Reference Architectures 2017
Build and Deployment of Java Applications on OpenShift Container Platform 3

Calvin Zhu

Reference Architectures 2017 Build and Deployment of Java Applications on OpenShift Container Platform 3

Calvin Zhu
refarch-feedback@redhat.com

Legal Notice

Copyright © 2017 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 reference architecture explores some of the common options like S2I, docker images and CI tools for building and deploying Java EE applications in an OpenShift environment.

Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

CHAPTER 1. EXECUTIVE SUMMARY

This reference architecture explores some of the common options for building and deploying Java EE applications in an OpenShift environment.

This includes the use of S2I (Source-to-Image) as well as deploying custom container images to OpenShift. The S2I process can be use with source code, where the build takes place as part of the OpenShift deployment, as well as to deploy already built Java packages such as WAR files.

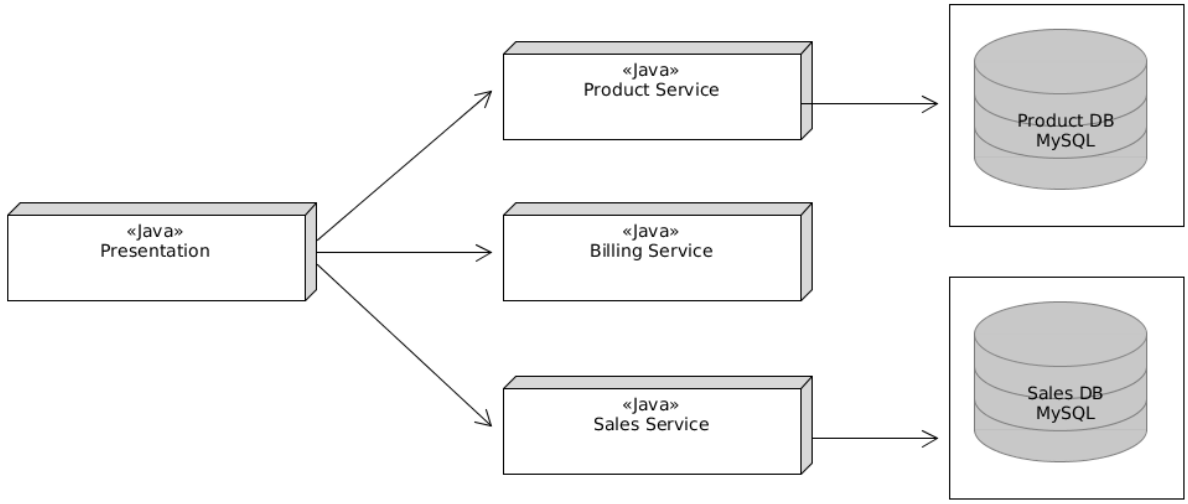
The paper also explores the potential use of continuous integration (CI) tools such as Jenkins, on OpenShift and outside of it, and the use of Maven or any custom build tools to produce an application archive or even a container image before deployment.

CHAPTER 2. REFERENCE ARCHITECTURE ENVIRONMENT

This reference architecture builds and deploys a distributed microservice architecture on top of OpenShift. The application architecture is based on the design laid out and explained in depth in the reference architecture document on EAP Microservice Architecture, implemented on OpenShift in previous reference architecture work.

The starting point for this project includes six services, where two of these are database services based on the standard and supported MySQL image. The remaining four images in the original reference architecture are based on the supported JBoss EAP xPaaS image and rely on the Source to Image (S2I) functionality to build and deploy. The EAP images are customized for two of these services to add a MySQL datasource.

Figure 2.1. Service Diagram, only Java EE implementation



The services in the original implementation are loosely coupled, relying only on REST web service calls to communicate with one another. As a result, any of them can be replaced with a new implementation, as long as the API and behavior is not changed.

All OpenShift services can be configured and scaled with the desired number of replicas. In a default OpenShift installation, the nodes are the only targets for service deployment. However, the master hosts can also be configured as schedulable in order to host service replicas.

The *presentation* service is the only entry point to the application and serves HTML to the client browser over HTTP. This web tier is stateless, and can therefore be replicated without concern for multicast communication or alternative EAP clustering setup. This service relies on the *product*, *sales*, and *billing* services, while the first two in turn rely on the MySQL *product-db* and *sales-db* services.

2.1. OVERVIEW

This reference architecture may be deployed with either a single, or three master hosts. In both cases, it is assumed that *ocp-master1* refers to one (or the only) OpenShift master host and that the environment includes two OpenShift node hosts with the host names of *ocp-node1* and *ocp-node2*.

It is further assumed that OpenShift has been installed by the *root* user and that a regular user has been created with basic access to the host machine, as well as access to OpenShift through its *identity providers*.

2.1.1. OpenShift Login

Log in to the master host as a regular user, and use the *oc* utility to authenticate against OpenShift:

```
$ oc login -u ocuser --certificate-authority=/etc/origin/master/ca.crt
\ --server=https://ocp-master1.hostname.example.com:8443
Authentication required for https://ocp-
master1.hostname.example.com:8443 (openshift)
Username: ocuser
Password: PASSWORD
Login successful.

Welcome! See 'oc help' to get started.
```

2.1.2. Creating a New Project

Create a new project called *deploy-project* for this user:

```
$ oc new-project deploy-project --display-name="deployment on openshift
project" --description="This is a project to demo different deployment
options on openshift"
```

CHAPTER 3. BUILD AND DEPLOY

3.1. SOURCE DEPLOYMENT

3.1.1. Overview

Source-to-Image (S2I) is a framework that makes it easy to write images that take application source code as input, and produce a new image that runs the assembled application as output.

The main reasons one might be interested in using source builds are:

- Speed – with S2I, the assemble process can perform a large number of complex operations without creating a new layer at each step, resulting in a fast process.
- Patchability – S2I allows you to rebuild the application consistently if an underlying image needs a patch due to a security issue.
- User efficiency – S2I prevents developers from performing arbitrary yum install type operations during their application build, which results in slow development iteration.
- Ecosystem – S2I encourages a shared ecosystem of images where you can leverage best practices for your applications.

3.1.2. Usage

To build a Java EE application from source, simply set up a new application and point it to the git repository where the source code is checked in. The OpenShift S2I process detects the presence of a *pom.xml* file in the root of the project and recognizes it as a Java EE application. At this point, the S2I process looks for a default image stream with a *jee* tag to handle the build and deployment. In the absence of such a tagged image stream, or if more than one has a matching tag, the intended image stream would have to be explicitly provided as part of the *new-app* command.

For example, to build and deploy an application with a maven-based source code checked into github, the following command line instruction can be used.

```
$ oc new-app jboss-eap70-openshift~https://github.com/PROJECT.git --
name=APPLICATION_NAME
```

This uses the *oc* client library with the *jboss-eap70-openshift* image stream to build and deploy an application on JBoss EAP 7. It is also possible to set up applications for source deployment, and build and deploy them with Jenkins, using available plugins and/or scripting. Refer to the section on Chapter 4, *Jenkins* for further details.

As an example, source deployment is used to build and deploy the sample microservice application in the following section.

3.1.3. Example

3.1.3.1. MySQL Images

This reference application includes two database services built on the supported MySQL image. This reference architecture uses version 5.6 of the MySQL image.

To deploy the database services, use the *new-app* command and provide a number of required and optional environment variables along with the desired service name.

To deploy the product database service:

```
$ oc new-app -e MYSQL_USER=product -e MYSQL_PASSWORD=password -e
MYSQL_DATABASE=product -e MYSQL_ROOT_PASSWORD=passwd mysql --
name=product-db
```

To deploy the sales database service:

```
$ oc new-app -e MYSQL_USER=sales -e MYSQL_PASSWORD=password -e
MYSQL_DATABASE=sales -e MYSQL_ROOT_PASSWORD=passwd mysql --name=sales-
db
```

Warning

Database images created with this simple command are ephemeral and result in data loss in the case of a pod restart. Run the image with mounted volumes to enable persistent storage for the database. The data directory where MySQL stores database files is located at */var/lib/mysql/data*.

Note

Refer to the official documentation to configure persistent storage.

Warning

Enabling clustering for database images is currently in Technology Preview and not intended for production use.

These commands create two OpenShift services, each running **MySQL** in its own container. In each case, a MySQL user is created with the value specified by the *MYSQL_USER* attribute and the associated password. The *MYSQL_DATABASE* attribute results in a database being created and set as the default user database.

Make sure the database services are successfully deployed before deploying other services that may depend on them. The service log clearly shows if the database has been successfully deployed. Use *tab* to complete the pod name, for example:

```
$ oc logs product-db-1-3drkp
---> 01:57:48     Processing MySQL configuration files ...
---> 01:57:48     Initializing database ...
---> 01:57:48     Running mysql_install_db ...
...omitted...
```

```
2017-03-17 01:58:01 1 [Note] /opt/rh/rh-
mysql56/root/usr/libexec/mysqld: ready for connections.
Version: '5.6.34'  socket: '/var/lib/mysql/mysql.sock'  port: 3306
MySQL Community Server (GPL)
```

3.1.3.2. JBoss EAP 7 xPaaS Images

The microservice project used in this reference architecture can be cloned from its public repository:

```
git clone https://github.com/RHsyseng/MSA-EAP7-OSE.git
```

The *Billing*, *Product*, *Sales*, and *Presentation* services rely on OpenShift S2I for Java EE applications and use the Red Hat xPaaS EAP Image. You can verify the presence of this image stream in the *openshift* project as the *root* user, but first switch from the *default* to the *openshift* project as the root user:

```
# oc project openshift
Now using project "openshift" on server "https://ocp-
master1.hostname.example.com:8443".
```

Query the configured image streams for the project:

```
# oc get imagestreams
NAME       DOCKER REPO
...omitted...
jboss-eap70-openshift
registry.access.redhat.com/jboss-eap-7/eap70-openshift
latest,1.4,1.3 + 2 more...         3 weeks ago
...omitted...
```

3.1.3.3. Building the Services

The microservice application for this reference architecture is made available in a public git repository at https://github.com/RHsyseng/MSA-EAP7-OSE.git. This includes four distinct services, provided as subdirectories of this repository: *Billing*, *Product*, *Sales*, and *Presentation*.They are all implemented in Java and tested on the JBoss EAP 7 xPaaS Images.

Start by building and deploying the *Billing* service, which has no dependencies on either a database or another service. Switch back to the regular user with the associated project and run:

```
$ oc new-app jboss-eap70-openshift~https://github.com/RHsyseng/MSA-
EAP7-OSE.git --context-dir=Billing --name=billing-service
```

Once again, *oc status* can be used to monitor the progress of the operation. To monitor the build and deployment process more closely, find the running build and follow the build log:

```
$ oc get builds
NAME                TYPE      FROM          STATUS     STARTED
DURATION
billing-service-1   Source    Git           Running    1 seconds ago
1s

$ oc logs -f bc/billing-service
```

Once this service has successfully deployed, use similar commands to deploy the *Product* and *Sales* services, bearing in mind that both have a database dependency and rely on previous MySQL services. Change any necessary default database parameters by passing them as environment variables.

To deploy the *Product* service:

```
$ oc new-app -e MYSQL_USER=product -e MYSQL_PASSWORD=password jboss-
eap70-openshift~https://github.com/RHsyseng/MSA-EAP7-OSE.git --context-
dir=Product --name=product-service
```

To deploy the *Sales* service:

```
$ oc new-app -e MYSQL_USER=sales -e MYSQL_PASSWORD=password jboss-
eap70-openshift~https://github.com/RHsyseng/MSA-EAP7-OSE.git --context-
dir=Sales --name=sales-service
```

Finally, deploy the *Presentation* service, which exposes a web tier and an aggregator that uses the three previously deployed services to fulfill the business request:

```
$ oc new-app jboss-eap70-openshift~https://github.com/RHsyseng/MSA-
EAP7-OSE.git --context-dir=Presentation --name=presentation
```

Note that the *Maven* build file for this project specifies a *war* file name of *ROOT*, which results in this application being deployed to the root context of the server.

Once all four services have successfully deployed, the *presentation* service can be accessed through a browser to verify application functionality. First create a route to expose this service to clients outside the OpenShift environment:

```
$ oc expose service presentation --hostname=msa.example.com
NAME   HOST/PORT   PATH   SERVICE    LABELS             TLS
TERMINATION
presentation msa.example.com  presentation app=presentation
```

The route tells the deployed router to load balance any requests with the given host name among the replicas of the *presentation* service. This host name should map to the IP address of the hosts where the router has been deployed, not necessarily where the service is hosted. For clients outside of this network and for testing purposes, simply modify your */etc/hosts* file to map this host name to the IP address of the master host.

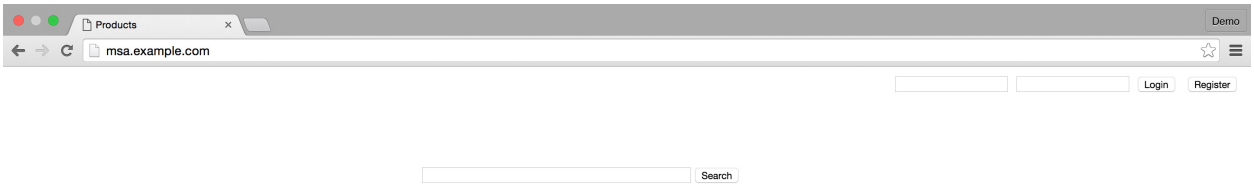
To verify that every step has been correctly performed and was successful, this document reviews the working application and allows you to ensure its correct behavior.

3.1.3.4. Running the Application

3.1.3.4.1. Browser Access

To use the application, simply point your browser to the address exposed by the route. This address should ultimately resolve to the IP address of the OpenShift host where the router is deployed.

Figure 3.1. Application Homepage before initialization

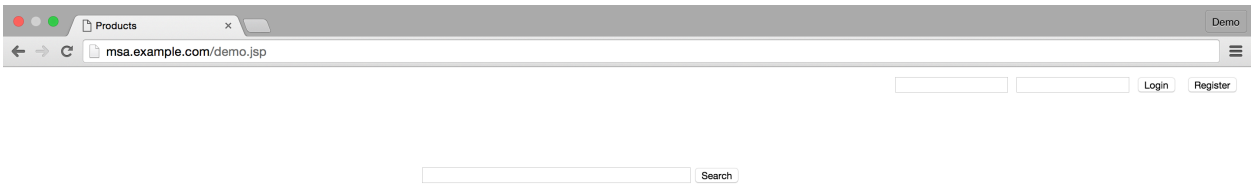


At this stage, the database tables are still empty and content needs to be created for the application to function properly.

3.1.3.4.2. Sample Data

The application includes a demo page that when triggered, populates the database with sample data. To use this page and populate the sample data, point your browser to http://msa.example.com/demo.jsp:

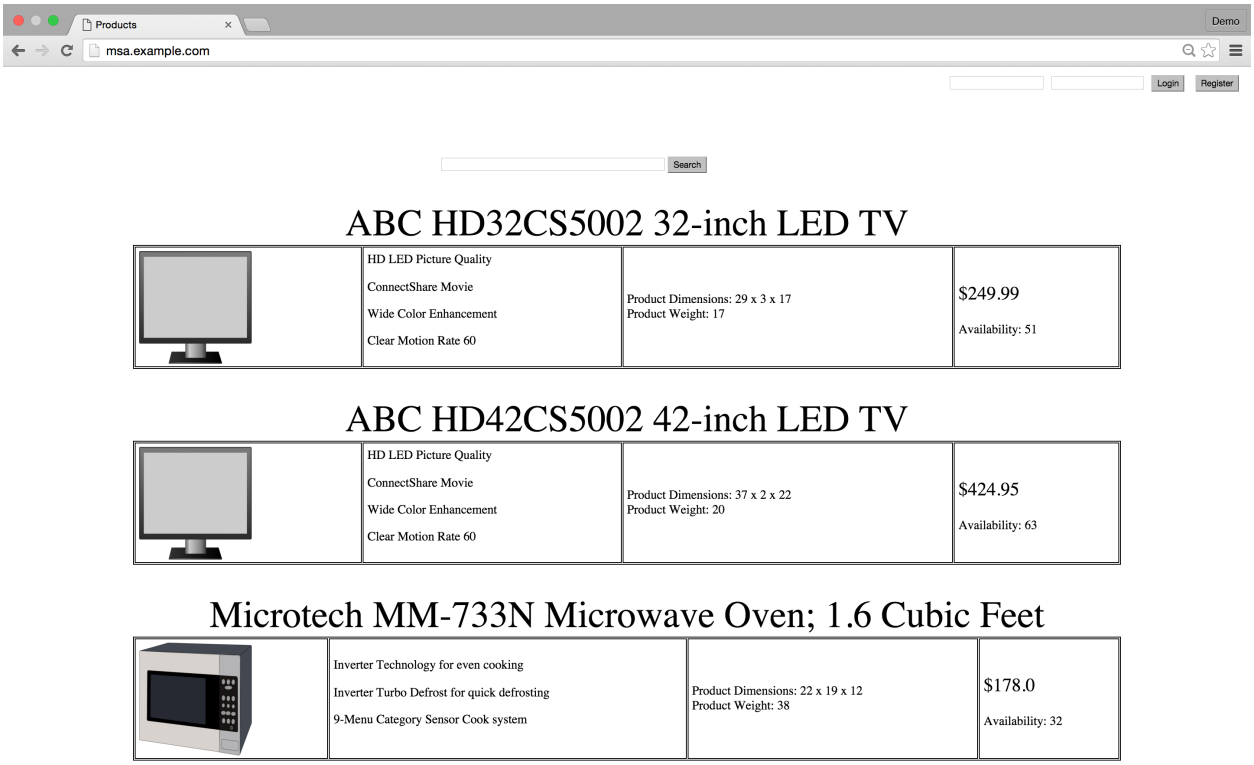
Figure 3.2. Trigger sample data population



3.1.3.4.3. Featured Product Catalog

After populating the product database, the demo page redirects your browser to the route address, but this time you will see the featured products listed:

Figure 3.3. Application Homepage after initialization

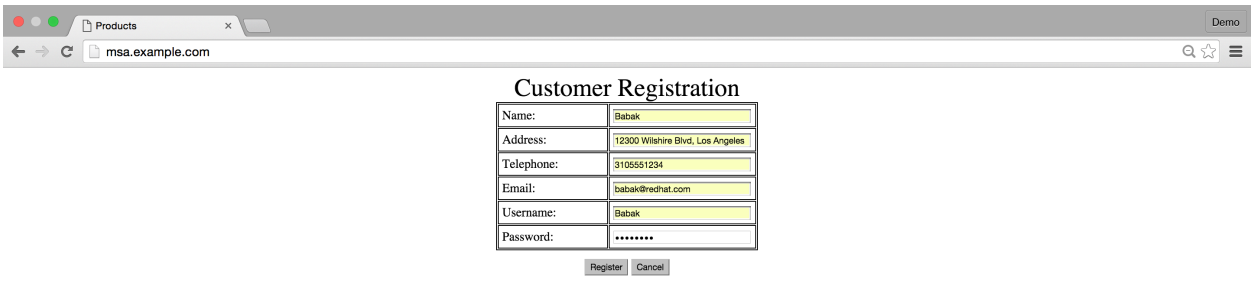


3.1.3.4.4. User Registration

Anonymous users are constrained to browsing inventory, viewing featured products and searching the catalog. To use other features that result in calls to the *Sales* and *Billing* services, a valid customer must be logged in.

To register a customer and log in, click on the *Register* button in the top-right corner of the screen and fill out the registration form:

Figure 3.4. Customer Registration Form



After registration, the purchase button allows customers to add items to their shopping cart, to subsequently visit the shopping cart and check out, and review their order history.

For details on application functionality and how each feature is implemented by the provided services and exposes through their REST API, refer to the previously published reference architecture on Building microservices with JBoss EAP 7.

3.2. BINARY SOURCE DEPLOYMENT

3.2.1. Overview

In environments where a pre-existing build management tool like *maven*, *ant* or other is used, as the target environment transitions from a standalone application server to one based on OpenShift, there may be a requirement or preference to continue using the existing build setup and simply deploy the generated *war* or *ear* file.

The obvious advantage of using binary source deployment is the lower effort and impact of migration from existing environments to OpenShift, as there is no need to change the existing build strategy, source code management, release workflow, or other processes. Only the deployment target is switched to OpenShift.

3.2.2. Usage

To deploy a Java EE application that has already been built, copy the archive file to a location where the *oc* client binary is available. The deployment is a two-step process, where the new application is first defined and created without providing a source of any type. To ensure that no source is provided to the *oc* client, create an empty directory and use it to set up the new application:

```
$ mkdir /tmp/nocontent
$ oc new-app jboss-eap70-openshift~/tmp/nocontent --
name=APPLICATION_NAME
```

This results in a build by the same name being created, which can now be started by providing the binary source, for example:

```
$ oc start-build APPLICATION_NAME --from-file=APPLICATION.war
```

It is in fact rare for an application and all its dependencies to be self-contained in a *war* file. There are often dependencies and custom configuration that require changes to the server. This often means providing a customized *standalone-openshift.xml* for each service.

Place the *standalone-openshift.xml* in a directory called *configuration*, which itself is created at the same level as the *war* file. Assuming you create a *deploy* directory with the *war* file and the *configuration* directory as the only two items inside, the following command is used to deploy the content of the deploy directory:

```
$ oc start-build APPLICATION_NAME --from-dir=deploy
```

It is also possible to set up applications for binary deployment, and build and deploy them with Jenkins, using a combination of available plugins and scripting. Refer to the section on Chapter 4, *Jenkins* for further details.

As an example and to provide further context and details, the following section leverages the binary deployment of Java EE applications to build and deploy the sample microservice application.

3.2.3. Example

3.2.3.1. Deploying Services

This section deploys the same four services as before, but assumes that these services are first built and packaged into *war* files in the local environment, then uploaded to the OpenShift environment.

The deployment of database services is not covered here again, since there is no change in the process of database setup between the different build strategies. Please refer to the previous example for details.

Note

When using *oc new-app*, the source code argument provided (the part after "~" in the [image]~[source code] combination) is assumed to be the location of a local or remote git repository, where the source code resides. For binary deployment, the directory provided to this command should not contain a git repository. To ensure this, create an empty directory to use for this purpose, for example: *mkdir /tmp/nocontent*

Start by building and deploying the *Billing* service, which has no dependencies on either a database or another service. Switch back to the regular user with the associated project and run:

```
$ oc new-app jboss-eap70-openshift:latest~/tmp/nocontent --
name=billing-service
$ oc start-build billing-service --from-file=billing.war
```

Using the OC client, type the *oc start-build --help* command to see the explanation for the *from-file* option.

```
from-file='': A file to use as the binary input for the build; example
a pom.xml or Dockerfile. Will be the only file in the build source.
```

In this example, the *war* file is specified as the input for the *from-file* option, so the *war* file will be sent as a binary stream to the builder. The builder then stores the data in a file with the same name at the top of the build context.

Next, use similar start-build commands to deploy the *Product* and *Sales* services, however both services need to use previously deployed MySQL services, and the configuration is inside */configuration/standalone-openshift.xml*, which is outside the *war* file.

Below is the datasource configuration of *standalone-openshift.xml*.

```
<datasources>
    <!-- ##DATASOURCES## -->
    <datasource jndi-name="java:jboss/datasources/ProductDS"
enabled="true" use-java-context="true" pool-name="ProductDS">
        <connection-
url>jdbc:mysql://${env.DATABASE_SERVICE_HOST:product-
db}:${env.DATABASE_SERVICE_PORT:3306}/${env.MYSQL_DATABASE:product}
</connection-url>
        <driver>mysql</driver>
        <security>
```

```
            <user-name>${env.MYSQL_USER:product}</user-name>
            <password>${env.MYSQL_PASSWORD:password}</password>
        </security>
    </datasource>
```

So merely using the "from-file" option to deploy the web archive is not enough, and the configuration file also needs to be deployed for both *Product* and *Sales* services. To achieve this, the *start-build* command includes another option called "from-dir".

Using the OC client, type "oc start-build --help" command to see the help for the *from-dir* option

```
from-dir='': A directory to archive and use as the binary input for a
build.
```

To include both the web archive and the *standalone-openshift.xml* server configuration file, prepare the directory in this format.

For *Product* service.

For *Sales* services

Command to deploy the *Product* service:

```
$ oc new-app jboss-eap70-openshift:latest~/tmp/nocontent --
name=product-service
$ oc start-build product-service --from-dir=deploy
```

To deploy the *Sales* service:

```
$ oc new-app jboss-eap70-openshift:latest~/tmp/nocontent --name=sales-
service
$ oc start-build sales-service --from-dir=deploy
```

Finally, deploy the *Presentation* service, which exposes a web tier and an aggregator that uses the three previously deployed services to fulfill the business request:

```
$ oc new-app jboss-eap70-openshift:latest~/tmp/nocontent --
name=presentation
```

```
$ oc start-build presentation --from-file=ROOT.war
```

Once all four services have successfully deployed, the *presentation* service can be accessed through a browser to verify application functionality. First create a route to expose this service to clients outside the OpenShift environment:

```
$ oc expose service presentation --hostname=msa.example.com
NAME   HOST/PORT   PATH   SERVICE    LABELS             TLS
TERMINATION
presentation msa.example.com  presentation app=presentation
```

The use of the "from-dir" option is likely to be much more common in production scenarios than "from-file", since most Red Hat JBoss Enterprise Application Platform deployments involve some configuration changes to the server.

To validate the steps taken in this section and ensure that the application is behaving correctly, please refer to the section on running the application in the previous example.

3.3. CONTAINER IMAGE DEPLOYMENT

3.3.1. Overview

An alternative deployment scenario is one where a container image is created outside of OpenShift and then deployed to create a service. This may be entirely a custom image, but is often the result of layers on top of a standard and supported base image. It is common to use the **`Dockerfile`** syntax to add layers on top of existing images.

For more details on creating Docker images, refer to the Red Hat documentation on Creating Docker images

3.3.2. Usage

Assuming the existence of a proper image in a docker registry that is accessible to OpenShift, deploying the image and creating an OpenShift service is very easy and follows a similar syntax to that used for S2I and other methods of creating services:

```
$ oc new-app --docker-image=172.30.12.190:5000/MYNAMESPACE/MYIMAGE --
name=MYSERVICENAME
```

OpenShift provides an internal registry, called the Integrated OpenShift Container Platform Registry, which is ideal for storing images, and to use for the purpose of creating and deploying services.

To directly access this registry, an OpenShift user must be set up with the required roles, and use the login token to directly authenticate against the docker registry. Follow the steps in the Red Hat documentation to grant this user the required privileges.

3.3.3. Example

Use a system admin account, such as the *root* user, to grant the OpenShift user the required roles:

```
# oadm policy add-role-to-user system:registry ocuser
# oadm policy add-role-to-user system:image-builder ocuser
```

Also use this system admin account to determine the connection address of the internal docker registry:

```
# oc get services -n default

NAME                      CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
docker-registry           172.30.182.75    <none>        5000/TCP
81d
...
```

The *ocuser* credentials can now be used to push new images to the registry. Make sure this linux user is part of a user group called *docker* in order to be able to invoke docker commands.

Obtain the user's authentication token and use it to log in to the docker registry:

```
$ oc whoami -t
UVicwxVTEWUGJyj84vT0cGHeM-K2Cq9Mhiq5mQXSl1o
$ docker login -u ocuser -p UVicwxVTEWUGJyj84vT0cGHeM-K2Cq9Mhiq5mQXSl1o
172.30.182.75:5000
Login Succeeded
```

Create a *Dockerfile* using a simple text editor and use it to create a layer on top of the standard *JBoss EAP 7* image, to add the built archive and the server configuration files.

```
# Product Service Docker image
# Version 1

# Pull base image
FROM registry.access.redhat.com/jboss-eap-7/eap70-openshift

# Maintainer
MAINTAINER Calvin Zhu

# deploy app
ADD product.war $JBOSS_HOME/standalone/deployments/
ADD standalone-openshift.xml $JBOSS_HOME/standalone/configuration

USER root
# In the JBoss EAP container, JBoss EAP runs as the jboss user
# but copied files are brought in as root
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/product.war
RUN chown jboss:jboss $JBOSS_HOME/standalone/configuration/standalone-
openshift.xml
USER jboss
```

Note

The creating of layered images on top of the standard EAP 7 image for the purpose of deployment is only done to demonstrate the use of docker images for OpenShift deployment. It would otherwise be simpler and preferrable to use source or binary deployment for the EAP image.

In order to deploy the product service, place *Dockerfile* in a directory along with *product.war*, and

standalone-openshift.xml, which includes the database configuration:

```
$ ls -l
total 52
-rw-rw-r--. 1 czhu czhu   469 Apr 12 22:42 Dockerfile
-rw-rw-r--. 1 czhu czhu 19391 Feb 14 19:07 product.war
-rw-rw-r--. 1 czhu czhu 25757 Apr 12 22:33 standalone-openshift.xml
```

The next step is to build and tag the new image. This tag is subsequently used to push the image to the internal registry.

```
$ docker build -t 172.30.182.75:5000/deploy-project/msa-product .
```

Once the new image is successfully built, push it to the local docker registry:

```
$ docker push 172.30.182.75:5000/deploy-project/msa-product
```

The final step is deploying the service on OpenShift by having it pull this new docker image from the registry:

```
$ oc new-app -e MYSQL_USER=product -e MYSQL_PASSWORD=password --docker-
image=172.30.182.75:5000/deploy-project/msa-product --name=product-
service
```

The process for building the remaining three images is almost identical. The content of the *Dockerfile* for each follows.

Dockerfile for *Billing* service:

```
# Billing Service Docker image
# Version 1

# Pull base image
FROM registry.access.redhat.com/jboss-eap-7/eap70-openshift

# Maintainer
MAINTAINER Calvin Zhu

# deploy app
ADD billing.war $JBOSS_HOME/standalone/deployments/

USER root
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/billing.war
USER jboss
```

Dockerfile for *Sales* service:

```
# Sales Service Docker image
# Version 1

# Pull base image
FROM registry.access.redhat.com/jboss-eap-7/eap70-openshift

# Maintainer
MAINTAINER Calvin Zhu
```

```
# deploy app
ADD sales.war $JBOSS_HOME/standalone/deployments/
ADD standalone-openshift.xml $JBOSS_HOME/standalone/configuration

USER root
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/sales.war
RUN chown jboss:jboss $JBOSS_HOME/standalone/configuration/standalone-
openshift.xml
USER jboss
```

Dockerfile for *Presentation* service

```
# Presentation Service Docker image
# Version 1

# Pull base image
FROM registry.access.redhat.com/jboss-eap-7/eap70-openshift

# Maintainer
MAINTAINER Calvin Zhu

# deploy app
ADD ROOT.war $JBOSS_HOME/standalone/deployments/

USER root
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/ROOT.war
USER jboss
```

Docker scripts for building the new docker images:

```
$ docker build -t 172.30.182.75:5000/deploy-project/msa-billing . $
docker build -t 172.30.182.75:5000/deploy-project/msa-sales . $ docker
build -t 172.30.182.75:5000/deploy-project/msa-presentation .
```

Docker scripts for pushing the new docker images to local registry.

```
$ docker push 172.30.182.75:5000/deploy-project/msa-billing $ docker
push 172.30.182.75:5000/deploy-project/msa-presentation $ docker push
172.30.182.75:5000/deploy-project/msa-sales
```

CLI scripts for deploy the new docker images to OpenShift and expose the service.

```
$ oc new-app --docker-image=172.30.182.75:5000/deploy-project/msa-
billing --name=billing-service $ oc new-app --docker-
image=172.30.182.75:5000/deploy-project/msa-presentation --
name=presentation $ oc new-app -e MYSQL_USER=sales -e
MYSQL_PASSWORD=password --docker-image=172.30.182.75:5000/deploy-
project/msa-sales --name=sales-service $ oc expose service presentation
--hostname=msa.example.com
```

3.3.4. Maven Plugin

The *fabric8-maven-plugin* is a Maven plugin that accelerates development by allowing easy and

quick deployment of Java applications to OpenShift. This plugin reduces the required steps by building the application, creating a docker image and deploying it to OpenShift in a single command, which is especially useful in the development stage, where multiple retries often accompany each step.

The following example demonstrates using *fabric8-maven-plugin* alongside the Dockerfile assets used in the previous section, to build and deploy the same four services to OpenShift.

Each project's *pom.xml* file needs to be updated with the following *plugin* elements. The newly added configuration for the *Product* service looks as follows:

```
<plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>fabric8-maven-plugin</artifactId>
        <version>3.2.28</version>
        <configuration>
            <images>
                <image>
                    <name>172.30.182.75:5000/deploy-project/msa-
product</name>
                  <alias>product-svc</alias>
                  <build>

<dockerFileDir>/home/czhu/dockerBuild/product/</dockerFileDir>
                  </build>
            </image>
            </images>
            <enricher>
                <config>
                      <fmp-controller>
                        <name>product-service</name>
                      </fmp-controller>
                </config>
            </enricher>
            <env>
              <MYSQL_USER>product</MYSQL_USER>
              <MYSQL_PASSWORD>password</MYSQL_PASSWORD>
            </env>
        </configuration>
        <executions>
            <execution>
                <id>fmp</id>
                <goals>
                    <goal>resource</goal>
                    <goal>build</goal>
                    <goal>push</goal>
                </goals>
          </execution>
        </executions>
</plugin>
<plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.0.2</version>
    <executions>
      <execution>
        <id>copy-resources</id>
        <phase>package</phase>
```

```
        <goals>
          <goal>copy-resources</goal>
        </goals>
        <configuration>

<outputDirectory>/home/czhu/dockerBuild/product/</outputDirectory>
          <resources>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Product/target</directory>
                <includes>
                    <include>product.war</include>
                </includes>
            </resource>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Product/configuration</directory>
                <includes>
                    <include>standalone-openshift.xml</include>
                </includes>
            </resource>
          </resources>
        </configuration>
      </execution>
    </executions>
</plugin>
```

The *image* element contains the docker repository name, and the Dockerfile information:

```
<images>
    <image>
        <name>172.30.182.75:5000/deploy-project/msa-product</name>
      <alias>product-svc</alias>
      <build>
        <dockerFileDir>/home/czhu/dockerBuild/product/</dockerFileDir>
      </build>
</image>
</images>
```

The *fmp-controller enricher* is used to name the service. In its absence, the *artifactId* configured in the *pom.xml* will be used.

```
<enricher>
    <config>
          <fmp-controller>
            <name>product-service</name>
          </fmp-controller>
    </config>
</enricher>
```

The env element is used to configure environment variables and replaces passing environment variables through the *-e* flag:

```
<env>
  <MYSQL_USER>product</MYSQL_USER>
  <MYSQL_PASSWORD>password</MYSQL_PASSWORD>
```

```
</env>
```

By using *maven-resources-plugin*, our build file copies both *product.war* and *standalone-openshift.xml* to the *Dockerfile* folder and prepares the for the image build:

```
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.0.2</version>
    <executions>
      <execution>
        <id>copy-resources</id>
        <phase>package</phase>
        <goals>
          <goal>copy-resources</goal>
        </goals>
        <configuration>

<outputDirectory>/home/czhu/dockerBuild/product/</outputDirectory>
          <resources>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Product/target</directory>
                <includes>
                    <include>product.war</include>
                </includes>
            </resource>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Product/configuration</directory>
                <includes>
                    <include>standalone-openshift.xml</include>
                </includes>
            </resource>
          </resources>
        </configuration>
      </execution>
    </executions>
```

Deploying through the plugin also requires the use of Resource Fragments to configure OpenShift services. Create the src/main/fabric8 folder for each project, and add a svc.yml file inside it:

```
[czhu@mw-ocp-master fabric8{master}]$ pwd
/home/czhu/MSA-EAP7-OSE/Product/src/main/fabric8
[czhu@mw-ocp-master fabric8{master}]$ ls -l
total 4
-rw-rw-r--. 1 czhu czhu 157 Apr 19 18:55 svc.yml
```

The content of svc.yml for *Product* service is:

```
apiVersion: v1
kind: Service
metadata:
  name: product-service
spec:
  ports:
```

```
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

With these two changes, you can use the following command to trigger the maven build and deployment:

```
mvn clean fabric8:deploy -Dfabric8.mode=kubernetes
```

To clean up all the OpenShift resources that were just created, run:

```
mvn fabric8:undeploy
```

Additions to *pom.xml* for other services should be as follows.

For the *Billing* service:

```
<plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>fabric8-maven-plugin</artifactId>
        <version>3.2.28</version>
        <configuration>
          <images>
                <image>
                  <name>172.30.182.75:5000/deploy-project/msa-
billing</name>
                  <alias>billing-svc</alias>
                  <build>
<dockerFileDir>/home/czhu/dockerBuild/billing/</dockerFileDir>
                  </build>
            </image>
          </images>
          <enricher>
                <config>
                      <fmp-controller>
                        <name>billing-service</name>
                      </fmp-controller>
                </config>
          </enricher>
        </configuration>
        <executions>
            <execution>
                <id>fmp</id>
                <goals>
                    <goal>resource</goal>
                    <goal>build</goal>
                    <goal>push</goal>
                </goals>
          </execution>
        </executions>
</plugin>
<plugin>
<artifactId>maven-resources-plugin</artifactId>
<version>3.0.2</version>
<executions>
```

```
  <execution>
    <id>copy-resources</id>
    <phase>package</phase>
    <goals>
      <goal>copy-resources</goal>
    </goals>
    <configuration>

<outputDirectory>/home/czhu/dockerBuild/billing/</outputDirectory>
      <resources>
        <resource>
          <directory>/home/czhu/MSA-EAP7-
OSE/Billing/target</directory>
            <includes>
                <include>billing.war</include>
            </includes>
        </resource>
      </resources>
    </configuration>
  </execution>
</executions>
</plugin>
```

For the *Presentation* service:

```
<plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>fabric8-maven-plugin</artifactId>
        <version>3.2.28</version>
        <configuration>
          <images>
                <image>
                    <name>172.30.182.75:5000/deploy-project/msa-
presentation</name>
                  <alias>presentation-svc</alias>
                  <build>

<dockerFileDir>/home/czhu/dockerBuild/presentation/</dockerFileDir>
                  </build>
            </image>
          </images>
          <enricher>
                <config>
                      <fmp-controller>
                        <name>presentation</name>
                      </fmp-controller>
                </config>
          </enricher>
        </configuration>
        <executions>
            <execution>
                <id>fmp</id>
                <goals>
                    <goal>resource</goal>
                    <goal>build</goal>
                    <goal>push</goal>
```

```
                </goals>
          </execution>
        </executions>
</plugin>
<plugin>
<artifactId>maven-resources-plugin</artifactId>
<version>3.0.2</version>
<executions>
  <execution>
    <id>copy-resources</id>
    <phase>package</phase>
    <goals>
      <goal>copy-resources</goal>
    </goals>
    <configuration>

<outputDirectory>/home/czhu/dockerBuild/presentation/</outputDirectory>
      <resources>
        <resource>
          <directory>/home/czhu/MSA-EAP7-
OSE/Presentation/target</directory>
            <includes>
                <include>ROOT.war</include>
            </includes>
        </resource>
      </resources>
    </configuration>
  </execution>
</executions>
</plugin>
```

For the *Sales* service:

```
 <plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>fabric8-maven-plugin</artifactId>
        <version>3.2.28</version>
        <configuration>
            <images>
                <image>
                    <name>172.30.182.75:5000/deploy-project/msa-
sales</name>
                  <alias>sales-svc</alias>
                  <build>

<dockerFileDir>/home/czhu/dockerBuild/sales/</dockerFileDir>
                  </build>
            </image>
            </images>
            <enricher>
                <config>
                      <fmp-controller>
                        <name>sales-service</name>
                      </fmp-controller>
                </config>
            </enricher>
```

```
            <env>
              <MYSQL_USER>sales</MYSQL_USER>
              <MYSQL_PASSWORD>password</MYSQL_PASSWORD>
            </env>
        </configuration>
        <executions>
            <execution>
                <id>fmp</id>
                <goals>
                    <goal>resource</goal>
                    <goal>build</goal>
                    <goal>push</goal>
                </goals>
          </execution>
        </executions>
</plugin>
  <plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.0.2</version>
    <executions>
      <execution>
        <id>copy-resources</id>
        <phase>package</phase>
        <goals>
          <goal>copy-resources</goal>
        </goals>
        <configuration>

<outputDirectory>/home/czhu/dockerBuild/sales/</outputDirectory>
          <resources>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Sales/target</directory>
                <includes>
                    <include>sales.war</include>
                </includes>
            </resource>
            <resource>
              <directory>/home/czhu/MSA-EAP7-
OSE/Sales/configuration</directory>
                <includes>
                    <include>standalone-openshift.xml</include>
                </includes>
            </resource>
          </resources>
        </configuration>
      </execution>
    </executions>
</plugin>
```

The *svc.yml* resource fragment would also be created under src/main/fabric8 folder for each service as described below.

For the *Billing* service:

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: billing-service
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

For the *Presentation* service:

```
apiVersion: v1
kind: Service
metadata:
  name: presentation
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

For the *Sales* service:

```
apiVersion: v1
kind: Service
metadata:
  name: sales-service
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

Once all four services are created in OpenShift, create the route using OpenShift CLI:

```
$ oc expose service presentation --hostname=msa.example.com
```

CHAPTER 4. JENKINS

4.1. OVERVIEW

The use of CI tooling is often required to gain the full benefit of continuous integration. Jenkins is a popular CI tool, with support for build and deployment to OpenShift environments through plugins and other approaches. Openshift provides a supported Jenkins image. Refer to the official OpenShift documentation for details on using this image.

This reference architecture explores two scenarios:

- Using Jenkins outside OpenShift for deployment, where CI tooling is hosted outside of OpenShift and orchestrates the workflow inside OpenShift
- Using an OpenShift Jenkins image, where OpenShift hosts the CI service, either as a central service or per project

4.2. EXTERNAL JENKINS INSTALLATION

4.2.1. Installing the OpenShift Pipeline Plugin

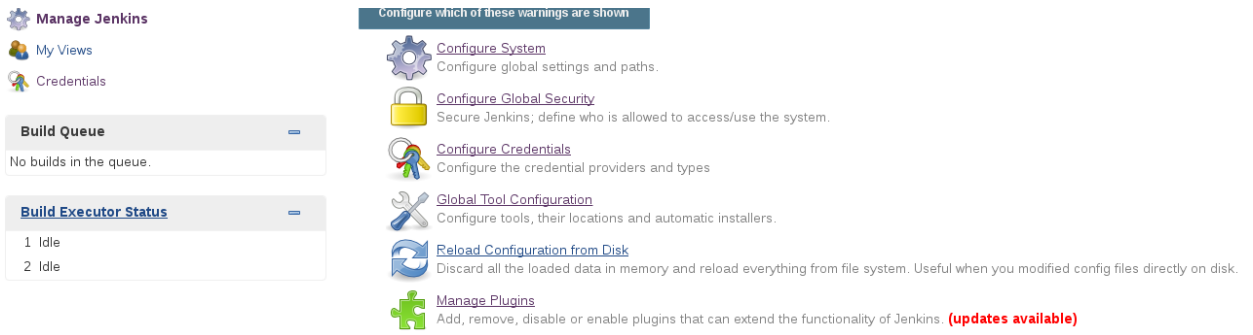
In order for Jenkins to communicate with a remote OpenShift installation, the *OpenShift Pipeline Plugin* needs to be installed first.

https://wiki.jenkins-ci.org/display/JENKINS/OpenShift+Pipeline+Plugin

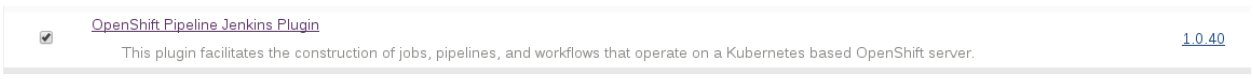
Note

This plugin does *NOT* require the oc binary/CLI to be present. However, for certain features not provided in this plugin, (for example, new-app and start-build with binary source), it's required to download and install the *oc* client binary on the machine where Jenkins is installed.

To install the OpenShift Pipeline Plugin, in Jenkins console, click *Manage Jenkins*, then choose the *Manage Plugins* link.

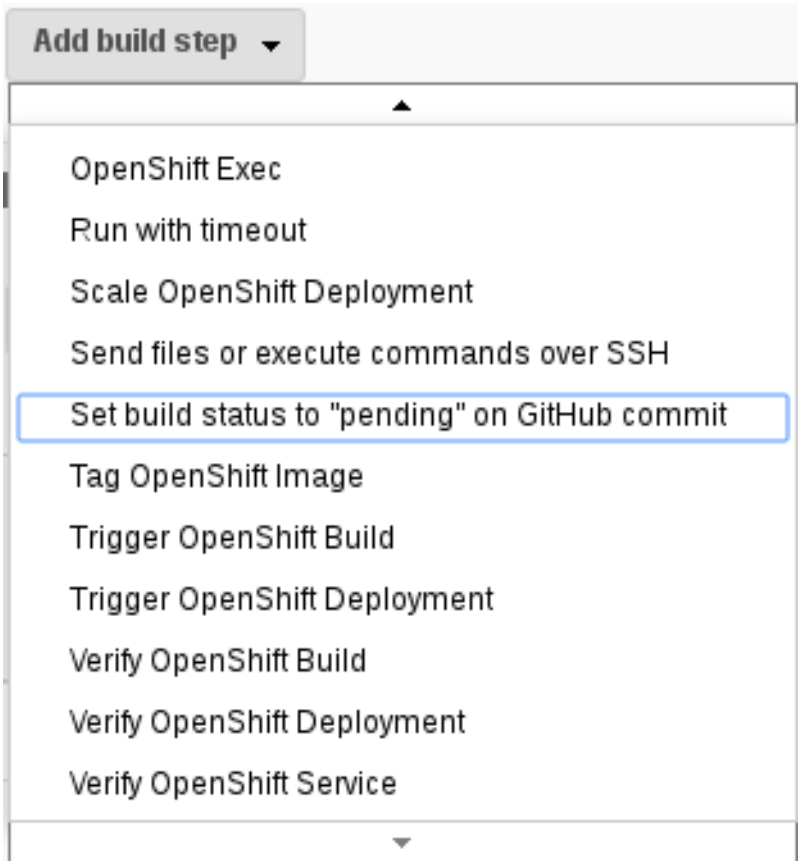


Then select *OpenShift Pipeline Jenkins Plugin*



4.2.2. Using the OpenShift Pipeline Plugin

After the OpenShift Pipeline Plugin is installed, the Jenkins project will expose more OpenShift related steps and options for Build and Post-build Actions.



There is no *new-app* functionality among these options. We currently recommend that if *oc new-app* functionality is desired in your Jenkins jobs, to either use the OpenShift Jenkins image, where the *oc* binary is already provided and can be invoked from the Jenkins jobs, or if not using the OpenShift Jenkins image, include the *oc* binary in your Jenkins installation.

4.2.3. Security

In order for the Jenkins OpenShift plugin to operate against OpenShift resources, the OpenShift service account for the project(s) being operated against will need to have the necessary role and permissions. In particular, you will need to add the *edit* role to the *default* service account in the project those resources are located in.

For this project, called *deploy-project*, the command is:

```
oc policy add-role-to-user edit system:serviceaccount:deploy-
project:default
```

Next, obtain the the bearer authorization token that Jenkins uses to talk to OpenShift. Use the commands *oc describe serviceaccount default* and *oc describe secret <secret name>* for obtaining actual token strings.

```
oc describe serviceaccount default
 Name:  default
 Namespace: deploy-project
 Labels:  <none>
```

```
 Image pull secrets: default-dockercfg-s4iuo

 Mountable secrets:  default-token-zwlpl
             default-dockercfg-s4iuo

 Tokens:             default-token-8tvtm
             default-token-zwlpl



oc describe secret default-token-zwlpl
Name:  default-token-zwlpl
Namespace: deploy-project
Labels:  <none>
Annotations: kubernetes.io/service-account.name=default
  kubernetes.io/service-account.uid=624012ea-e194-11e6-b40e-
beeffeed0002

Type: kubernetes.io/service-account-token

Data
====
namespace: 14 bytes
service-ca.crt: 2186 bytes
token:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2
VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZ
XBsb3ktcHJvamVjdCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5h
bWUiOiJkZWZhdWx0LXRva2VuLXp3bHBsIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3V
udC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3Nlcn
ZpY2VhY2NvdW50L3NlcnG1Z8DVjWB2QMYc4V6xzLcfO3nQbw65uXYtajDI9rTp1CywaBUyp
fvm0fN04zhuWI6dUB8AK9DYpwn8a7Qm2nNnZe37AXfeSpH1gi6f91YhNevOhjrsa8_XiUrP
bGAheSYln5Wt2gWj_SKpw9xvlq_3n8P1yG2QUxNxsJYBHlO7sUpUXA
ca.crt:  1070 bytes
```

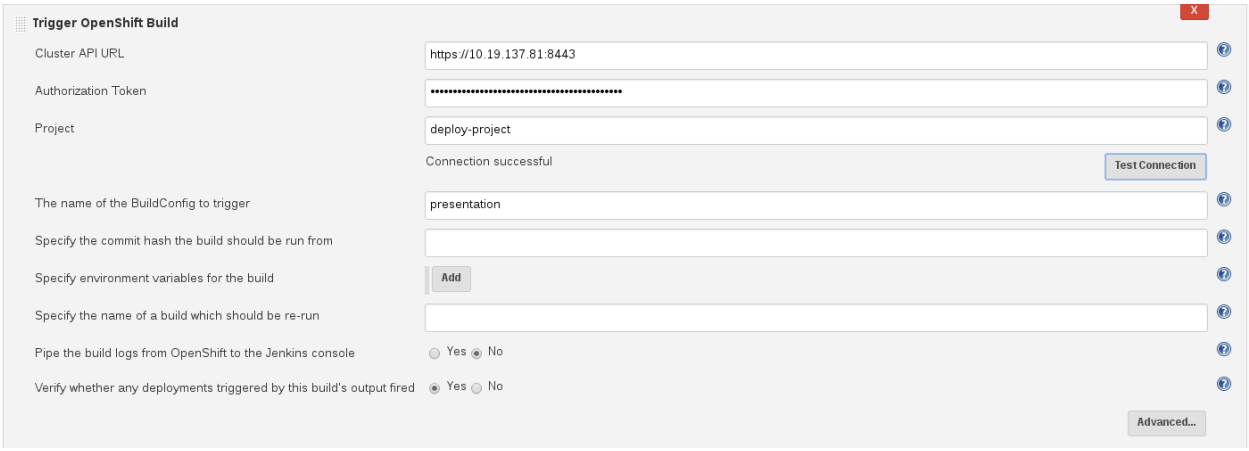
In the absence of a properly signed certificate that has been signed by a trusted certificate authority, the SKIP_TLS parameter needs to be used to avoid a warning:

```
Connection unsuccessful: You have not specified SKIP_TLS, but a cert is
not accessible.  See https://github.com/openshift/jenkins-
plugin#certificates-and-encryption
```

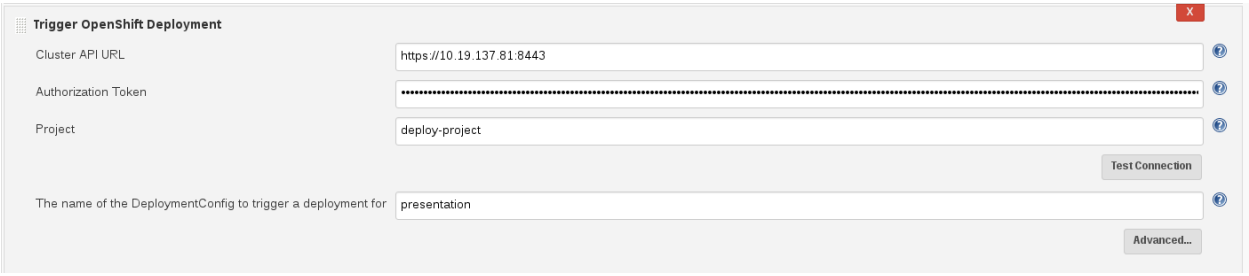
To avoid this exception, just add a string parameter as shown below, to the project configuration:

We can test the authorization token by adding a step like *Trigger Openshift Build*. There is a *Test*

Connection button to test if the *Cluster API URL* and *Authorization Token* are correct or not:



Once these settings are verified, other plugin configuration can be entered with the same values, for example, *Trigger Openshift Deployment*:



4.2.4. The new-app flag

The OpenShift Pipeline Plugin does not provide the *new-app* flag or its equivalent functionality; this restricts the use of the plugin to existing OpenShift applications.

In cases where it's absolutely required to start a new OpenShift application from Jenkins, the following two solutions can help bypass this limitation:

1- Install the *Publish Over SSH Plugin*, and invoke oc client on the OpenShift server.

Using this plugin, Jenkins can remote execute the *oc new-app* command on the OpenShift server.

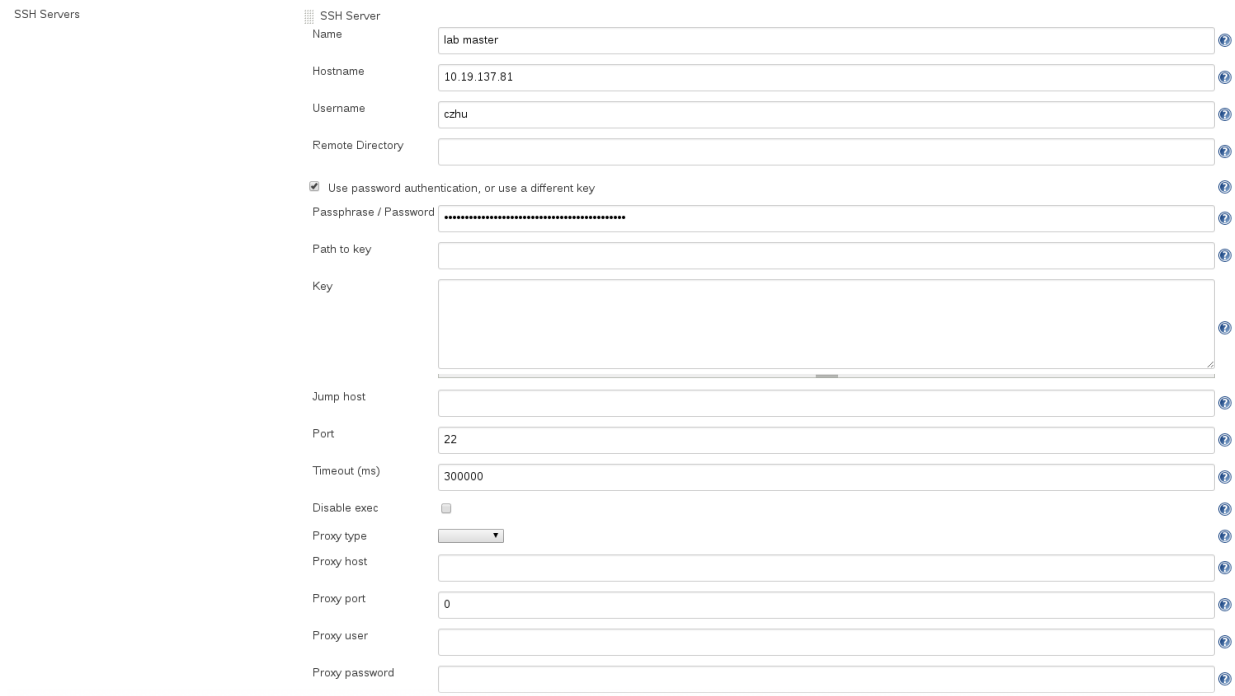
2- Install the oc client binary/CLI on the node where Jenkins is installed. Then invoke oc client from Jenkins.

Further details on these two solutions follows.

4.2.4.1. Publish Over SSH Plugin

The *Publish over SSH* plugin invokes the existing oc client on the OpenShift server.

After installing the *Publish Over SSH Plugin*, define the SSH Server in *Manage Jenkins*, under the *Configure System* section:



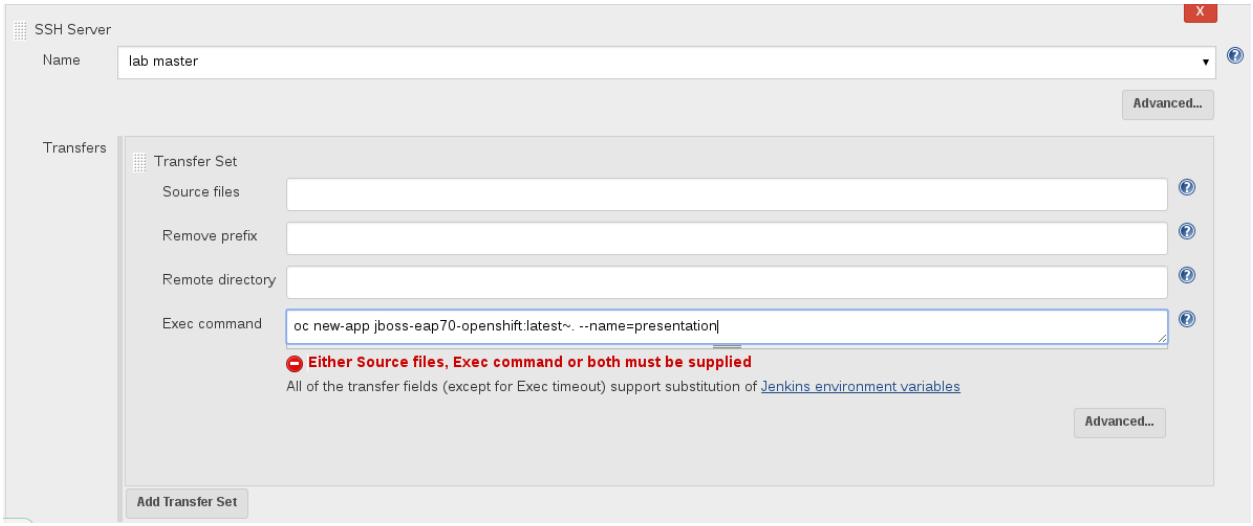
With this plugin in place, a new step called *Send files or execute commands over SSH* can be added and used in projects.

Example 1: Transfer *war* file to OpenShift server.

Console output for the build:

```
SSH: Transferred 1 file(s)
```

Example 2: Trigger *oc new-app* command on OpenShift.



Console output for the build:

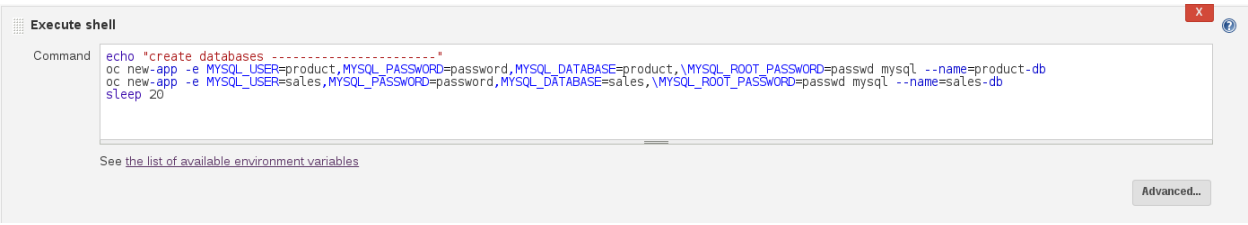
```
SSH: EXEC: STDOUT/STDERR from command [oc new-app jboss-eap70-
openshift:latest~. --name=presentation] ...
--> Found image 926ec6a (11 weeks old) in image stream "jboss-eap70-
openshift" in project "openshift" under tag "latest" for "jboss-eap70-
openshift:latest"

    JBoss EAP 7.0
    -------------
    Platform for building and running JavaEE applications on JBoss EAP
7.0

    Tags: builder, javaee, eap, eap7

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream
"presentation:latest"
      * Use 'start-build --from-dir=DIR|--from-repo=DIR|--from-
file=FILE' to trigger a new build
      * WARNING: a binary build was created, you must specify one of --
from-dir|--from-file|--from-repo when starting builds
    * This image will be deployed in deployment config "presentation"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by
service "presentation"
      * Other containers can access this service through the hostname
"presentation"

--> Creating resources with label app=presentation ...
    imagestream "presentation" created
    buildconfig "presentation" created
    deploymentconfig "presentation" created
    service "presentation" created
--> Success
    Use 'oc start-build presentation' to start a build.
    Run 'oc status' to view your app.
SSH: EXEC: completed after 600 ms
```

4.2.4.2. Using CLI with Jenkins

This solution requires the oc client binary/CLI on the node where Jenkins is installed, in order to invoke oc client from Jenkins.

Download command line interface (CLI) tools, as instructed in the official documentation.

After installing the CLI on the same machine where Jenkins is installed, the *oc* command can be invoked directly using the *Execute Shell* step. Remember to add CLI to the PATH, so that it can be invoked directly everywhere.



The *OpenShift Pipeline Plugin* has a step called *Trigger OpenShift Build*, but this step does not have a corresponding option to do binary source deployment. To deploy *war* files, use a similar approach to the above, to bypass the limitation.

4.3. OPENSHIFT JENKINS IMAGE

There are two major differences in terms of the functionality and usage of the Jenkins image, compared to the Jenkins installation outside of OpenShift:

- No need to install the *OpenShift Pipeline Plugin* as it's already included as part of the Jenkins image.
- No need to configure security token, etc, in the Jenkins steps.

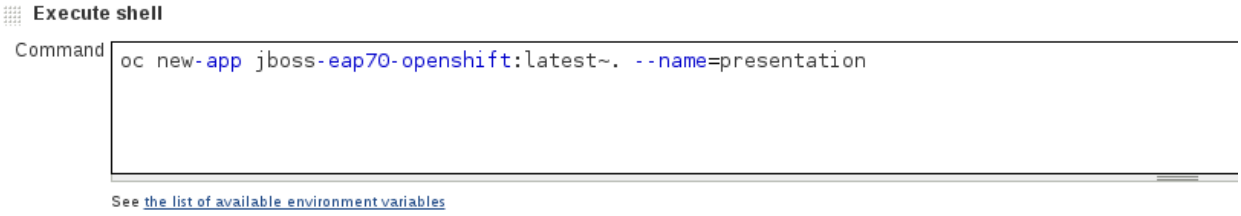
As documented on *OpenShift Pipeline Plugin* project page:

> If that project is also where Jenkins is running out of, and hence you are using the OpenShift Jenkins image (https://github.com/openshift/jenkins[https://github.com/openshift/jenkins]), then the bearer authorization token associated with that service account is already made available to the plugin (mounted into the container Jenkins is running in at "/run/secrets/kubernetes.io/serviceaccount/token"). So you don't need to specify it in the various build step config panels.

Here is one example of configuring the *Trigger OpenShift Build* step on the Jenkins image. In contrast to the standalone Jenkins configuration, the only value that needs to be configured here is the *BuildConfig* name.

However, there is still no *new-app* functionality in the Jenkins console. In order to start a new OpenShift application, either run the *oc new-app* directly in OpenShift before using the Jenkins image, or tigger it from the Jenkins console using *Execute Shell*.

Below is an example of triggering *new-app* from the Jenkins console:



Console output:

```
[workspace] $ /bin/sh -xe /tmp/hudson6043280415050924590.sh
+ oc new-app jboss-eap70-openshift:latest~. --name=presentation
--> Found image 926ec6a (11 weeks old) in image stream
"openshift/jboss-eap70-openshift" under tag "latest" for "jboss-eap70-
openshift:latest"

    JBoss EAP 7.0
    -------------
    Platform for building and running JavaEE applications on JBoss EAP
7.0

    Tags: builder, javaee, eap, eap7

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream
"presentation:latest"
      * A binary build was created, use 'start-build --from-dir' to
trigger a new build
    * This image will be deployed in deployment config "presentation"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by
service "presentation"
      * Other containers can access this service through the hostname
"presentation"

--> Creating resources ...
    imagestream "presentation" created
```

```
    buildconfig "presentation" created
    deploymentconfig "presentation" created
    service "presentation" created
--> Success
    Use 'oc start-build presentation' to start a build.
    Run 'oc status' to view your app.
Finished: SUCCESS
```

4.3.1. Jenkins Image Installation

Follow the instructions provided in the official OpenShift documentation to install and configure the Jenkins image.

To configure persistent volumes:

```
$ oc new-app jenkins-persistent
```

For the ephemeral mode, where configuration does not persist across pod restarts:

```
$ oc new-app jenkins-ephemeral
```

Note

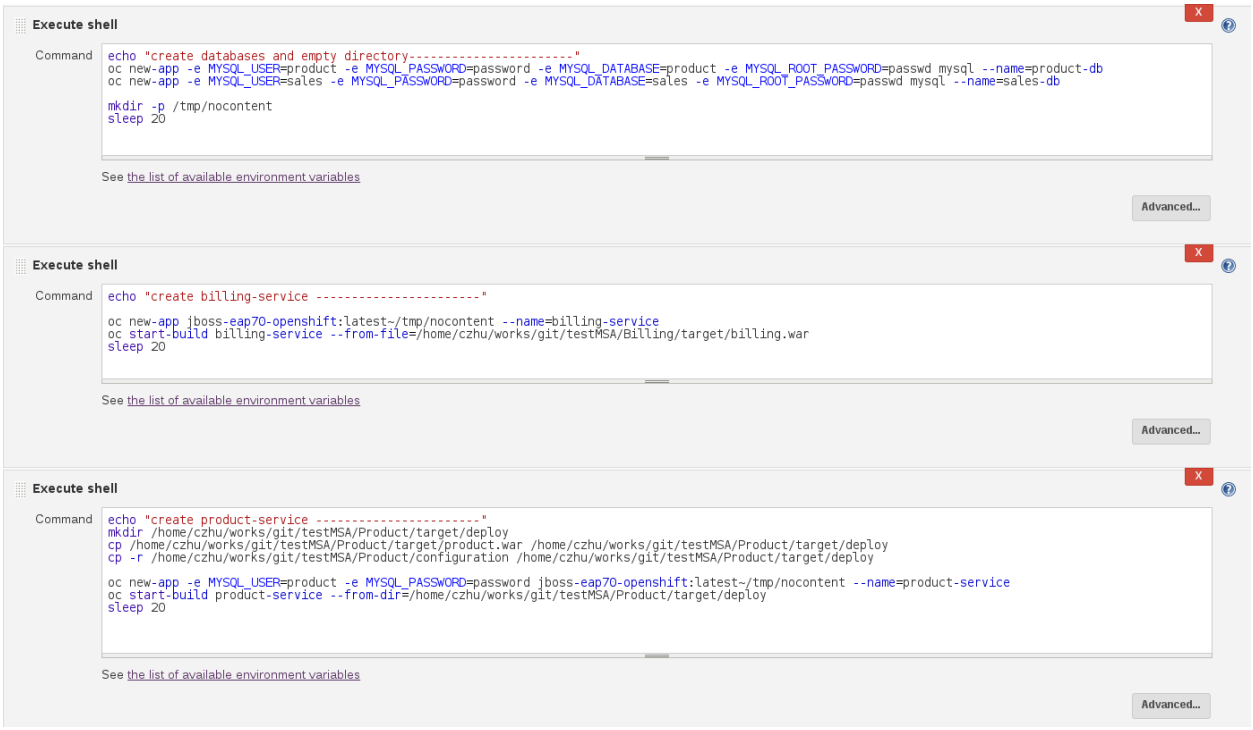
If you instantiate the template against releases prior to v3.4 of OpenShift Container Platform, standard Jenkins authentication is used, and the default *admin* account will exist with a password of *password*.

4.4. EXAMPLES

4.4.1. Using Jenkins to deploy war files to a remote OpenShift

First create a project "MSA_create_all_services_cli" which builds the demo application through maven.

Then, in this project, add the *Execute Shell* step to invoke the locally installed CLI. Create all six services through the *oc new-app* command, and create routes to expose the service. This project should only need to be executed once for the whole lifecycle of the OCP application.

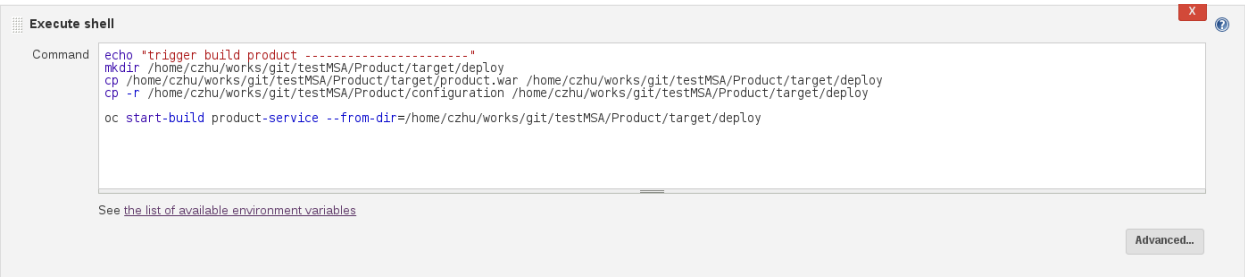


Since database services and the application route only need to be created once, there is no need to configure any additional steps for them. For *Billing*, *Product*, *Sales*, and _Presentation_services, multiple deployments are required for new code release, so proceed to define four new projects for these services:

For each, use a similar *Execute Shell* step to invoke the "oc start-build" command for *war* files.

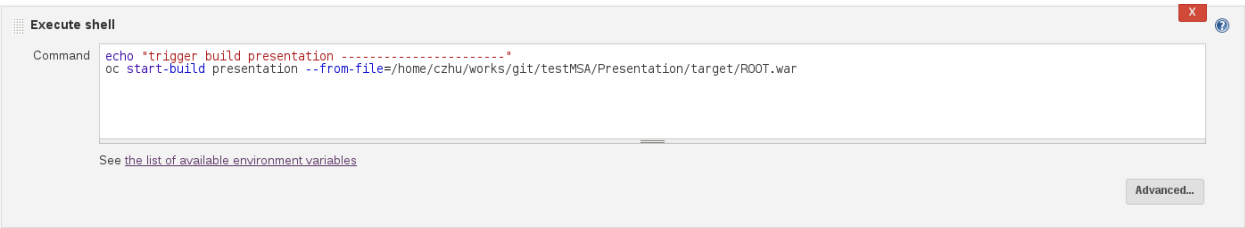
Redeploy *Billing*:

Redeploy *Product*:



Redeploy *Sales*:

Redeploy *Presentation*:



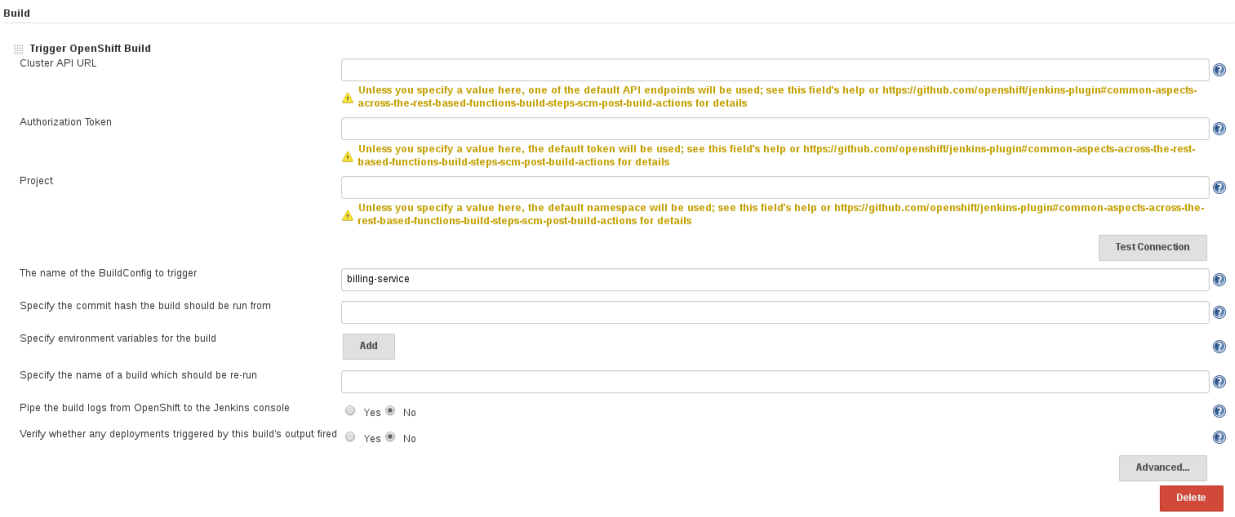
4.4.2. Using the Jenkins image to deploy with S2I

This example also creates 5 projects.

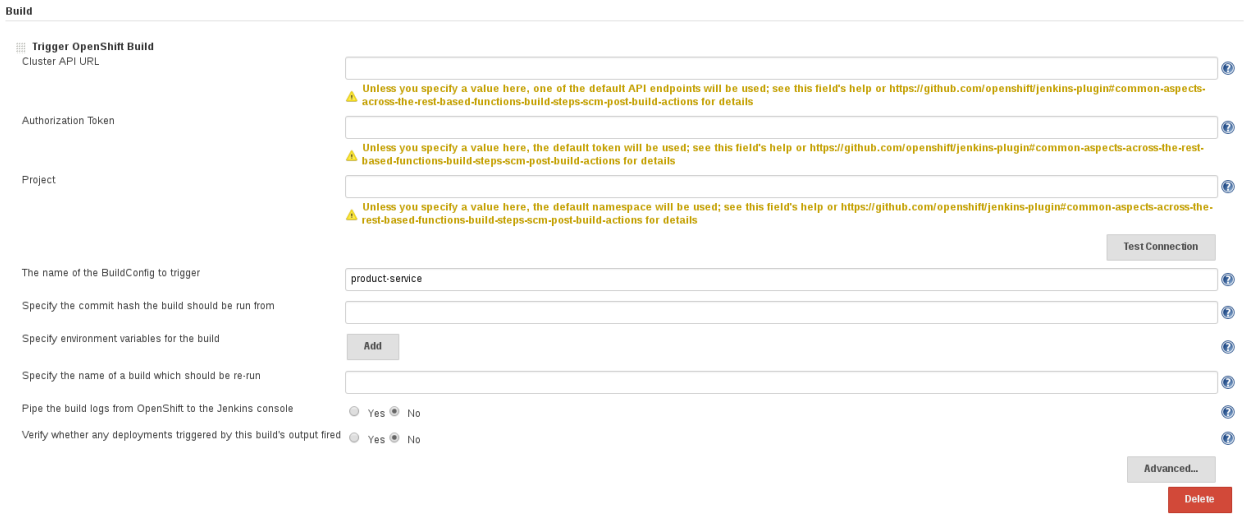
The first project, "MSA_create_all_services" uses the *Execute Shell* step to invoke CLI, creating all six services through the *oc new-app* command and creating a route to expose a service. This time, services are created through S2I, instead of an already-built war file being deployed in the case of the last example.

Since database services and the application route only need to be created once, there is no need to configure any additional steps for them. For *Billing*, *Product*, *Sales*, and _Presentation_services, multiple deployments are required for new code release, so proceed to define four new projects for these services:

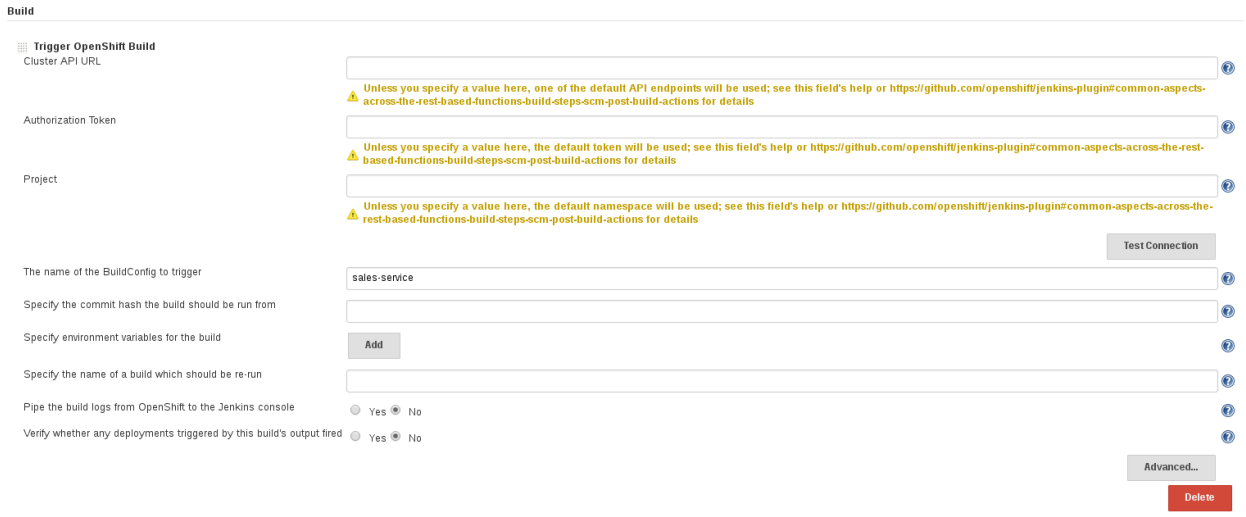
Redeploy *Billing*:



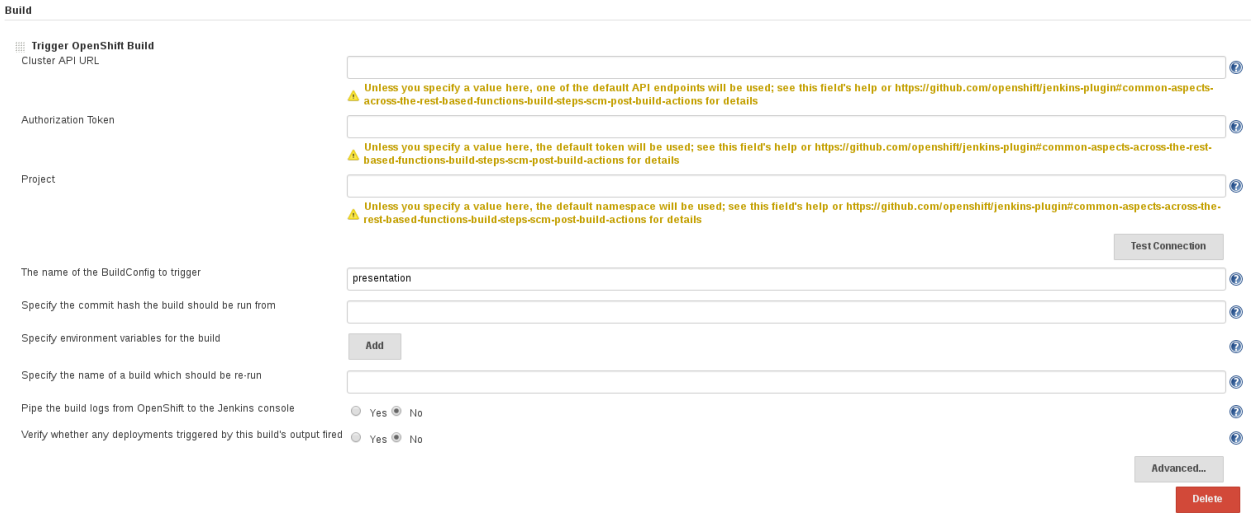
Redeploy *Product*:



Redeploy *Sales*:



Redeploy *Presentation*:



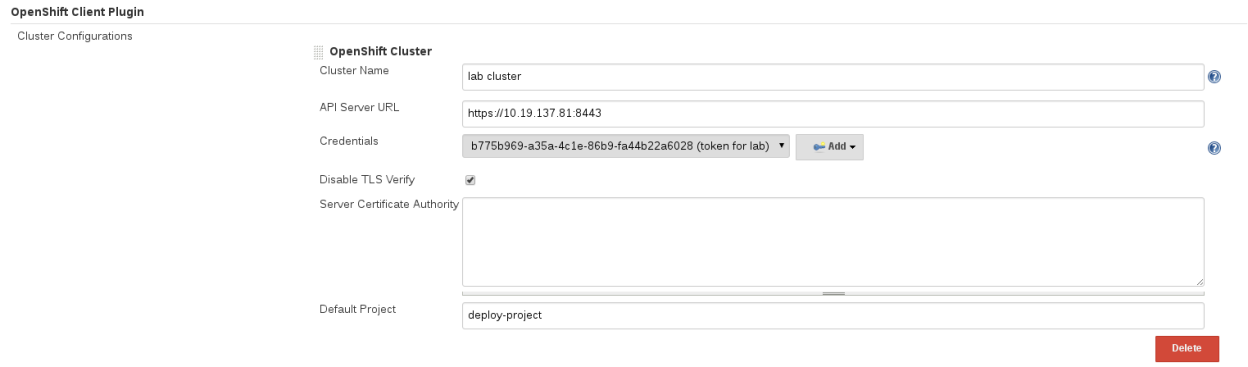
This approach uses S2I to build and deploy the application, and by relying on a Jenkins image running in OpenShift, the project configuration is simplified. It is enough to simply use the *Trigger OpenShift Build* step with the corresponding *buildConfig* name.

4.5. EXPERIMENTAL JENKINS PLUGIN

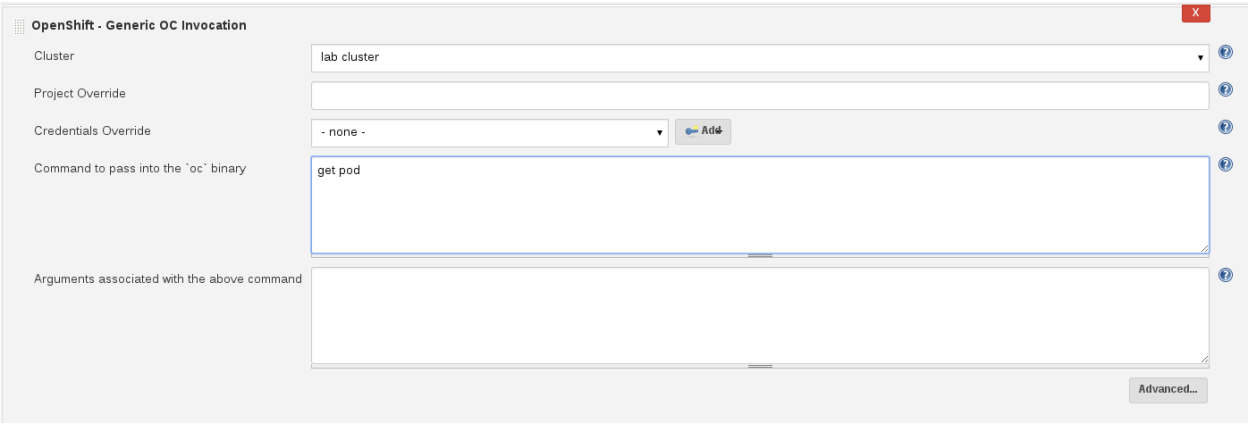
4.5. EXPERIMENTAL JENKINS PLUGIN

The OpenShift Pipeline Plugin does not have native support for some functionalities, including the *new-app* and *start-build* features with binary source. However, there is a new Jenkins plugin with support for various CLI features. This plugin wraps the *oc* client component. The OpenShift Jenkins Pipeline (DSL) Plugin is not currently supported and this paper therefore places less focus on it.

This plugin provides easier integration with OpenShift; for example, an Openshift Cluster can be configured simply with URL, credentials and the default project name.



Sending custom command to OpenShift using this plugin is much easier. The *oc* command can be supplied after simply selecting the newly defined Openshift Cluster:



Warning

As of the time of writing, this plugin is experimental and not supported by Red Hat.

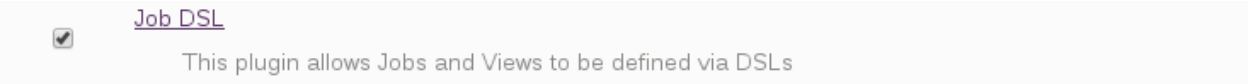
4.6. JOB DSL PLUGIN

Job DSL is a very popular plugin for building Jenkins jobs. This plugin simplifies creating and managing pipelines and can define Jenkins jobs in plain text, using a Groovy-based domain specific language (DSL).

The DSL language output is a job configuration XML file (*config.xml* file under the job folder). DSL plugin provides high level API to work with this configuration file, and in the absence of high level API, the DSL supports configuration blocks to directly access the XML content of *config.xml*.

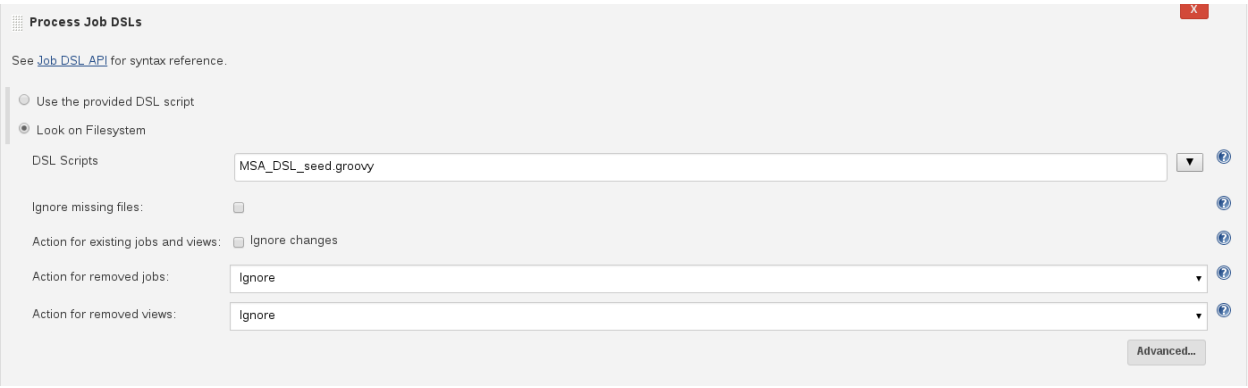
For example, we will explore using this DSL plugin to build the MSA project in an OpenShift environment.

First, install the Job DSL plugin from Manage Jenkins section of the administration web console.



After installation, create a DSL seed job that uses the *Free-style* project style to run the Scripts.

Seed jobs can use the DSL script directly in the console, or use a DSL script on the file system. In this example, a script from the file system is used, as it's easier to edit, and the script can be stored in a Git repository.



Details of the MSA_DSL_seed.groovy script:

The script sets up five jobs, which are: MSA_DSL_create_all, MSA_DSL_redeploy_billing, MSA_DSL_redeploy_product, MSA_DSL_redeploy_sales and MSA_DSL_redeploy_presentation.

It uses configuration blocks to define the same *customWorkspace* for all five jobs, since they all apply to the same project.

```
    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }
```

The script then sets up all jobs to use the same repository, git://github.com/RHsyseng/MSA-EAP7-OSE.git, through the SCM API.

```
    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }
```

The following sections are similar to previous examples. The script uses the *steps* API to set up the build steps for each job, they will run a maven build, then will create services through a shell terminal, and finally deploy the *war* files through the *oc* client, which is installed locally next to Jenkins.

The full script content follows:

```
job('MSA_DSL_create_all') {

    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }

    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }

    steps {
        maven('-e clean package')

        shell('oc new-app -e MYSQL_USER=product -e
MYSQL_PASSWORD=password -e MYSQL_DATABASE=product -e
MYSQL_ROOT_PASSWORD=passwd mysql --name=product-db')
        shell('oc new-app -e MYSQL_USER=sales -e
MYSQL_PASSWORD=password -e MYSQL_DATABASE=sales -e
MYSQL_ROOT_PASSWORD=passwd mysql --name=sales-db')
        shell('mkdir -p /tmp/nocontent')
        shell('sleep 60')

        shell('oc new-app jboss-eap70-openshift:latest~/tmp/nocontent -
-name=billing-service')
        shell('oc start-build billing-service --from-
file=Billing/target/billing.war')

        shell('mkdir Product/target/deploy')
        shell('cp Product/target/product.war Product/target/deploy')
        shell('cp -r Product/configuration Product/target/deploy')
        shell('oc new-app -e MYSQL_USER=product -e
MYSQL_PASSWORD=password jboss-eap70-openshift:latest~/tmp/nocontent --
name=product-service')
        shell('oc start-build product-service --from-
dir=Product/target/deploy')

        shell('mkdir Sales/target/deploy')
        shell('cp Sales/target/sales.war Sales/target/deploy')
        shell('cp -r Sales/configuration Sales/target/deploy')
        shell('oc new-app -e MYSQL_USER=sales -e
MYSQL_PASSWORD=password jboss-eap70-openshift:latest~/tmp/nocontent --
name=sales-service')
        shell('oc start-build sales-service --from-
dir=Sales/target/deploy')

        shell('oc new-app jboss-eap70-openshift:latest~/tmp/nocontent -
-name=presentation')
        shell('oc start-build presentation --from-
file=Presentation/target/ROOT.war')

        shell('oc expose service presentation --
hostname=msa.example.com')
```

```
    }
}

job('MSA_DSL_redeploy_billing') {
    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }


    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }

    steps {
        maven('-e clean package')
        shell('oc start-build billing-service --from-
file=Billing/target/billing.war')
    }
}


job('MSA_DSL_redeploy_product') {
    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }


    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }

    steps {
        maven('-e clean package')
        shell('mkdir Product/target/deploy')
        shell('cp Product/target/product.war Product/target/deploy')
        shell('cp -r Product/configuration Product/target/deploy')
        shell('oc start-build product-service --from-
dir=Product/target/deploy')

    }
}


job('MSA_DSL_redeploy_sales') {
    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }


    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }
```

```
    steps {
        maven('-e clean package')
        shell('mkdir Sales/target/deploy')
        shell('cp Sales/target/sales.war Sales/target/deploy')
        shell('cp -r Sales/configuration Sales/target/deploy')
        shell('oc start-build sales-service --from-
dir=Sales/target/deploy')
    }
}

job('MSA_DSL_redeploy_presentation') {
    configure { project ->
        project / 'customWorkspace' <<
'${JENKINS_HOME}/workspace/MSA_DSL'
    }


    scm {
        git('git://github.com/RHsyseng/MSA-EAP7-OSE.git')
    }

    steps {
        maven('-e clean package')
        shell('oc start-build presentation --from-
file=Presentation/target/ROOT.war')
    }
}
```

Building the MSA_seed project will generate these sub-projects:

Inside each sub-project, the SCM, Maven and build steps are already defined. They are ready to be executed.

As demonstrated through these example, the *Job DSL* plugin is a powerful and user friendly plugin that can be used with *oc* client for deployment on OpenShift. It uses Groovy-based scripts to create new jobs, can invoke other plugins through their API, and to perform any other required function that's not defined in the existing API, it can use configuration blocks to achieve the same effect. The resulting scripts can be checked into a source code repository, and easily reused in different environments.

APPENDIX A. REVISION HISTORY

Revision	Release Date	Author(s)
1.1	May 2017	Babak Mozaffari
1.0	Apr 2017	Calvin Zhu

APPENDIX B. CONTRIBUTORS

We would like to thank the following individuals for their time and patience as we collaborated on this process. This document would not have been possible without their many contributions.

Contributor	Title	Contribution
Babak Mozaffari	Manager, Software Engineering & Consulting Engineer	Technical Content Review
Siamak Sadeghianfar	Senior Product Marketing Manager	Technical Content Review
Christoph Goern	Principal Software Engineer	Technical Content Review
Jorge Morales Pou	Senior Product Marketing Manager	Technical Content Review

APPENDIX C. REVISION HISTORY

Revision 1.1-0	**May 2017**	**CZ**