Red Hat JBoss Developer Studio 11.0

Getting Started with Container and Cloud-based Development

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

Last Updated: 2017-10-16

Red Hat JBoss Developer Studio 11.0 Getting Started with Container and Cloud-based Development

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

Misha Husnain Ali
mhusnain@redhat.com

Supriya Takkhi
sbharadw@redhat.com

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. DEVELOPING USING CONTAINERS AND THE CLOUD

1.1. USING RED HAT CONTAINER DEVELOPMENT KIT 3.X TOOLING IN JBOSS DEVELOPER STUDIO 11.X

1.1.1. About Using Red Hat Container Development Kit with the IDE

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 JBoss Developer Studio (DevStudio), without spending additional time in setting up and configuring the supplementary tooling.

Following are the two ways to install CDK with DevStudio:

1. Installing CDK and DevStudio Using the Red Hat Development Suite Installer
2. Installing CDK and DevStudio as Separate Products

Once installed, you can use the installed components with the Docker Tooling.

1.1.2. Installing CDK and DevStudio Using the Red Hat Development Suite Installer

Use the Red Hat Development Suite (DevSuite) Installer to install CDK, DevStudio, and other relevant components. The Installer automatically configures these components for use together. This option is currently available for Windows, macOS, and RHEL 7. For instructions about using the DevSuite Installer, see Red Hat Development Suite Installation Guide.

The installation configures CDK tooling (creates the CDK server adapter, passes user credentials for the **access.redhat.com** domain used in the Installer). This makes the CDK tooling ready for use in the IDE. After the installation is complete, run DevStudio directly from the Installer, open the **Servers** view and start the Container Development Environment server adapter. This will start CDK and create OpenShift Container Platform and Docker connections. Note that if you close the Installer window and then run DevStudio, your user credentials will not be passed automatically and you must enter your user credentials manually.

1.1.3. Installing CDK and DevStudio as Separate Products

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

1.1.3.1. 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
  - CDK 3.0

  - DevStudio 11.0

- Ensure that you have a Red Hat Developer account. For a new account, visit https://developers.redhat.com/.

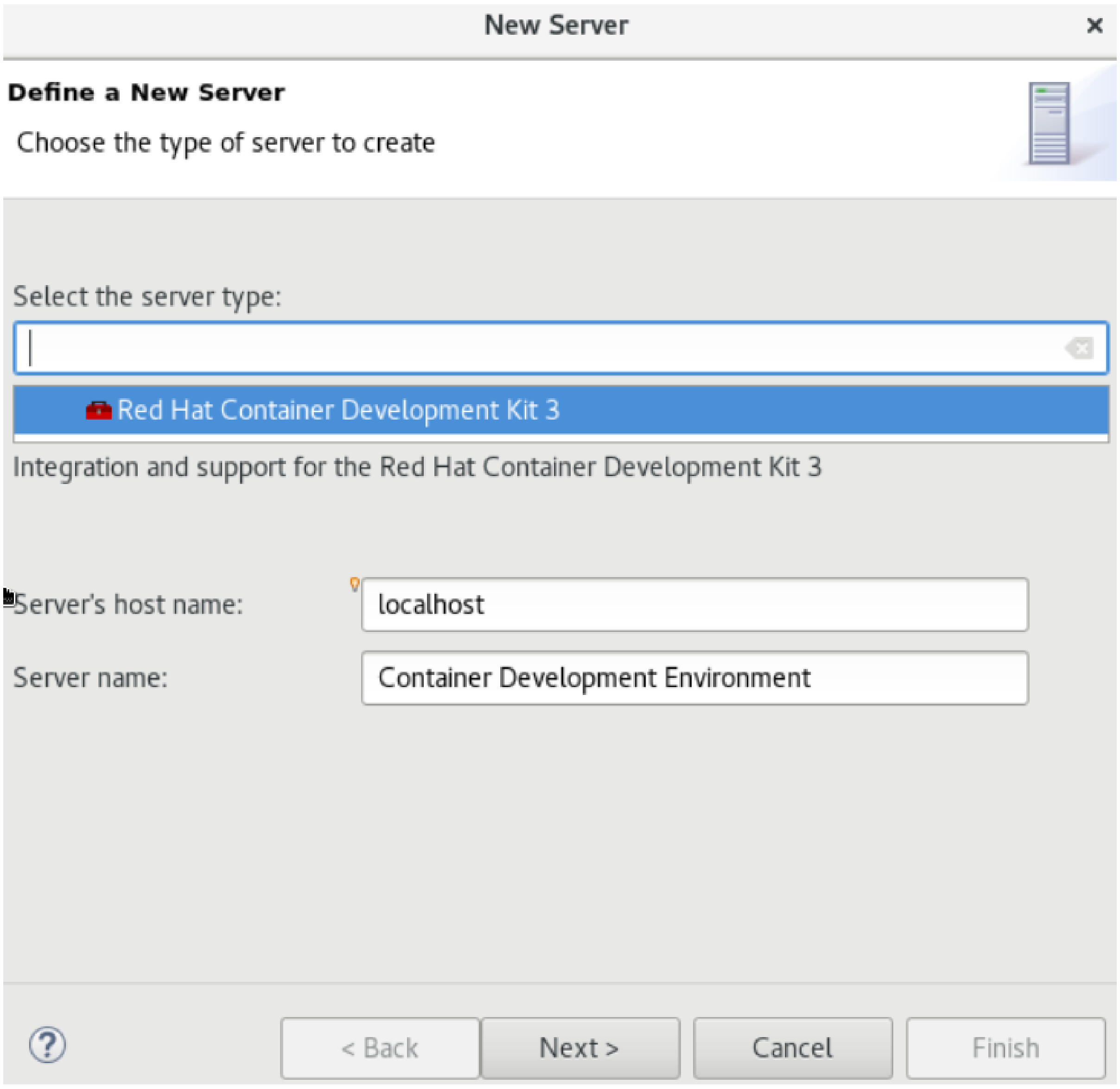
For details about installing these prerequisites, see the Red Hat Container Development Kit Installation Guide.

1.1.3.2. Set Up the CDK Connection in the IDE

To set up the CDK connection in the IDE:

1. Start the IDE.
2. Press Ctrl+3 and in the **Quick Access** bar, type *CDK*.
3. From the results, click **Launch Container Development Environment using Red Hat CDK**.
4. If asked, enter your user credentials.
5. In the **New Server** dialog box:
    a. Ensure that **Red Hat Container Development** is selected by default.
    b. In the **Server's host name** field, type the desired server host name.
    c. In the **Server Name** field, type the desired server name.
6. Click **Next** to continue.

Figure 1.1. Selecting Red Hat Container Development Kit 3



New Server

Define a New Server

Choose the type of server to create

Select the server type:

Red Hat Container Development Kit 3

Integration and support for the Red Hat Container Development Kit 3

Server's host name: localhost

Server name: Container Development Environment

7. In the **New Server Red Hat Container Development Environment** window, add the security information and your **access.redhat.com** credentials:

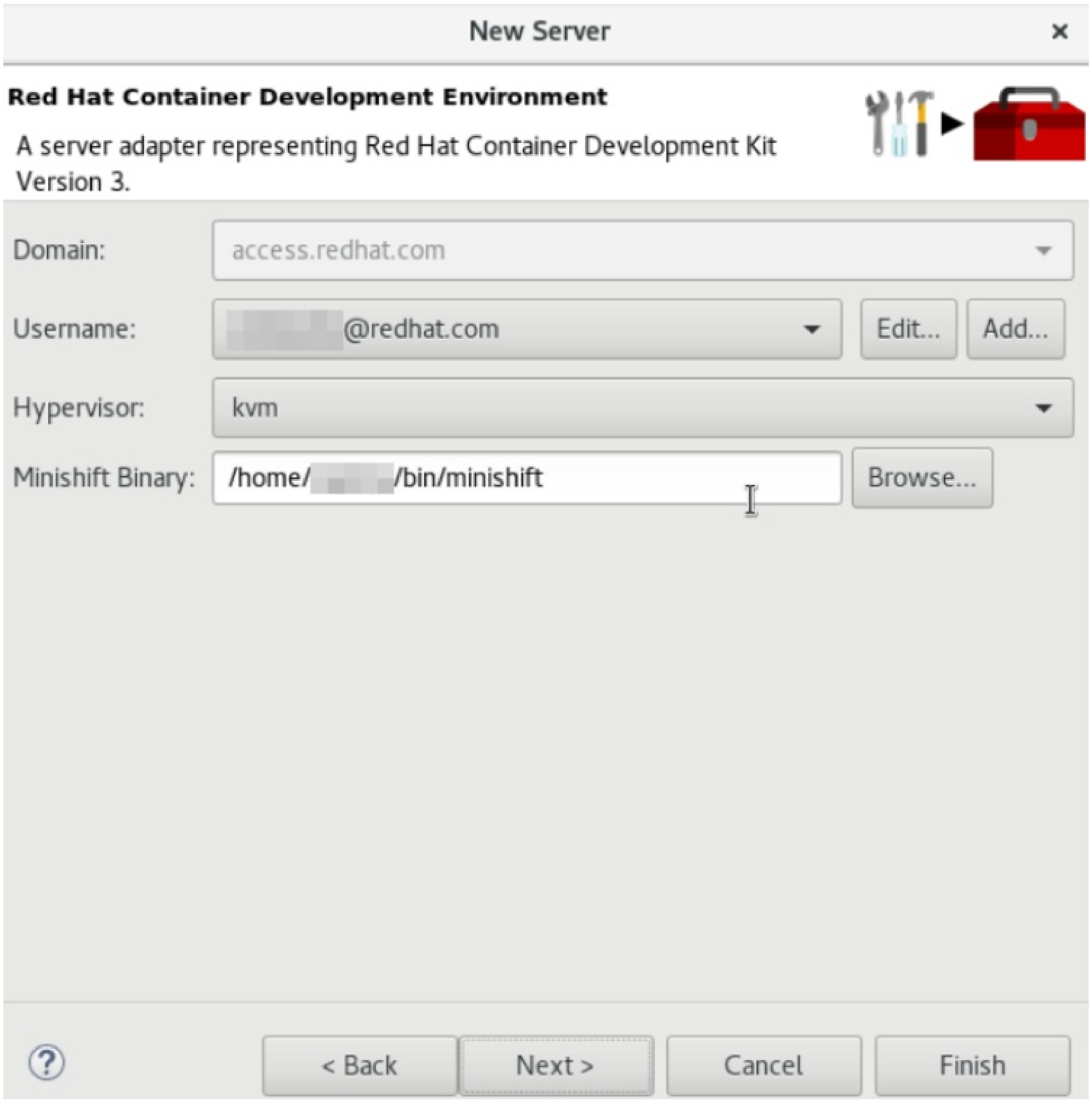
   a. In the **Domain** field, ensure that **access.redhat.com** appears.

   b. In the **Username** field, ensure that your username appears.

   c. In the **Hypervisor** field, ensure that the relevant virtualization system for your operating system appears.

   d. In the **Minishift Binary** field, ensure that the location of the directory where minishift is installed appears. . Click **Finish**.

Figure 1.2. Entering User Credentials



New Server

Red Hat Container Development Environment

A server adapter representing Red Hat Container Development Kit Version 3.

Domain: access.redhat.com

Username: [redacted]@redhat.com

Hypervisor: kvm

Minishift Binary: /home/[redacted]/bin/minishift

8. If it appears, in the **Secure Storage Password** window, **Password** field, type your password and click **OK**.

9. Open the **Servers** view, and right-click the Container Development Environment server adapter and click **Start**.

10. The **Console** view is the view in focus showing the progress of starting the Container Development Environment.

11. If it appears, in the **Untrusted SSL Certificate** dialog box, click **Yes**.

Result: The **Starting local OpenSHift cluster using 'kvm' hypervisor** message appears in the **Console** view of the IDE.

In case you are not signed in to your OpenShift Container Platform account, the **New OpenShift Connection** wizard appears. Either create a new connection or sign in to your existing account using the **Basic** or **OAuth** protocol.

0pen the **OpenShift Explorer** view to see 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 connection and expand the connection to see the Containers and Images. Choose to continue working with OpenShift Container Platform within DevStudio or view instructions for Container-based Development with DevStudio.

1.1.4. Using the Docker Tooling

After starting the CDK server in the IDE, you can follow one of the two container development workflows:

1. Use Docker for Container-based Development
2. Build Docker Image Using the Container Development Environment

1.1.4.1. Use Docker for Container-based Development

Use Docker for Container-based Development as follows:

1. Create a new project with your Dockerfile. .. Click **File** > **New** > **Project**.
    a. Type *java* in the search field and from the results, select **Java Project** and click **Next** to continue.
    b. 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.
    c. Click **File** > **New** > **File**.
    d. 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**.
    e. 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.

1.1.4.2. Build Docker Image Using the Container Development Environment

To do a Docker image build using the Container Development Environment:

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.

1.1.4.2.1. Next Steps for the Docker Tooling

For further information about the basics of Docker Tooling, see Configure Docker Tooling (Basic).

1.1.5. Using OpenShift Container Platform Tooling

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, rfer to Creating an OpenShift Container Platform Application in Red Hat JBoss Developer Studio.

1.1.5.1. 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.

1.1.6. Known Issues

- When the **Docker Explorer** view is first started, attempting to extend the Containers or Images causes the explorer to fail and throw an exception. To work around this issue, restart Eclipse/JBoss Developer Studio. For details, see JBIDE-21983.

CHAPTER 2. DEVELOPING FOR THE CLOUD WITH OPENSHIFT 3

2.1. CREATING AN OPENSHIFT CONTAINER PLATFORM APPLICATION IN RED HAT JBOSS DEVELOPER STUDIO

Using the OpenShift Container Platform Tooling you can create, import, and modify OpenShift Container Platform applications by:

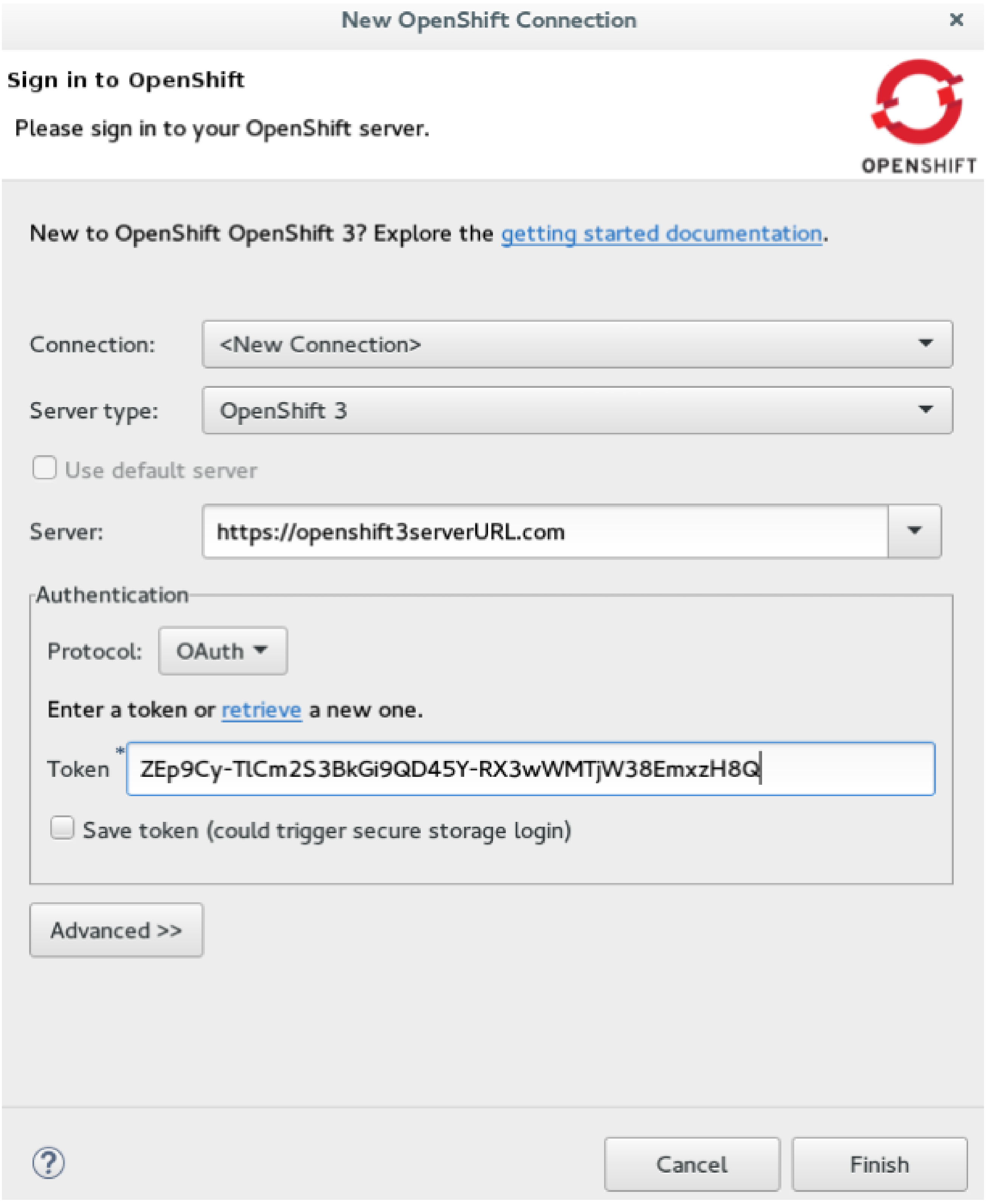
1. Creating a New OpenShift Container Platform Connection
2. Creating a New OpenShift Container Platform Project
3. Creating a New OpenShift Container Platform Application
4. Importing an Existing OpenShift Container Platform Application into the IDE
5. Deploying an Application Using the Server Adapter
6. Viewing an Existing Application in a Web Browser
7. Deleting an OpenShift Container Platform Project

2.1.1. Creating a New OpenShift Container Platform Connection

To use the OpenShift Container Platform tooling in the IDE, you must first create an OpenShift Container Platform connection. To create a new connection:

1. In the **OpenShift Explorer** view, click **New Connection Wizard**. If the **OpenShift Explorer** view is not available, click **Window** > **Show View** > **Other** and then search for **OpenShift Explorer** and after you find it, click **OK**.
2. In the **New OpenShift Connection** wizard:
    a. In the **Connection** list, click **<New Connection>**.
    b. In the **Server type** list, click **OpenShift 3**.
    c. In the **Server** field, type the URL for an OpenShift Container Platform server.
    d. In the **Authentication** section, in the **Protocol** list, click **OAuth** to authenticate using the token or click **Basic** to authenticate using login credentials.
3. 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.

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)

Result: The connection is listed in the **OpenShift Explorer** view.

2.1.2. Creating a New OpenShift Container Platform Project

To create a new OpenShift Container Platform project:

1. In the **OpenShift Explorer** view, right-click the connection and click **New** > **Project**.

2. In the **Create OpenShift Project** window:

   a. 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.

   b. 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.

   c. In the **Description** field, type a description of the project.

3. Click **Finish**.

Result: 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 to create OpenShift Container Platform applications from default or custom templates. Using a template to create an application is useful because the same template can be used to create multiple similar applications with different or identical configurations for each of them.

NOTE

To learn more about using and creating templates with OpenShift Container Platform, see Templates.

To create a new OpenShift Container Platform application:

1. In the **OpenShift Explorer** view, 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.

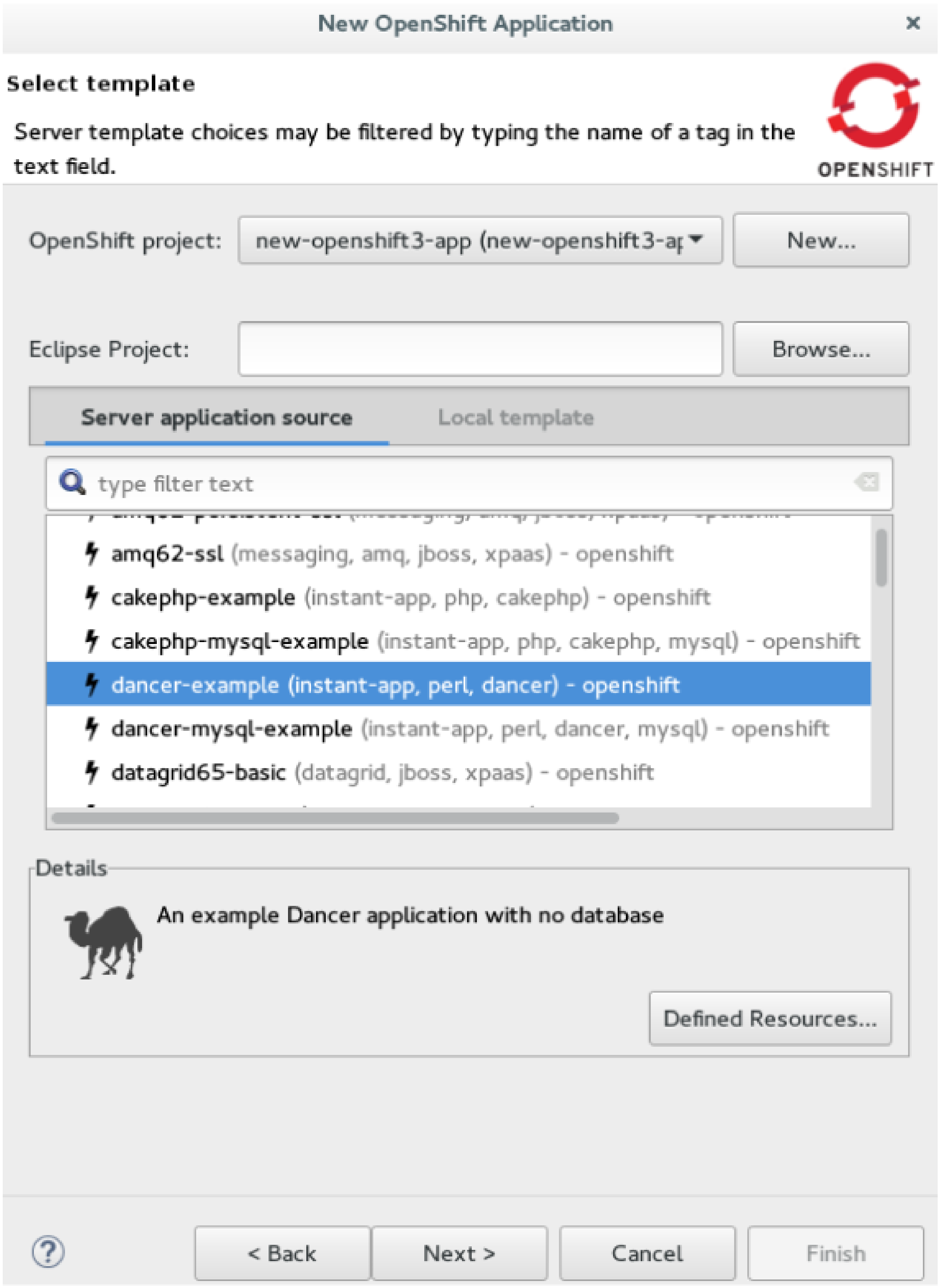
   NOTE

   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



Server template choices may be filtered by typing the name of a tag in the text field.

OpenShift project: new-openshift3-app (new-openshift3-ap

Eclipse Project:

amq62-ssl (messaging, amq, jboss, xpaas) - openshift
cakephp-example (instant-app, php, cakephp) - openshift
cakephp-mysql-example (instant-app, php, cakephp, mysql) - openshift
dancer-example (instant-app, perl, dancer) - openshift
dancer-mysql-example (instant-app, perl, dancer, mysql) - openshift
datagrid65-basic (datagrid, jboss, xpaas) - openshift

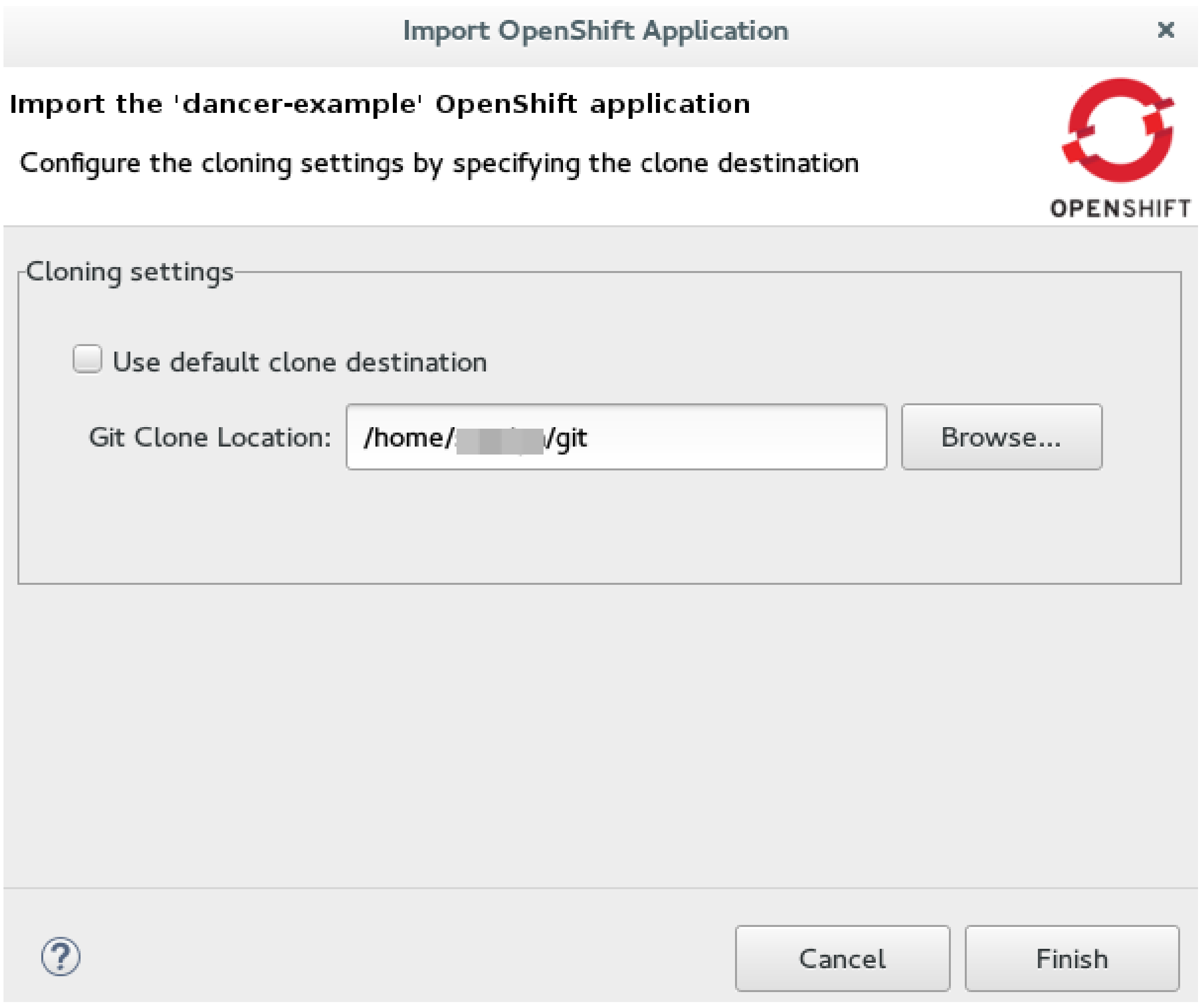
An example Dancer application with no database

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**.

9. In the **Results of creating the resources from the *{template_name}*** window, review the details and click **OK**.

10. 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



Import the 'dancer-example' OpenShift application

Configure the cloning settings by specifying the clone destination

Cloning settings

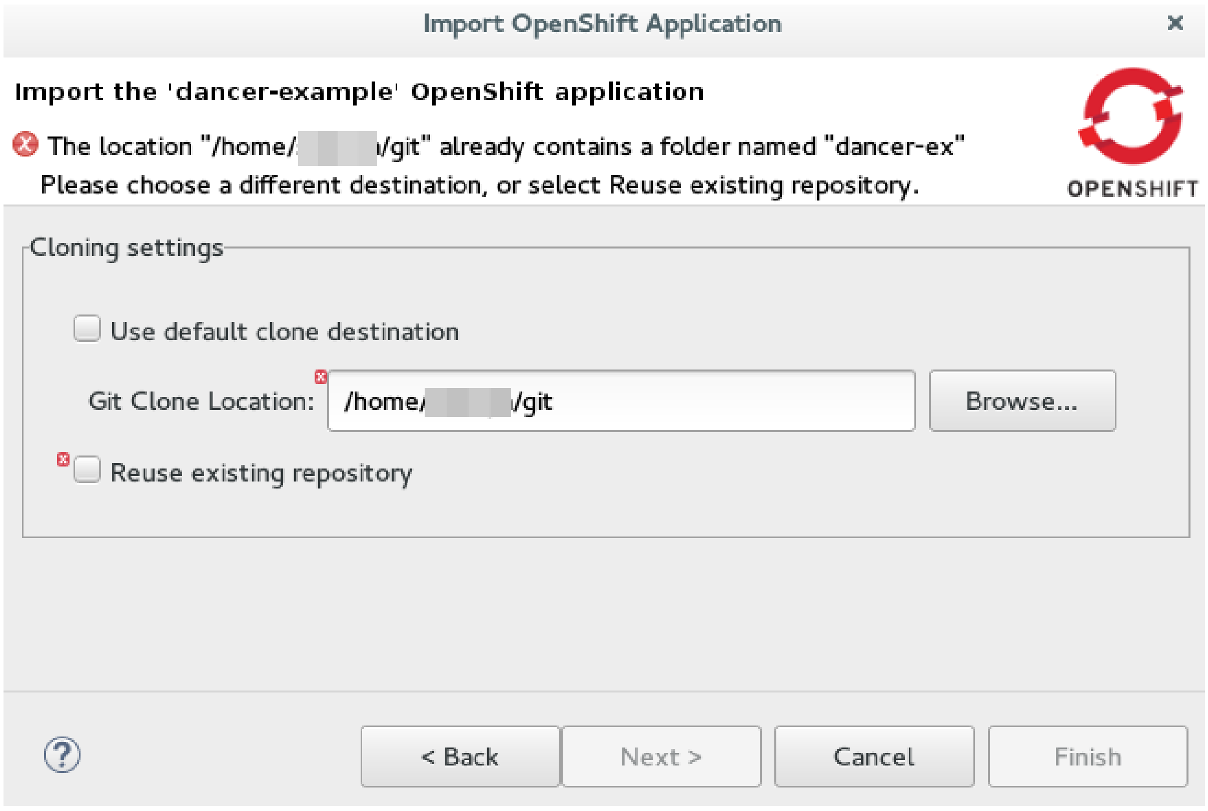
☐ Use default clone destination

Git Clone Location: /home/[redacted]/git

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



Import OpenShift Application

Import the 'dancer-example' OpenShift application

The location "/home/[redacted]/git" already contains a folder named "dancer-ex"
Please choose a different destination, or select Reuse existing repository.

Cloning settings

- [ ] Use default clone destination

Git Clone Location: /home/[redacted]/git

- [ ] Reuse existing repository

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

2.1.4. Importing an Existing OpenShift Container Platform Application into the IDE

NOTE

Only an application that has its source specified in the `build config` file can be imported into the workspace.

Applications associated with your OpenShift Container Platform account(s) are listed in the **OpenShift Explorer** view. The source code for these applications can be individually imported into the IDE using the **Import OpenShift Application** wizard. Once imported, the user can easily modify the application source code, as required, build the application, and view it in a web browser.

To import an existing OpenShift Container Platform application as a new project in the existing IDE workspace:

1. If required, sign into your OpenShift Container Platform server using the **Basic** protocol or the **OAuth** protocol.

2. In the **OpenShift Explorer** view, expand the connection to locate the application to import.

3. Right-click the *{project name}* and click **Import Application**.

NOTE

To import a particular application from a service, right-click the service and then click **Import Application**. If you right-click a project and click **Import Application**, and if there are more than one build configurations with source code under a project, you will be prompted to select the desired application for import because of existence of several applications under one project.

4. In the **Import OpenShift Application** wizard, **Existing Build Configs** list, click the application that you want to import and click **Next**.

5. Ensure the location in the **Git Clone Destination** field corresponds to where you want to make a local copy of the application Git repository and click **Finish**.

Result: The application is listed in the **Project Explorer** view.

2.1.5. Deploying an Application Using the Server Adapter

The server adapter allows incremental deployment of applications directly into the deployed pods on OpenShift Container Platform.

To deploy an application:

1. In the **OpenShift Explorer** view, 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.

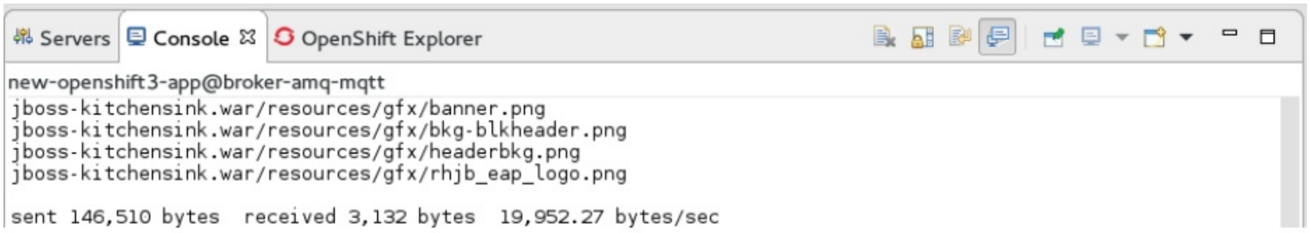
NOTE

A workspace project will be selected automatically, if the OpenShift service has a Build Configuration with a git URL matching the git remote URL of one of the workspace projects.

4. Click **Finish**.

Result: 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.

Figure 2.5. Console View Showing Application Publication Progress



```
new-openshift3-app@broker-amq-mqtt
jboss-kitchensink.war/resources/gfx/banner.png
jboss-kitchensink.war/resources/gfx/bkg-blkheader.png
jboss-kitchensink.war/resources/gfx/headerbkg.png
jboss-kitchensink.war/resources/gfx/rhjb_eap_logo.png

sent 146,510 bytes  received 3,132 bytes  19,952.27 bytes/sec
```

2.1.6. Viewing an Existing Application in a Web Browser

After an application has been successfully deployed, to view it in the internal web browser, in the **OpenShift Explorer** view, right-click the application, and click **Show In** > **Web browser**.

Result: The application displays in the built-in web browser.

2.1.7. 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. All resources associated with a project get deleted when the project is deleted.

To delete a project:

1. In the **OpenShift Explorer** view, expand the connection and then the project to locate the application 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**.

2.1.8. Did You Know

- Scale the project deployment, using the context menu for the service (the first node below the project). You can also scale the deployment from the **Properties** tab of a deployment (replication controller) and deploymentconfig.
- View the rsync output in the **Console** view. You can also see the progress of the file transfer after you publish local changes to OpenShift Container Platform.

2.2. SETTING UP AND REMOTELY MONITORING AN OPENSHIFT 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. Connect to and work with a remote OpenShift Container Platform instance by:

1. Setting up OpenShift Client Binaries
2. Setting up Port Forwarding
3. Streaming Pod Logs
4. Streaming Build Logs

2.2.1. Setting up OpenShift Client Binaries

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

To set up the OpenShift Client Binaries:

1. In the IDE, click **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 **Downloads** section and 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. Click **Window** > **Preferences** > **JBoss Tools** > **OpenShift 3**.

7. Click **Browse** and select the location of the OpenShift Client executable file.

8. Click **Apply and Close**.

Result: 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.

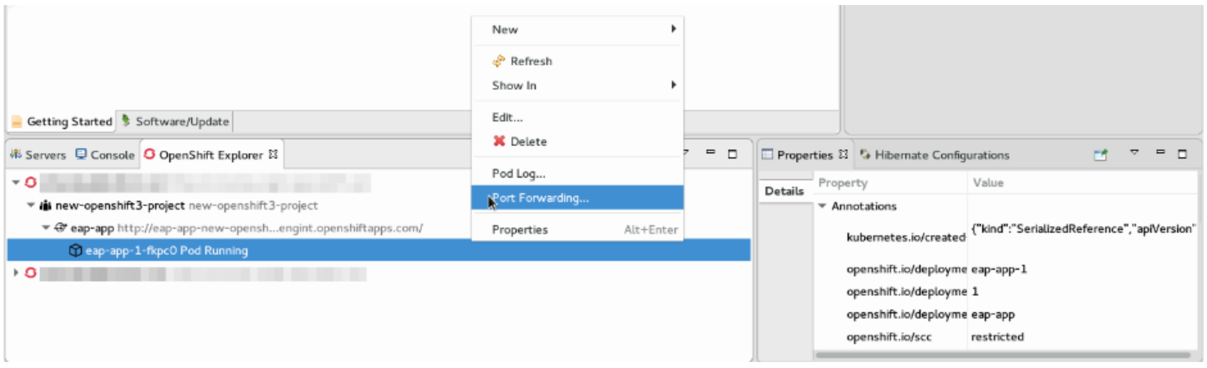
Prerequisite: Ensure that the OpenShift Client Binaries are set up (see Setting up OpenShift Client Binaries for instructions).

To set up port forwarding:

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

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

Figure 2.6. Setting up Port Forwarding



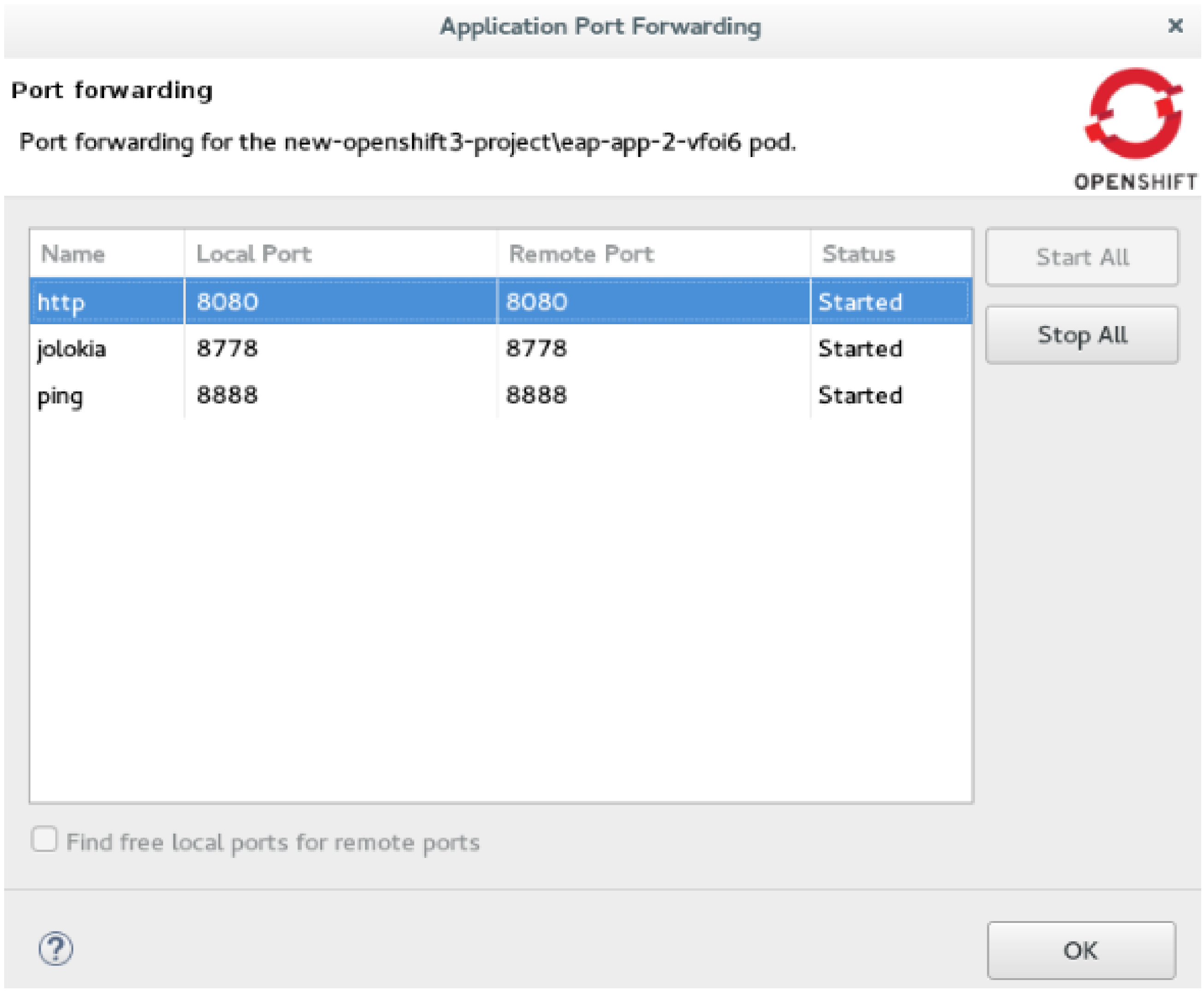
[redacted]
new-openshift3-project new-openshift3-project
eap-app http://eap-app-new-opensh...engint.openshiftapps.com/
eap-app-1-fkpc0 Pod Running
[redacted]

Property	Value
Annotations	
kubernetes.io/created	{"kind":"SerializedReference","apiVersion"
openshift.io/deployme	eap-app-1
openshift.io/deployme	1
openshift.io/deployme	eap-app
openshift.io/scc	restricted

3. In the **Application Port Forwarding** window, click the **Find free local ports for remote ports** check box and then click **Start All**.

Result: 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 2.7. Start Port Forwarding



Port forwarding

Port forwarding for the new-openshift3-project\eap-app-2-vfoi6 pod.

Name	Local Port	Remote Port	Status
http	8080	8080	Started
jolokia	8778	8778	Started
ping	8888	8888	Started

☐ Find free local ports for remote ports

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.

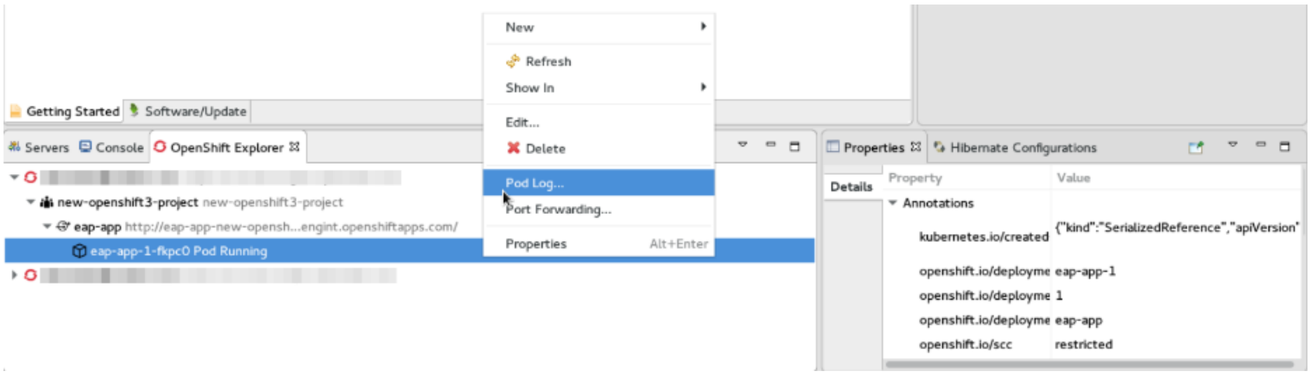
Prerequisite: Ensure that the OpenShift Client Binaries are set up (see Setting up OpenShift Client Binaries for instructions).

To stream the application logs:

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 2.8. Streaming Pod Log



Property	Value
Annotations	
kubernetes.io/created	{"kind":"SerializedReference","apiVersion"
openshift.io/deployme	eap-app-1
openshift.io/deployme	1
openshift.io/deployme	eap-app
openshift.io/scc	restricted

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

2.2.4. Streaming Build Logs

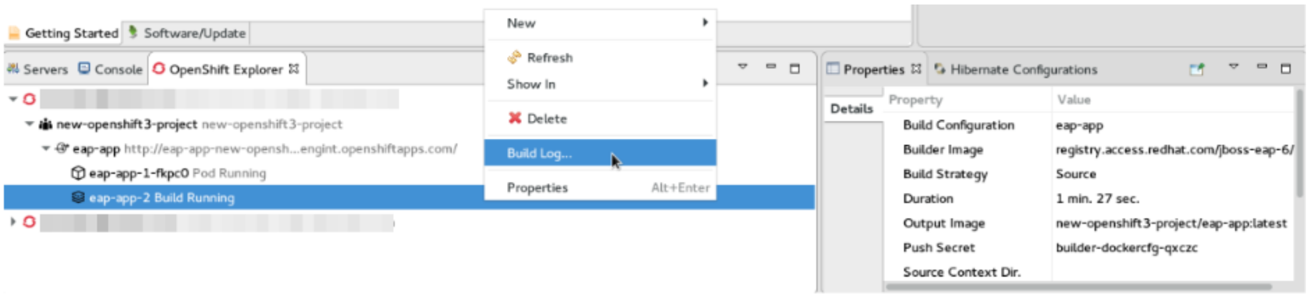
Build logs are logs that document changes to applications running on a remote OpenShift Container Platform instance. The streaming build logs feature in the IDE is used to view the progress of the application build process and to debug the application.

Prerequisite: Ensure that the OpenShift Client Binaries are set up (see Setting up OpenShift Client Binaries for instructions).

To stream build logs:

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 2.9. Streaming Build Log



Property	Value
Build Configuration	eap-app
Builder Image	registry.access.redhat.com/jboss-eap-6/
Build Strategy	Source
Duration	1 min. 27 sec.
Output Image	new-openshift3-project/eap-app:latest
Push Secret	builder-dockercfg-qxczc
Source Context Dir.	

Result: 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 JBoss Developer Studio 10. 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 JBoss Developer Studio. **frontend** and **bonjour** microservices, used here, are examples of such private images that are not present in Docker Hub.

You can build and deploy a Docker-formatted Container Image to Container Development Kit OpenShift Registry by:

1. Section 2.3.2, “Installing the javascript Modules”
2. Section 2.3.3, “Building the frontend Microservice”
   a. Section 2.3.3.1, “Deploying the frontend Microservice”
3. Section 2.3.4, “Connecting the frontend and bonjour Microservices”
   a. Section 2.3.4.1, “Deploying the bonjour Microservice”
   b. Section 2.3.4.2, “Scalling the Pod”
4. Section 2.3.5, “Editing the bonjour Microservice”
   a. Section 2.3.5.1, “Viewing the Edited bonjour Microservice on the frontend Microservice”

2.3.1. Prerequisites

1. Install nmp: Before running JBoss 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 JBoss Developer Studio and Red Hat Container Development Kit.
   a. On a Windows system: Install Red Hat Development Suite to automatically install both: JBoss Developer Studio and Red Hat Container Development Kit (for installation instructions, see https://access.redhat.com/documentation/en/red-hat-development-suite/1.1/paged/installation-guide/).
   b. On other operating systems: Install JBoss Developer Studio (for installation instructions, see: https://access.redhat.com/documentation/en/red-hat-jboss-developer-studio/10.1/paged/installation-guide/) and install Red Hat Container Development Kit (for installation instructions, see https://access.redhat.com/documentation/en/red-hat-container-development-kit/2.2/paged/installation-guide/).
4. Clone the following projects and then import them into JBoss 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**.

2.3.2. Installing the javascript Modules

To download and install all the required javascript modules:

1. In the **Project Explorer** view, expand **frontend** and right-click `package.json`.

2. Click **Run As** > **npm Install** to download and install the required javascript modules in the project.

Result: After the build is complete, a new **`node_modules`** folder is listed under the project in the **Project Explorer** view.

2.3.3. Building the frontend Microservice

In this section we build the **frontend** microservice which is the landing page for the application being built. The **frontend** microservice calls other microservices (**bonjour**, in this case) and displays the results from these calls.

To build the Docker-formatted Container image:

1. In the **Project Explorer** view, expand **frontend** and right-click **Dockerfile** and then click **Run As** > **Docker Image Build**.
2. 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**.

Result: The Docker-formatted Container image starts building against the Docker Daemon running in the Container Development Environment.

2.3.3.1. Deploying the frontend Microservice

After the build is complete, the Docker-formatted Container image **demo/frontend** is available in the **Docker Explorer** under **Container Development Environment**.

To deploy the frontend microservice:

1. In the **Docker Explorer** view, **Container Development Environment** > **Images**, right-click **demo/frontend** and click **Deploy to OpenShift**.
2. In the **Deploy an Image** window, click **New**.
3. In the **Create OpenShift Project** window:
    a. In the **Project Name** field, type the name of the new project, **demo**.
    b. Optionally, in the **Display Name** and **Description** fields, enter the required details.
    c. Click **OK**.
4. In the **Deploy an Image** window, click the **Push Image to Registry** check box and click **Next**.
5. In the **Deployment Configuration & Scalability** window, change the following environment variables:
    a. 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 and right-click the service and click **Show In** > **Web Browser**.

Result: The **frontend** microservice, in the Bonjour Service shows: **Error getting value from service <microservice>** meaning the **bonjour** microservice must be connected.

2.3.4. Connecting the frontend and bonjour Microservices

In this section we build the **bonjour** microservice and then view it on the **frontend** microservice. The **bonjour** microservice is a simple **node.js** application that returns the string **bonjour-de-<pod_ID>**.

To connect the Microservices:

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. In the **Docker Image Build Configuration** window:

   a. In the **Connectio*n list, select *Container Development Environment**.

   b. In the **Repository Name** field, type **demo/bonjour**.

6. Click **OK**.

2.3.4.1. Deploying the bonjour Microservice

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

1. In the **OpenShift Explorer** view, right-click the project (**demo**), and click **Deploy Docker Image**.

2. In 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**.

2.3.4.2. Scalling the Pod

To see the **bonjour** service with the Pod running:

1. In the **OpenShift Explorer** view, 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 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.

Result: Navigate to the browser and click **Refresh Results** to see the service balancing between the two Pods.

2.3.5. Editing the bonjour Microservice

In this section we edit the **bonjour** microservice and then view the results on the **frontend** microservice.

To edit the **bonjour** microservice:

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

2. Find

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

3. Change it to:

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

4. Save the file.

2.3.5.1. Viewing the Edited bonjour Microservice on the frontend Microservice

After you have edited the **bonjour** microservice:

1. In the **Project Explorer** view, expand **bonjour**, and right-click **Dockerfile**.

2. Click **Run As** > **Docker Image Build**.

> **NOTE**
>
> Here, the Docker run configuration, the connection, and the repository name used earlier are being reused. To edit the configuration, open the **Run Configuration** window.

After the **Console** view shows that the Docker-formatted Container image has been successfully pushed to the Docker Daemon:

3. In the **Docker Explorer** view, expand **Container Development Environment** > **Images**.
4. Right-click the image and click **Deploy to OpenShift**.
5. In the **Deploy an Image** window, click **Push Image to Registry** and then click **Next**.
6. In the **Deployment Configuration & Scalability** window, click **Finish**. The **OpenShift Explorer** view, under **bonjour** shows the Pods being added and then running. Navigate to the browser and click **Refresh Results**.

Result: The new greeting appears.

2.3.6. Troubleshooting

2.3.6.1. No Docker Connection Available

Error message: No Docker Connection available to build the image.

Issue: You have installed JBoss Developer Studio through Red Hat Development Suite and you must start Red Hat Container Development Kit for it to be available. **Resolution**:

1. In the **Servers** view, right-click **Container Development Environment** and click **Start**.
2. Enter your credentials in the box provided.

If, after doing this the Container Development Environment does not start and you get the following error: **Error message**: Server Container Development Environment failed to start.

On the command prompt, cd to `cdk/components/rhel/rhel-ose` and run the **vagrant destroy** command. After it is destroyed, run the **vagrant up** command. In the IDE, in the **Servers** view, right-click **Container Development Environment** and click **Start** once again.

CHAPTER 3. DEPLOYING THE OPENSHIFT 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.

Prerequisite

- 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/.

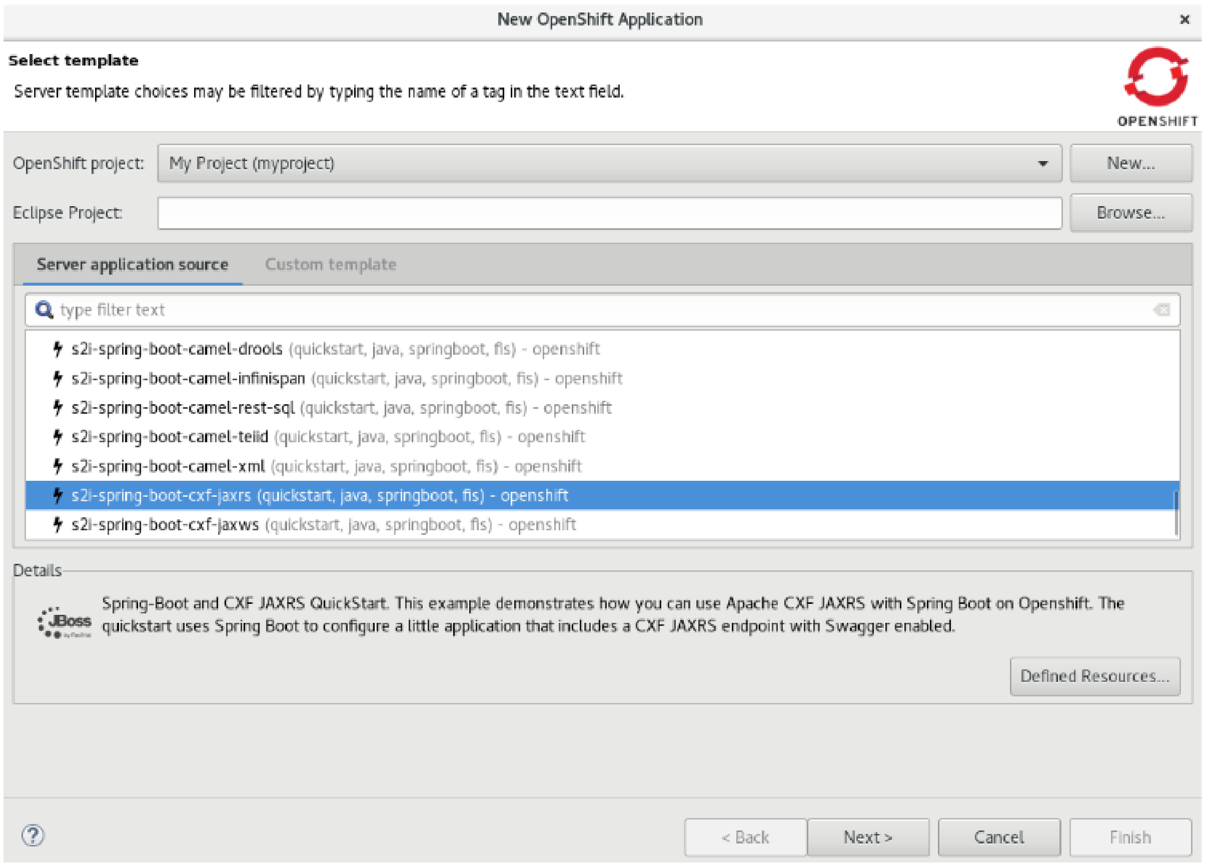
3.1. DEPLOYING THE S2I-SPRING-BOOT-CFX-JAXRS TEMPLATE

Set up the IDE to work with CDK 3 as described in Using Container Development Kit 3.x Tooling in JBoss Developer Studio 11.x. 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.

To deploy the template:

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 3.1. Selecting the s2i-spring-boot-cfx-jaxrs Template


New OpenShift Application

Select template

Server template choices may be filtered by typing the name of a tag in the text field.

OpenShift project: My Project (myproject)

Eclipse Project:

Server application source | Custom template

type filter text

- s2i-spring-boot-camel-drools (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-camel-infinispan (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-camel-rest-sql (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-camel-teiid (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-camel-xml (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-cxf-jaxrs (quickstart, java, springboot, fis) - openshift
- s2i-spring-boot-cxf-jaxws (quickstart, java, springboot, fis) - openshift

Details

Spring-Boot and CXF JAXRS QuickStart. This example demonstrates how you can use Apache CXF JAXRS with Spring Boot on Openshift. The quickstart uses Spring Boot to configure a little application that includes a CXF JAXRS endpoint with Swagger enabled.

6. 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.

Result: The **Console** view shows the **latest: digest: sha256:{checksum} size: 9033 Push successful** message.

3.2. VIEWING THE APPLICATION IN THE WEB CONSOLE

NOTE

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.

To view the application in the Pod:

1. In the web console, click **Applications** > **Pod** and then click the **s2i-spring-boot-cfx-jaxrs-1** pod.

2. Click the **Logs** tab.

3. 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).

4. Click the **Terminal** tab.

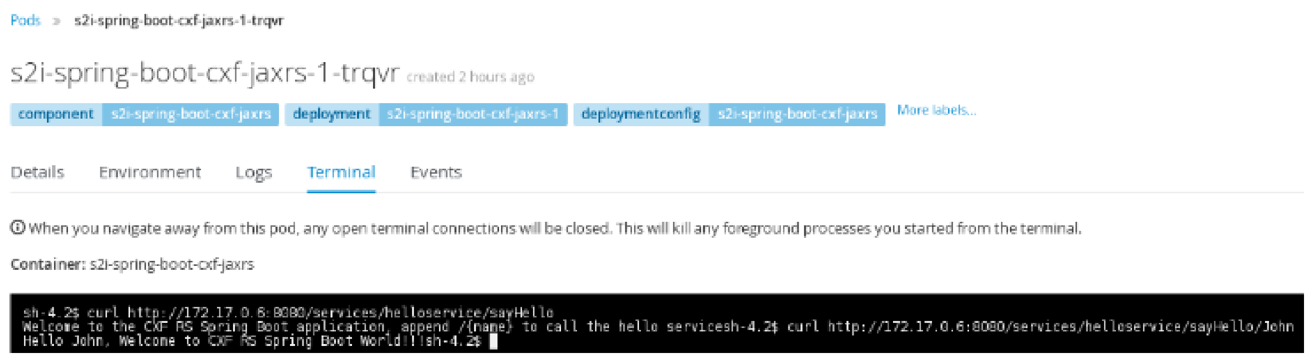
5. In the terminal, type the following command: **`curl http://{IP_address}:8080/services/helloservice/sayHello`**.
   Example: **`curl`** `http://172.17.0.6:8080/services/helloservice/sayHello`

6. Press Enter.

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

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

Figure 3.2. s2i-spring-boot-cfx-jaxrs Application in the Web Console


Pods » s2i-spring-boot-cxf-jaxrs-1-trqvr

s2i-spring-boot-cxf-jaxrs-1-trqvr created 2 hours ago

component s2i-spring-boot-cxf-jaxrs | deployment s2i-spring-boot-cxf-jaxrs-1 | deploymentconfig s2i-spring-boot-cxf-jaxrs | More labels...

Details | Environment | Logs | Terminal | Events

When you navigate away from this pod, any open terminal connections will be closed. This will kill any foreground processes you started from the terminal.

Container: s2i-spring-boot-cxf-jaxrs

```
sh-4.2$ curl http://172.17.0.6:8080/services/helloservice/sayHello
Welcome to the CXF RS Spring Boot application, append /{name} to call the hello servicesh-4.2$ curl http://172.17.0.6:8080/services/helloservice/sayHello/John
Hello John, Welcome to CXF RS Spring Boot World!!!sh-4.2$ 
```

3.3. DEFINING SERVICES AND ROUTES USING A JSON FILE

Use the **`services-route.json`** file to create the service for the **s2i-spring-boot-cfx-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.

To define the resources:

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**.

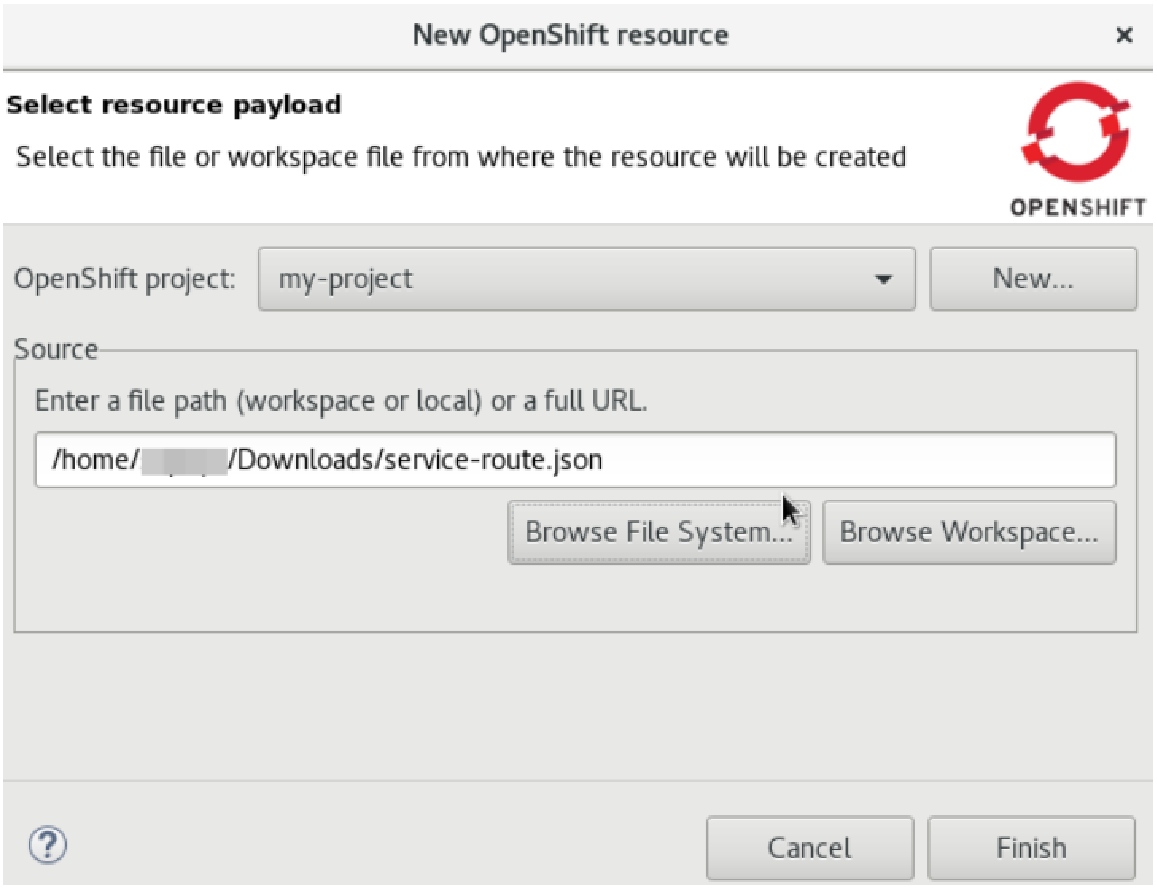
3. In the **New OpenShift Resource** window:

   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 3.3. Selecting the service-routes.json File


New OpenShift resource

Select resource payload

Select the file or workspace file from where the resource will be created

OpenShift project: my-project

Source

Enter a file path (workspace or local) or a full URL.

/home/[redacted]/Downloads/service-route.json

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.

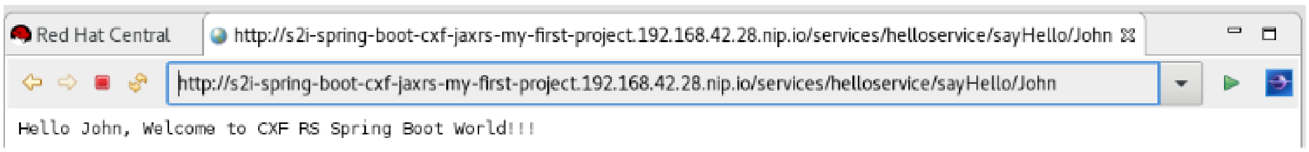
6. In the address bar, append the URL with `services/helloservice/sayHello/`. The URL should now look like: **http://s2i-spring-boot-cxf-jaxrs-**.*{IP_address}.nip.io/services/helloservice/sayHello/.*

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/helloservice/sayHello/John****.*

Result: *The page displays the message:* ***Hello John, Welcome to CXF RS Spring Boot World!!!***

Figure 3.4. Viewing the s2i-spring-boot-cxf-jaxrs Application in the Web Browser



http://s2i-spring-boot-cxf-jaxrs-my-first-project.192.168.42.28.nip.io/services/helloservice/sayHello/John

```
Hello John, Welcome to CXF RS Spring Boot World!!!
```

CHAPTER 4. DEVELOPING FOR THE CLOUD WITH OPENSHIFT 2

IMPORTANT

Starting with JBoss Developer Studio 11.0, development with OpenShift 2 is no longer supported.

CHAPTER 5. DEVELOPING WITH DOCKER

5.1. CONFIGURING DOCKER TOOLING BASICS

Docker offers images that can be managed in one of two ways: build/create your own image using a script file, or pull down an existing image file from public or private registries online. This procedure contains instructions for pulling down an image from the JBoss registry. Such registries are useful to share images between developers or between environments to ensure a standardized software stack for development or testing requirements. Once the Docker Container is created and running, users can manage the container.

JBoss Developer Studio 9.0. 0 Beta 1 includes the Docker tooling out of the box. This article introduces the basic usage for the Docker tooling, such as:

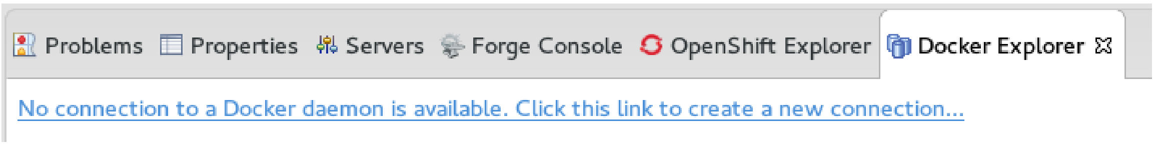
1. *Prerequisites for Docker Tooling*
2. *Pulling Docker Images from a Docker Registry*
3. *Managing Docker Containers*

5.1.1. Prerequisites for Docker Tooling

Ensure that the following prerequisites are met when using the Docker tooling with the IDE:

1. *Install and set up JBoss Developer Studio 9.0.0 Beta 1 or higher.*
2. *Install and set up Docker.*
3. *Establish a connection to a Docker daemon from within the IDE:*
   a. *Click **Window** > **Show View** > **Other**.*
   b. *In the **Show View** window, type docker in the filter text box to view Docker-related options in the list.*
   c. *Expand **Docker**, click **Docker Explorer**, and then click **Open**.*
   d. *In the **Docker Explorer** view, if no connection is configured, a message appears stating that **No connection to a Docker daemon is available. Click this link to create a new connection**. Click the link to start configuring a new Docker daemon connection.*

      Figure 5.1. Creating a new Docker Daemon Connection

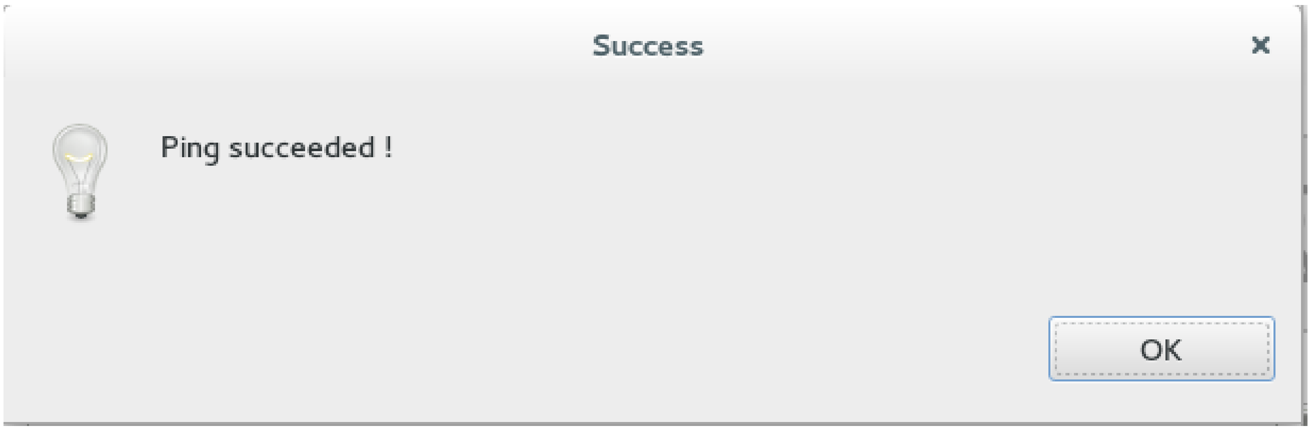
      No connection to a Docker daemon is available. Click this link to create a new connection...

   e. *In the **Connect to a Docker daemon** wizard:*
      i. *In the **Connection Name** field, use the default value or set a desired connection name.*
      ii. *Click the **Use custom connection settings** check box.*
      iii. *Add the relevant socket information for your host. If you are unsure about this step, use the default Unix socket Location value.*

iv. *Click **Test Connection** to test the connection.*

***Result:** When a connection is successfully established, the **Ping Succeeded** message appears.*

Figure 5.2. Ping Success Message


Success

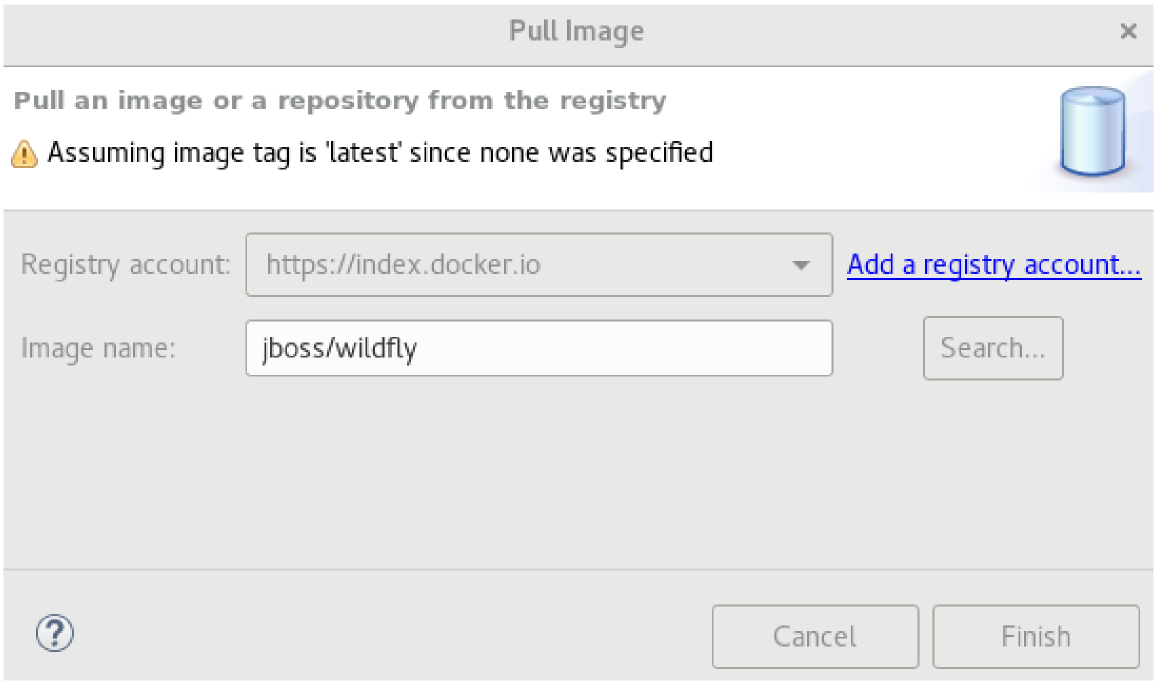
Ping succeeded !

5.1.2. Pulling Docker Images from a Docker Registry

To pull a new WildFly Docker Image from the JBoss registry:

1. *Click **Window** > **Show View** > **Other**.*
2. *Type Docker in the filter text field to view Docker-related options in the list.*
3. *Expand **Docker** and double-click **Docker Images**.*
4. *In the **Docker Images** view, locate the entry that ends with **wildfly:latest**. If this entry is not listed, pull the Wildfly image as follows:*
   a. *In the **Docker Images** view, click the **Pull** icon.*
   b. *In the **Pull Image** wizard, **Image name** field, type jboss/wildfly and click **Finish**.*

   Figure 5.3. Pulling the WildFly Image from JBoss Registry


Pull Image

Pull an image or a repository from the registry

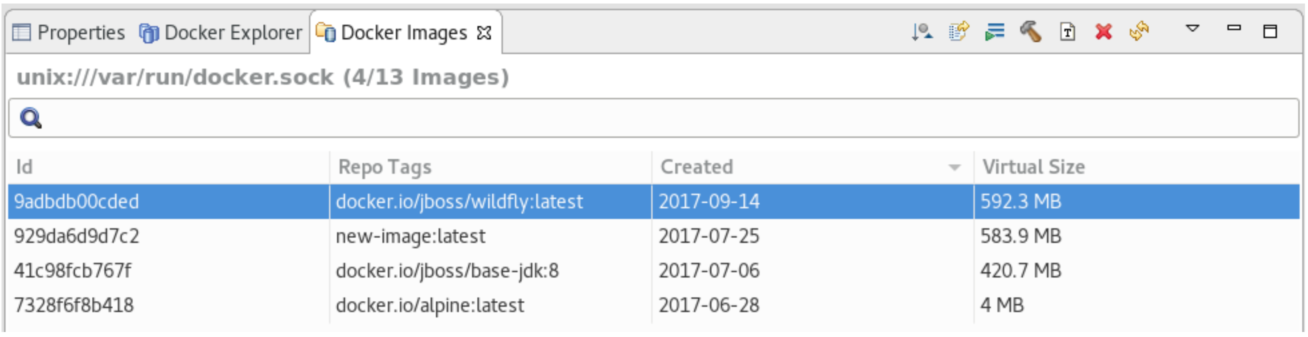
Assuming image tag is 'latest' since none was specified

Registry account: https://index.docker.io

Image name: jboss/wildfly

Result: *The bottom right corner displays the progress as the image is pulled down from the registry. When the pulling process completes, the appropriate entry appears in the Docker Images list.*

Figure 5.4. The Docker WildFly Image



unix:///var/run/docker.sock (4/13 Images)

Id	Repo Tags	Created	Virtual Size
9adbdb00cded	docker.io/jboss/wildfly:latest	2017-09-14	592.3 MB
929da6d9d7c2	new-image:latest	2017-07-25	583.9 MB
41c98fcb767f	docker.io/jboss/base-jdk:8	2017-07-06	420.7 MB
7328f6f8b418	docker.io/alpine:latest	2017-06-28	4 MB

5.1.3. 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:

1. *Click **Window** > **Show View** > **Other**.*
2. *Type Docker in the filter text field 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 the **Pause** button.*
   b. *To start the container, click the **Start** button.*
   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.*

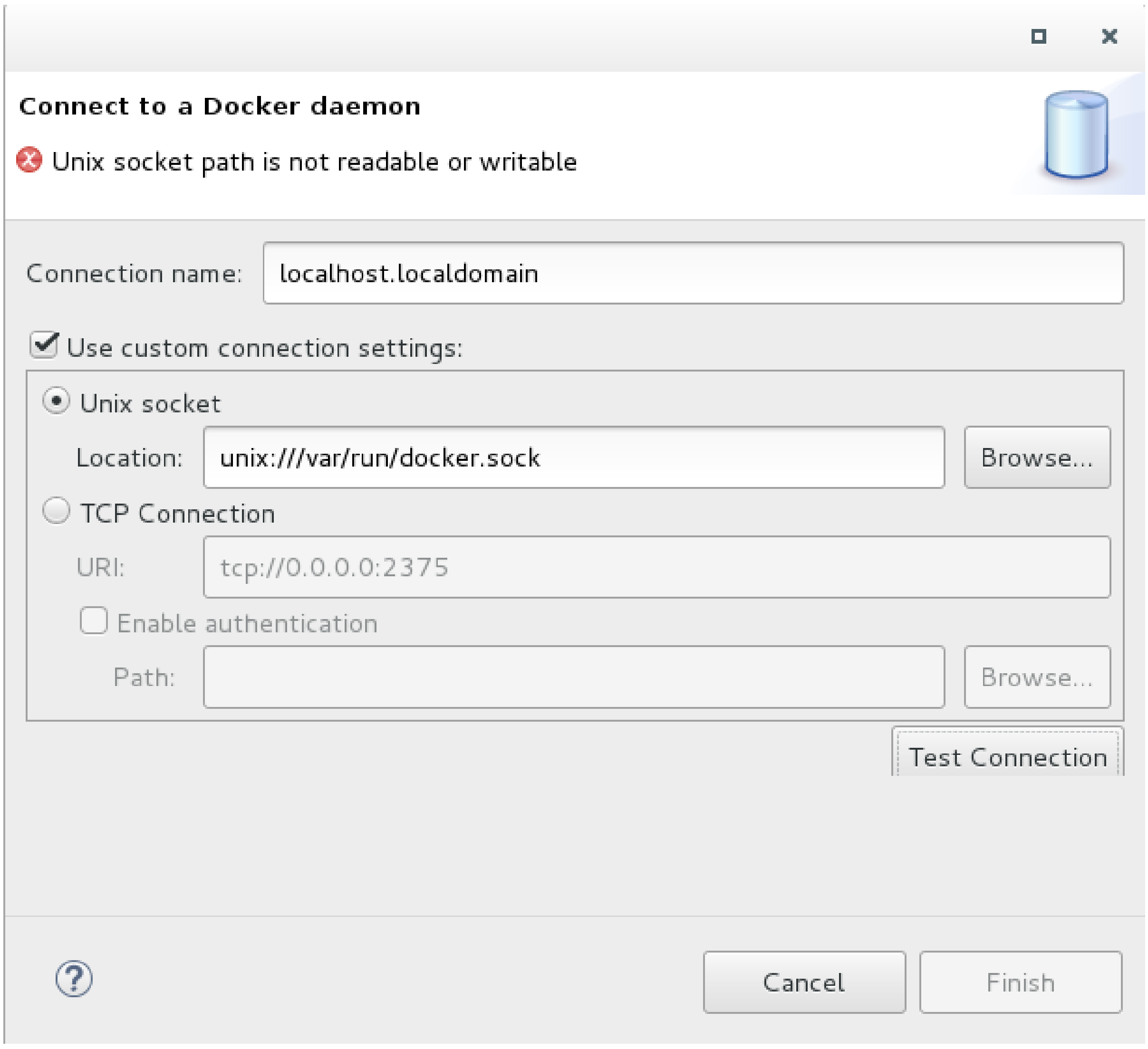
Result: *You have performed various management operations on your Docker container.*

5.1.4. Troubleshooting

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 5.5. Connect to a Docker Daemon



Connect to a Docker daemon

Unix socket path is not readable or writable

Connection name: localhost.localdomain

[✓] Use custom connection settings:

(●) Unix socket

Location: unix:///var/run/docker.sock

( ) TCP Connection

URI: tcp://0.0.0.0:2375

[ ] Enable authentication

Path:

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