_1
Creating dockercompose_redis_1
[[1A]]2K
Creating dockercompose_web_1 ... done
[[1B]]1A]]2K
Creating dockercompose_redis_1 ... done
[[1B]]1A]]2K
Attaching to dockercompose_web_1, dockercompose_redis_1
web_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1 | * Restarting with stat
web_1 | * Debugger is active!
web_1 | * Debugger PIN: 327-853-737
redis_1 | 1:C 20 Dec 09:35:24.924 # 000000000000 Redis is starting 000000000000
redis_1 | 1:C 20 Dec 09:35:24.924 # Redis version=6.0.6, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1 | 1:M 20 Dec 09:35:24.926 * Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1 | 1:M 20 Dec 09:35:24.926 * Running mode=standalone, port=6379
redis_1 | 1:M 20 Dec 09:35:24.926 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis_1 | 1:M 20 Dec 09:35:24.926 # Server initialized
redis_1 | 1:M 20 Dec 09:35:24.926 # WARNING overcommit memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
redis_1 | 1:M 20 Dec 09:35:24.927 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
redis_1 | 1:M 20 Dec 09:35:24.927 * Ready to accept connections
```

The **Docker Explorer** view shows the containers running. Navigate to `localhost:5000` to see the application running.



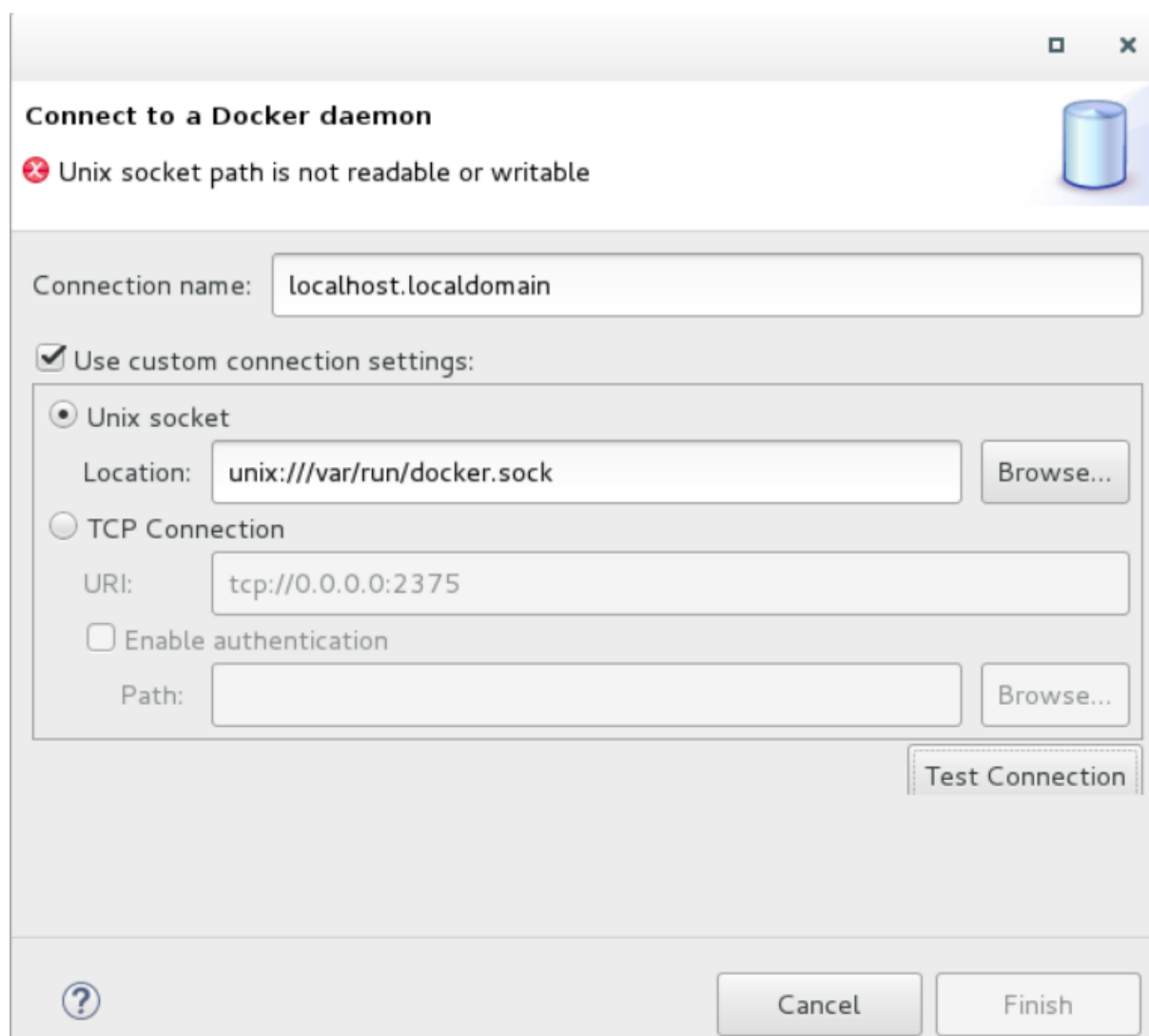
**Figure 3.10. Application Running on Localhost**

### 3.1.5. Troubleshooting

#### Procedure

Attempting to connect to a running local Docker instance as a non-root user results in errors being logged, but not displayed in the User Interface, which results in the error being non-obvious. The following workarounds are available for this problem:

- Connect to the Docker instance manually. Define a custom configuration file and specify the TCP URL displayed by the `systemctl status docker` service. As an example, you can use a TCP URL such as `tcp://0.0.0.0:2375` to connect to the running Docker instance instead of the default `unix:///var/run/docker.sock` configuration file.

**Figure 3.11. Error while Connection to Docker Daemon**

- *Run Eclipse as root. This solution avoids the problem but is not the recommended solution.*