Migrating a Java application to Liberty on Cloud Pak for Apps on Red Hat OpenShift

Contents

Overview

This document describes how to use the migration artifacts produced by IBM Cloud Transformation Advisor to migrate your Java applications to Liberty on Cloud Pak for Apps on Red Hat OpenShift.

You will complete the following steps:

1. Migrate the Java application to Liberty

- You configure the Liberty instance for your application and add the necessary application and third party binaries.

2. Containerize the Liberty instance running the application

- You create a docker image of your working application running in the correctly configured Liberty instance.

3. Deploy application to Cloud Pak for Apps on Red Hat OpenShift

- Tag and add the image to the Red Hat OpenShift image repository and create a running instance of your application.

Variable definitions for tasks in this document:

- **<APP_CONTEXT_ROOT>** is the context root for your application. If not defined elsewhere, for example in ibm-web-ext-xml, this corresponds to the "name" attribute in the <application> element of the server.xml.
- **<APPLICATION_NAME>** is the name of the application.
- **<CONTAINER_ID>** is the container ID for your docker image. To get this value, enter: docker ps
- **<IMAGE_NAME>** is the name you will use for this image, typically the name of the application.
- **<LIBERTY_HOME>** is the location where you have installed Liberty.
- **<LIBERTY_MACHINE>** is the machine where you have installed the Liberty profile.
- **<MIGRATION_ARTIFACTS_HOME>** is the location where you have unzipped the Transformation Advisor artifacts, or cloned the repository.
- **<OCP_PROJECT>** is the name of the OpenShift project where you want to install the application.

Step 1: Migrate the Java application to Liberty

In this step you will migrate your application to a local Liberty server. This will allow you to verify that your application works correctly on Liberty and make any configuration changes where necessary. After you have verified you are working on Liberty, you will be ready to create a Liberty container for your application.

Prerequisites

- A machine with Liberty installed
- A copy of the migration artifacts that you downloaded from Transformation Advisor available on the machine where you have installed Liberty. Alternatively, if you pushed the artifacts to Git, you can clone the repository that you pushed the artifacts to.

Tasks

1. Create a server in the Liberty installation to run your application:

```
cd <LIBERTY_HOME>/bin
./server create server1
```

2. Go to the location of your migration artifacts and copy the application binary (ear/war) in the target directory to the apps directory of Liberty:

```
cd <MIGRATION_ARTIFACTS_HOME>
cp target/*.war <LIBERTY_HOME>/usr/servers/server1/apps
```

3. If it doesn't already exist, create the directory <LIBERTY_HOME>/usr/shared/config/lib/global and copy any additional binaries from the migration location to that location:

```
mkdir -p <LIBERTY_HOME>/usr/shared/config/lib/global
cp src/main/liberty/lib/*
<LIBERTY_HOME>/usr/shared/config/lib/global
```

NOTE: If you did not upload all the binary files before generating the migration artifacts gather them now and copy them to the locations described in this step.

4. Update the server.xml if necessary:

- All passwords have been changed to '???'. Replace these with the correct passwords.
- If there are any additional binaries listed in this file that you do not need, remove any reference to them.

5. Copy the generated server.xml into place:

```
cp src/main/liberty/config/server.xml
<LIBERTY_HOME>/usr/servers/server1/server.xml
```

6. Start the Liberty server:

```
<LIBERTY_HOME>/bin/server start server1
```

7. Check the Liberty logs to confirm that your application has started correctly and to find the URL to for access:

```
cd <LIBERTY_HOME>/usr/servers/server1/logs
vi messages.log
```

NOTE: If you define a <dataSource> element in the server.xml, you may encounter an authentication issue similar to this:

invalid username/password; logon denied

If you see this issue, you may need to enter your username and password in line. Do this by adding the attributes 'user' and 'password' to the element. For example:

```
<proper:es.oracle portNumber="1521"
URL="jdbc:oracle:thin:@9.9.9.9:1522:orcl" user="system" password
="TransAdv01" ……/>
```

8. Verify that the logs contain a line similar to this:

```
TCP Channel defaultHttpEndpoint has been started and is now
listening for requests on host * (IPv6) port 9080
```

If you do not see this then there has been some problem starting the server or launching the application. Search through the log file for more details and debug accordingly.

9. Open your application in the browser by going to the following link:

```
http://<LIBERTY_HOME_MACHINE_IP>:9080/<APP_CONTEXT_ROOT>
```

NOTE: The migration artifacts assist you in the migration of your application. Depending on the nature and complexity of your application, additional configuration may be required to fully complete this task. For example, you may need to complete extra configuration to connect your application with a user registry, and or to configure user security role bindings. Consult the product documentation for more details.

Step 2: Containerize Liberty

In this Step you will containerize your working Liberty installation. You will create a Liberty image that has your migrated application installed and working, and then test the image to confirm that it is operating correctly

Prerequisites

- You have completed *Step 1: Migrate the Java application to Liberty*
- Docker is installed. (You can download from: https://www.docker.com/get-started).
- The machine where you complete this task requires access to the internet to download the Liberty base image.

Tasks

1. Stop the Liberty server if it is running to ensure that the necessary ports are available:

```
<LIBERTY_HOME>/bin/server stop server1
```

2. Ensure the docker service is running. If it's not, start it:

```
service docker start
```

3. Go to where your migration artifacts are located and build your image from the docker file:

```
cd <MIGRATION_ARTIFACTS_HOME>
docker build --no-cache -t "<IMAGE_NAME>:latest" .
```

The base Liberty image will be pulled down and used to create the image that includes your migrated application.

4. Run the image and confirm that it is working correctly:

```
docker run –p 9080:9080 <IMAGE_NAME>:latest
```

5. If everything looks good, the image has been started and mapped to the port 9080. You can access it from your browser with this link:

```
http://<LIBERTY_HOME_MACHINE_IP>:9080/<APP_CONTEXT_ROOT>
```

Optional: Check your image when it is up and running by logging into the container:

```
docker exec –ti <CONTAINER_ID> bash
```

This allows you to browse the file system of the container where your application is running.

Step 3: Deploy your application to Cloud Pak for Apps on Red Hat OpenShift

In this step you will deploy the image you have created to Red Hat OpenShift. These instructions relate to OpenShift 4+ and have been validated on OpenShift 4.2.

Prerequisites

- Access to either a public or private Red Hat OpenShift 4+ environment.

Before you begin

You will need push images to a location that is accessible to the OpenShift cluster. You may use the default image registry in OpenShift, a publicly available registry, or create your own registry.

The migration artifacts generated by Transformation Advisor (specifically the operator/application/application-cr.yaml file) assume that the default image registry is being used. If you choose to use a different registry, remember to update the image property in the YAML file appropriately.

If you are using the OpenShift cluster image registry, you will need to complete one of the following:

- Consult the Red Hat OpenShift documentation to configure OpenShift's image registry to be externally accessible. (For OpenShift 4.2 see: https://docs.openshift.com/container-platform/4.2/registry/securing-exposing-registry.html. Once configured, you can push your image from the location where you built it in the previous step.

- You can build the image on the OpenShift cluster. As of OpenShift 4, `podman` is the tool that you should use to build the image on the cluster. You can build the image using `podman` in a similar way to docker and push to the cluster image registry. Consult the OpenShift and podman documentation for more information.

- Export the image that was built in Step 2 (using "`docker save`"), copy it to the OpenShift cluster, load it there (using "`podman load`"), and push it to the cluster image registry.

The following tasks use the OpenShift cluster registry and assume the image is built and available. If you expose OpenShift cluster registry to be externally accessible, or if you are using another registry, make sure to update the commands accordingly.

IMPORTANT: The YAML file generated by Transformation Advisor assumes that the OpenShift project name is the same as the <APPLICATION_NAME>. If you choose another project name, you must update the <MIGRATION_ARTIFACTS_HOME>/operator/application/application-cr.yaml file to change the repository name for the image location to match your chosen project name:

```
image-registry.openshift-image-
registry.svc:5000/<OCP_PROJECT>/<IMAGE_NAME>
```

Tasks

1. Log in to the OpenShift cluster and create a new project in OpenShift.

```
oc login -u <USER_NAME>
oc new-project <OCP_PROJECT>
```

2. Tag your image appropriately. Ensure that your OpenShift project (namespace) is included:

```
podman tag <IMAGE_NAME>:latest image-registry.openshift-image-
registry.svc:5000/<OCP_PROJECT>/<IMAGE_NAME>:latest
```

3. Login to image registry:

```
podman login -u $(oc whoami) -p $(oc whoami -t) image-
registry.openshift-image-registry.svc:5000
```

4. Push the image to the registry:

```
podman push image-registry.openshift-image-
registry.svc:5000/<OCP_PROJECT>/<IMAGE_NAME>:latest
```

5. Optional: To do a quick deploy and test of the image, you can deploy using the OpenShift UI. Consult the Open Shift documentation for more details: (https://docs.openshift.com/container-platform/4.2/welcome/index.html)

6. Recommended: Deploy your image with its accompanying operator using the following instructions:

a. Change to the operator directory in <MIGRATION_ARTIFACTS_HOME>:

```
cd <MIGRATION_ARTIFACTS_HOME>/operator
```

b. Create the custom resource definition (CRD) for your Liberty application using the provided artifact:

```
oc apply -f application/application-crd.yaml
```

c. The Liberty operator requires the ServiceAccount, Role, and RoleBinding Kubernetes resources to be created. Run the following commands to create them:
```
oc apply -f deploy/service_account.yaml
oc apply -f deploy/role.yaml
oc apply -f deploy/role_binding.yaml
```

d. Create an instance of the Liberty operator:

```
oc apply -f deploy/operator.yaml
```

IMPORTANT: Wait for the Liberty operator installation to complete before going to the next step. You can check the status using `oc get pods` and wait until the <APPLICATION_NAME>-operator pod is ready.

e. Deploy your microservice application using the provided custom resource artifact. If you have chosen a project name other than the application name, review the beginning of this Step before you run this command:

```
oc apply -f application/application-cr.yaml
```

f. You can view the status of your deployment by running oc get deployments. If you don't see the status after a few minutes, query the pods and then fetch the Liberty pod logs:

```
oc get pods
oc logs <pod>
```

g. You can now access your application in two ways:
    1. Create a route for the service created for your application. This can be done from the OpenShift UI. Consult OpenShift documentation for more details on creating routes.
    2. Using the NodePort service. Run the following command to get the service port:
    ```
    oc get service <APPLICATION_NAME> -
    o=jsonpath='{.spec.ports[0].nodePort}'
    ```

    Now, from your browser, go to https://<openshift accessible domain>:<service_port>/<CONTEXT_ROOT>.

7. Optional: If you wish to delete your application from OpenShift, run the commands described here. Only delete the CRD if it is not being used by other applications.

If you have not deployed anything else to the project, you can run the following to delete all resources in the project:

```
oc -n <OCP_PROJECT> delete all --all
```

If you have other resources in the project run the oc get all to see all the resources and remove only the resources particular to the application. Such resources can be readily identified with the <APPLICATION-NAME>

In addition you need to run the following commands:

```
oc -n <OCP_PROJECT> delete role <APPLICATION-NAME>-operator
oc -n <OCP_PROJECT> delete rolebinding <APPLICATION-NAME>-
operator
oc -n <OCP_PROJECT> delete serviceaccount <APPLICATION-NAME>-
operator

oc delete OpenLiberty/<APPLICATION_NAME>
# If it's not working run the following command and delete again
kubectl patch OpenLiberty/<APPLICATION_NAME> -p
'{"metadata":{"finalizers":[]}}' --type=merge

# Only delete the CRD if it is not being used by other
applications
oc delete crd/openliberties.openliberty.io
# If it's not working do this and delete again
kubectl patch crd/openliberties.openliberty.io -p
'{"metadata":{"finalizers":[]}}' --type=merge
```

Building and deploying source projects with IBM Cloud Transformation Advisor

Overview

When migrating an application, you will often need to make changes to the source code to ensure a successful migration to the new target platform. The exact nature of the changes will vary from application to application. Transformation Advisor reports on the changes necessary for each individual application and will classify applications that require code changes as either Moderate or Complex. The WebSphere Application Migration Toolkit (WAMT) Eclipse plugin is available to pinpoint exactly where changes need to be made in the code. The tool can also suggest possible fixes. See this link for more details:
https://developer.ibm.com/wasdev/downloads/#asset/tools-WebSphere_Application_Server_Migration_Toolkit

The following tasks help you build your source code to an image.

Tasks

1. On the migration plan page for your application in Transformation Advisor, select “Source code” as the build type.
The "Send to Git" button will be enabled even if you have not uploaded any binaries (or provided Maven coordinates to binaries).

2. Click on the Send to Git button and review the artifacts in the repository.
The artifacts are essentially a Maven project described by the pom.xml. It has a default Maven project structure with a src and target directory.
In this "source build" mode, the Dockerfile artifact is automatically modified to invoke a Maven build as part of the image build.

Before adding your own source into the project, you can perform a build of the Dockerfile. In this case, a minimum war file will be built during the build of the Dockerfile and deployed in the Liberty container. You can containerize and deploy your application as described in Steps 2 and Step 3 in this document. The source build occurs as part of a multi-stage docker build and so there are not extra prerequisites for performing a source build.