Deployment of app in IBM cloud

Project Date	**September 2022**
Team ID	**PNT2022TMID34297**
Project Name	**Containment Zone Alerting Application**

Containerize the app

The first step in the modernization journey is to containerize applications. To run WebSphere applications like our sample application in a container, we will use ***IBM Cloud Transformation Advisor*.** Transformation Advisor is available **in** ***IBM WebSphere Hybrid Edition***, but it is also available to download separately and run locally using Docker Hub (or Podman).

Prerequisites :

Clone the repo to get the complete source code for the sample application:

```
$ git clone https://github.com/ibm/application-modernization-javaee-
quarkus.git && cd application-modernization-javaee-quarkus
$ ROOT_FOLDER=$(pwd)
```

- Because setting up the sample application in virtual machines or on bare metal servers is not that easy, the WebSphere 8.5.5 application is simulated running in a container in ***Docker desktop*.**

- If you no longer have a license for Docker desktop, Podman is a solid alternative that can be used in its place. Simply follow the commands in this ***blog post*.**

- Then, install the local version of ***Transformation Advisor*** to run on your developer machine using ***Docker desktop*.**

STEPS

1. Upgrade the WebSphere 8.5.5 app to WebSphere 9
2. Run the app in a container

Step 1: Upgrade the WebSphere 8.5.5 app to WebSphere 9 :

You'll use the data collector from Transformation Advisor. First, open `http://localhost:3000/`. Then, create a workspace named `workspace` and a collection named `collection855`. On the Collections page, download the data collector for Linux, `transformationadvisor-Linux_workspace_collection855.tgz`. Then, copy the data collector to `${ROOT_FOLDER}/transformation-advisor/tool`.

First, you must build and run the container that is running the WebSphere 8.5.5 legacy app.

```
$ sh ${ROOT_FOLDER}/scripts/install-dojo.sh
$ sh ${ROOT_FOLDER}/scripts/install-was-dependencies.sh
$ sh ${ROOT_FOLDER}/scripts-docker/build-and-run-monolith-app-was855.sh
```

Next, copy the data collector tool to the WebSphere container and open the container's terminal:

```
$ docker cp {ROOT_FOLDER}/transformation-
advisor/tool/transformationadvisor-Linux_workspace_collection855.tgz
storefront-was855:/tmp
$ docker exec -it storefront-was855 /bin/bash
```

Then, extract the compressed data collector file:

```
$ cd /tmp
$ tar xvfz transformationadvisor-Linux_workspace_collection855.tgz
$ exit
```

Then, copy the configuration file on to the container:

```
$ docker cp ${ROOT_FOLDER}/transformation-advisor/wast855-to-
wast90/customCmd.properties storefront-
was855:/tmp/transformationadvisor-2.4.0/conf
```

For this particular example it's important to define to include the classes from the 'org.pwte' packages.

```
--includePackages=org.pwte
```

Once the data collector and the configuration file has been copied onto the container, the collector tool can be run:

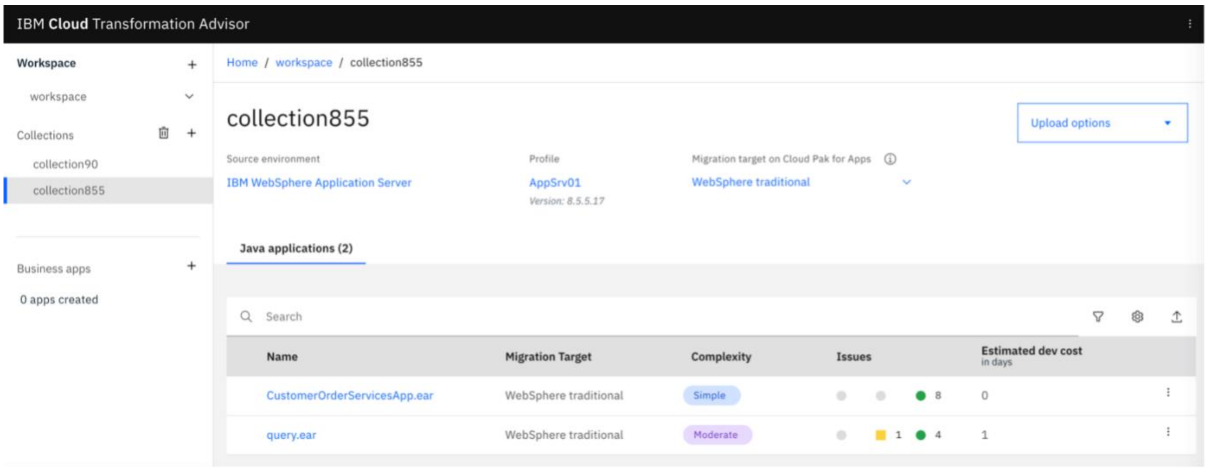
```
$ docker exec -it storefront-was855 /bin/bash
$ cd /tmp/transformationadvisor-2.4.0
$ ./bin/transformationadvisor -w /opt/IBM/WebSphere/AppServer -p
AppSrv01
$ exit
```

Usually the results are uploaded to Transformation Advisor automatically. If this doesn't work, you can do this manually:

```
$ cd ${ROOT_FOLDER}/transformation-advisor/wast855-to-wast90
$ docker cp storefront-was855:/tmp/transformationadvisor-
2.4.0/AppSrv01.zip .
```

Transformation Advisor generates an output file, ***AppSrv01.zip***

Here is the result of running Transformation Advisor in the Web Console. `CustomerOrderServicesApp.ear` is the application that is supposed to be containerized and the estimated effort is low. That's because not a single line of code needs to be changed!



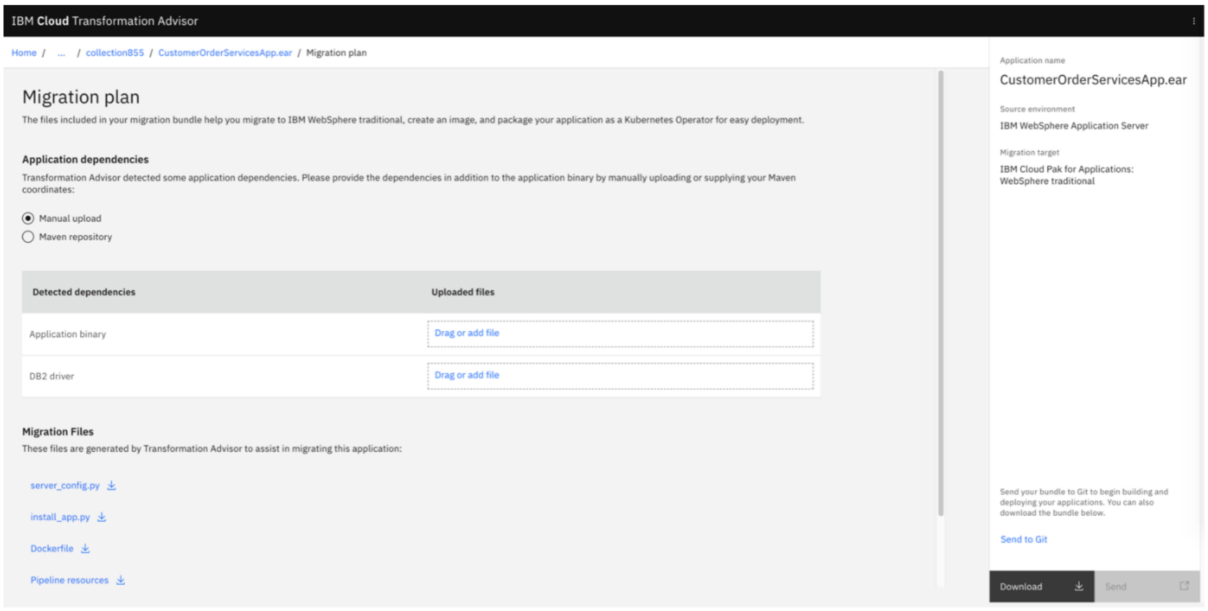
collection855

Source environment: IBM WebSphere Application Server
Profile: AppSrv01 Version: 8.5.5.17
Migration target on Cloud Pak for Apps: WebSphere traditional

Java applications (2)

Name	Migration Target	Complexity	Issues	Estimated dev cost in days
CustomerOrderServicesApp.ear	WebSphere traditional	Simple	● 8	0
query.ear	WebSphere traditional	Moderate	■ 1 ● 4	1

Instead Transformation Advisor automatically generates a ***Dockerfile*** and two Python scripts (which are included in the ***AppSrv01.zip*** file) to run the same application in a container. Click on the name of our app to display the migration plan.


IBM Cloud Transformation Advisor
Home / ... / collection855 / CustomerOrderServicesApp.ear / Migration plan
Send to Git
Download
Send

Migration plan

The files included in your migration bundle help you migrate to IBM WebSphere traditional, create an image, and package your application as a Kubernetes Operator for easy deployment.

Application dependencies

Transformation Advisor detected some application dependencies. Please provide the dependencies in addition to the application binary by manually uploading or supplying your Maven coordinates:

[✓ Manual upload] Maven repository

Detected dependencies	Uploaded files
Application binary	Drag or add file
DB2 driver	Drag or add file

Migration Files

These files are generated by Transformation Advisor to assist in migrating this application:

- server_config.py
- install_app.py
- Dockerfile
- Pipeline resources

Application name: CustomerOrderServicesApp.ear

Source environment: IBM WebSphere Application Server

Migration target: IBM Cloud Pak for Applications: WebSphere traditional

Send your bundle to Git to begin building and deploying your applications. You can also download the bundle below.

The only change that needs to be done in the ***Dockerfile*** is to replace **`{APPLICATION_BINARY}`** with the actual name for the **`.ear`** file.

```
FROM ibmcom/websphere-traditional:latest-ubi

# Manual fix: Replace {APPLICATION_BINARY} with
CustomerOrderServicesApp-0.1.0-SNAPSHOT.ear
COPY --chown=was:root CustomerOrderServicesApp-0.1.0-SNAPSHOT.ear
/work/config/CustomerOrderServicesApp-0.1.0-SNAPSHOT.ear
COPY --chown=was:root ./src/config /work/config
COPY --chown=was:root ./lib /work/config/lib
RUN /work/configure.sh
```

STEP 2: RUN THE SAMPLE APP IN A CONTAINER :

To run the application locally, you can use Docker desktop (or Podman, as described in the **Prerequisites** section above).

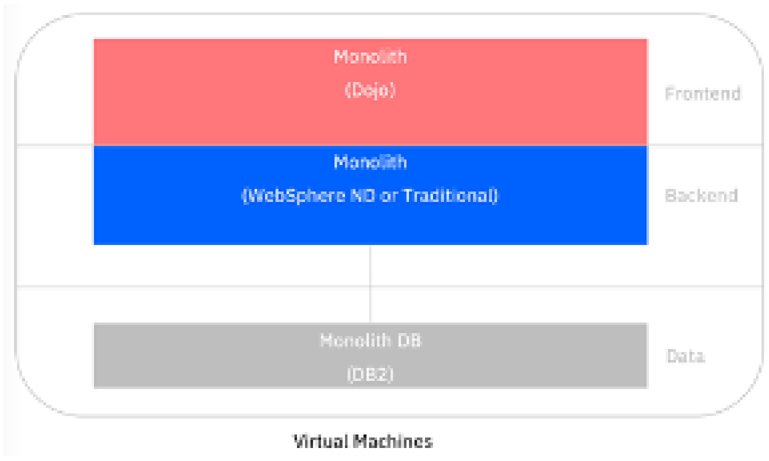
Run these commands to run the containerized version that is running on WebSphere Traditional 9.0:

```
$ sh ${ROOT_FOLDER}/scripts/install-dojo.sh
$ sh ${ROOT_FOLDER}/scripts/install-was-dependencies.sh
$ sh ${ROOT_FOLDER}/scripts-docker/build-and-run-monolith-app-was90.sh
```

- It was a Java EE 6 app that ran on WebSphere Application Server traditional, version 8.5.5, and was deployed to 2 virtual machines.

- It was built with EJBs and REST APIs, using the ***Backends for Frontends*** pattern not the ***RESTful API pattern*.**

The sample application essentially had 3 monoliths: a monolithic Db2 database, a monolithic backend (WebSphere Application Server), and a monolithic frontend (built with the Dojo toolkit). The following architecture diagram shows this 3-tier architecture of the 3 monoliths running in the virtual machines.



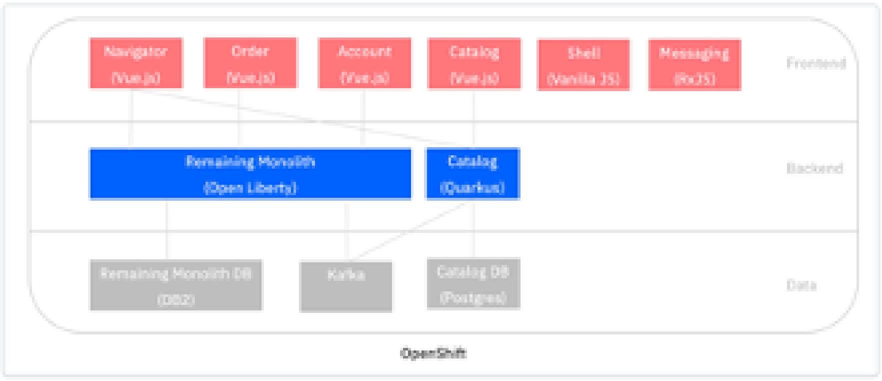
In this sample e-commerce app, you can navigate through products in a catalog using a set of categories. Then, you can select a product and add it to your shopping cart.

Architecture of modernized app :

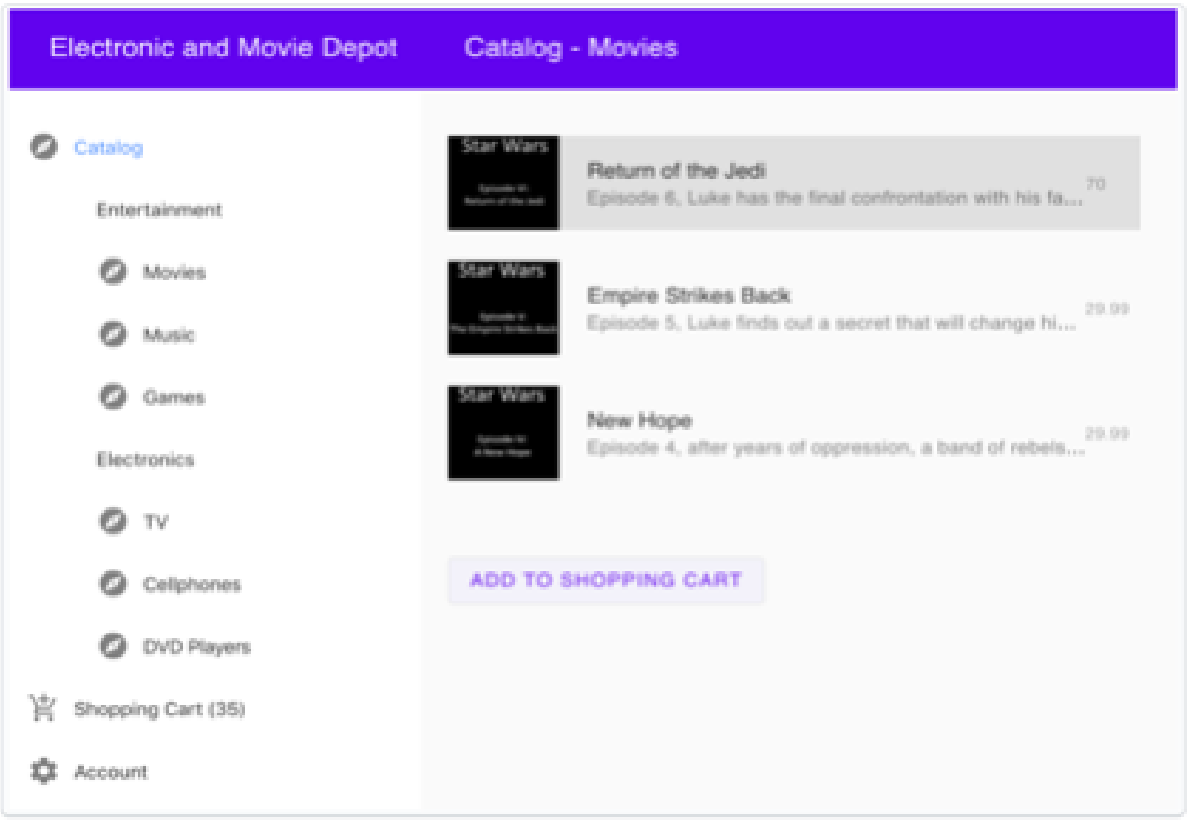
- I started to modernize the sample application back in the summer of 2020.

- The modernized app is a cloud-native, microservices-based application that uses Open J9 and Open Liberty (Jakarta EE, MicroProfile). A version using Quarkus is also available.

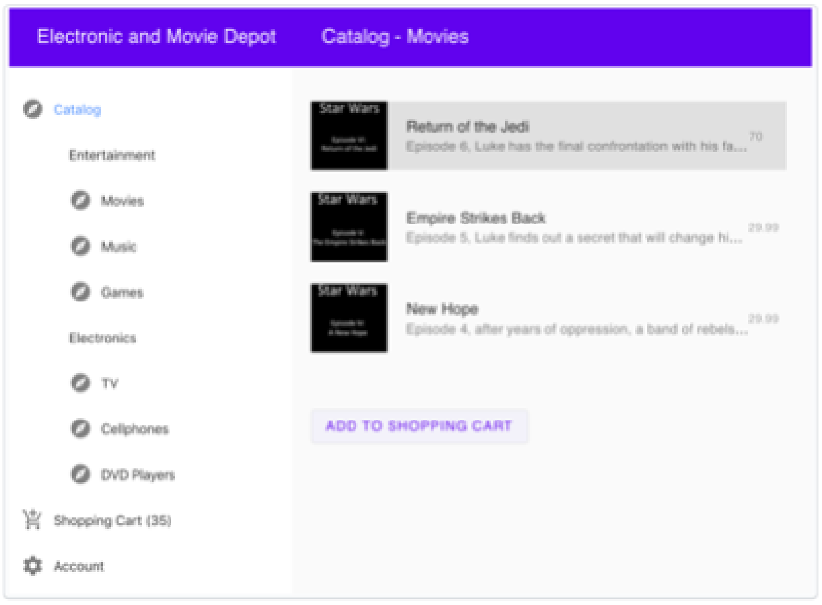
- It is deployed to Red Hat OpenShift in multiple containers: Quarkus or Open Liberty backend, Quarkus Catalog service, micro frontends (navigator, catalog, account, order, shell, and messaging), remaining monolith DB (Db2), Kafka, and Postgres (catalog DB). The following simplified architecture diagram shows the micro frontends, the microservices, and the remaining monolith running in containers deployed on Red Hat OpenShift.



- The new modernized web application supports the same functionality as the legacy one. For this application modernization effort, the goal was not to improve the UI but instead to modularize the UI to be able to update micro-frontends independently from each other.


Electronic and Movie Depot
Catalog - Movies
Catalog
Entertainment
Movies
Music
Games
Electronics
TV
Cellphones
DVD Players
Shopping Cart (35)
Account
Return of the Jedi
Episode 6, Luke has the final confrontation with his fa...
70
Empire Strikes Back
Episode 5, Luke finds out a secret that will change hi...
29.99
New Hope
Episode 4, after years of oppression, a band of rebels...
29.99
ADD TO SHOPPING CART

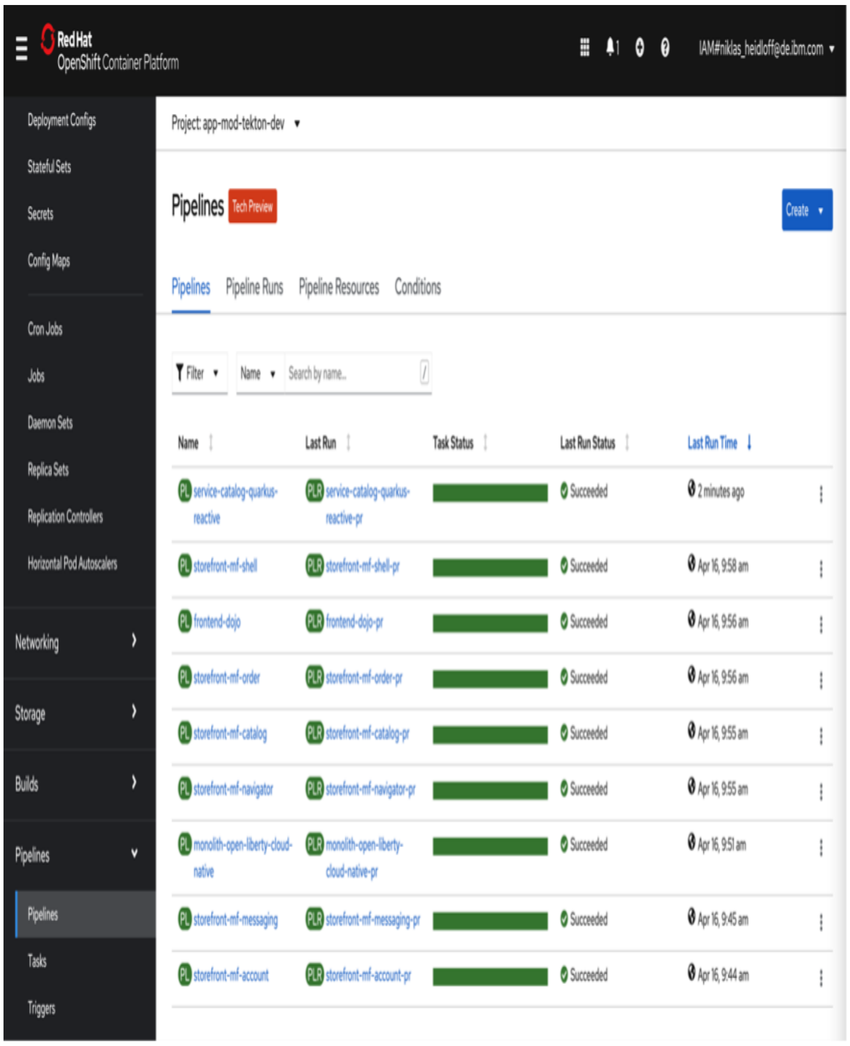
- The following illustration shows the breakdown of the micro-frontends.


Electronic and Movie Depot
Catalog - Movies
Catalog
Entertainment
Movies
Music
Games
Electronics
TV
Cellphones
DVD Players
Shopping Cart (35)
Account
Return of the Jedi
Episode 6, Luke has the final confrontation with his fa...
70
Empire Strikes Back
Episode 5, Luke finds out a secret that will change hi...
29.99
New Hope
Episode 4, after years of oppression, a band of rebels...
29.99
ADD TO SHOPPING CART

RESULTS OF THE MODERNIZED APP :

If we look at the modernized app running in Red Hat OpenShift Container Platform, you can see that you can deploy new functionality separately from each other. For example, let's say that you want to add ratings to the products. You don't have to touch the remaining monolith and just change everything in the different components

- Database of catalog service (Postgres)
- Catalog microservice
- Catalog micro frontend



Name	Last Run	Task Status	Last Run Status	Last Run Time
PL service-catalog-quarkus-reactive	PLR service-catalog-quarkus-reactive-pr	🟩	Succeeded	2 minutes ago
PL storefront-mf-shell	PLR storefront-mf-shell-pr	🟩	Succeeded	Apr 16, 9:58 am
PL frontend-dojo	PLR frontend-dojo-pr	🟩	Succeeded	Apr 16, 9:56 am
PL storefront-mf-order	PLR storefront-mf-order-pr	🟩	Succeeded	Apr 16, 9:56 am
PL storefront-mf-catalog	PLR storefront-mf-catalog-pr	🟩	Succeeded	Apr 16, 9:55 am
PL storefront-mf-navigator	PLR storefront-mf-navigator-pr	🟩	Succeeded	Apr 16, 9:55 am
PL monolith-open-liberty-cloud-native	PLR monolith-open-liberty-cloud-native-pr	🟩	Succeeded	Apr 16, 9:51 am
PL storefront-mf-messaging	PLR storefront-mf-messaging-pr	🟩	Succeeded	Apr 16, 9:45 am
PL storefront-mf-account	PLR storefront-mf-account-pr	🟩	Succeeded	Apr 16, 9:44 am

- Containerize your application
- Modernize your application runtimes
- Refactor your monoliths to microservices using the Strangler pattern
- Separate the frontend and develop micro frontends