Final Project - REPORT

Azure Container Service (AKS) for On-demand Build & Integration Environment

GITHUB: https://github.com/java-stack/sp

Problem Statement:

With the adoption of Agile development, build and test environment provisioning has grown ever challenging to manage. Even large enterprises with large number of environment pools face challenges in re-use and orchestration of multi-version-line concurrent and staggered development lifecycle. Imagine the need of every commit to be able to be fully integration tested in a fully functional environment before allowing to be merged into the release line. This project uses Microsoft Azure Services, in particular, Azure Container Service (AKS), to create, configure, run and manage containerized code build (managing all build-time dependencies) and deployment (provisioning all runtime infrastructures) into an on-demand integration testing environment.

Overview of the Technology:

To solve the above problem, I created a sample application that uses numerous dependencies during build time, in addition uses Kafka and ActiveMQ as runtime infrastructures. The sample application, itself uses web sockets to listen to server that consumes both Kafka and ActiveMQ messages to broadcast. I dockerized the actual build process using maven. I then use docker-compose to create a multi-container composition with separate containers for Kafka, ActiveMQ and the custom application container itself. After testing locally, I registered the container into Microsoft's Container Registry. And finally I used Azure Container Service (AKS) provisioned Kubernetes cluster to deploy the application that would build the code, start Kafka, start ActiveMQ, start the built application connecting to Kafka and ActiveMQ and serve the sample application page.

Detailed Steps:

1. Build the sample app locally and have it running w/ manual start of Kafka and ActiveMQ

The sample application was created using technologies including Java, Spring Boot, Kafka client api, Apache ActiveMQ client api, Bootstrap, Maven. The application front end had a web socket based communication with the server. The backend of the application has 3 major service:

a. Kafka consumer service
b. ActiveMQ consumer service
c. Websocket registration and broadcasting service

It also provided 2 minor convenience service for testing purposes

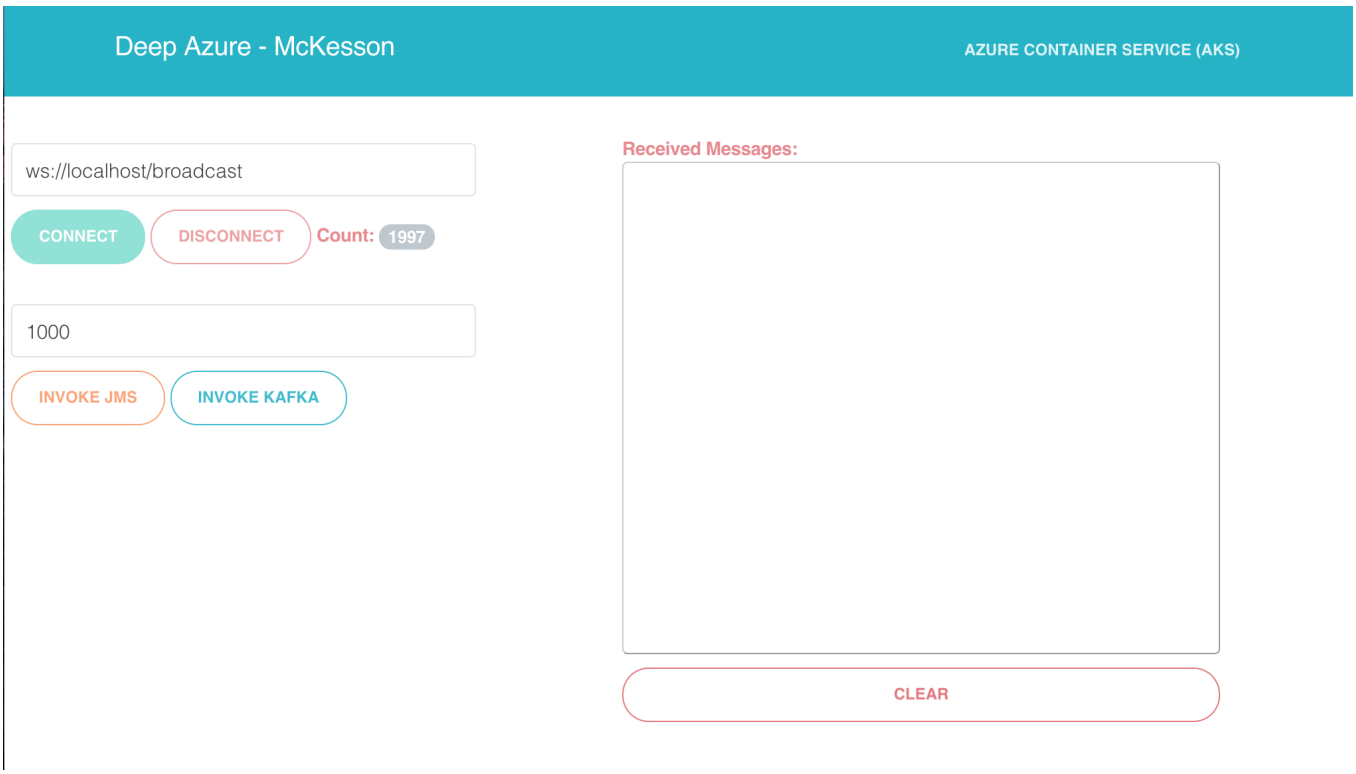
a. Kafka producer service based on an input from client of number of messages to publish
b. ActiveMQ producer service based on input from client of number of messages to publish

To build the application run:

man clean install

Download Apache kafka and install (unzip/untar in appropriate location)
Run zookeeper (in kafka installation dir):
bin/zookeeper-server-start.sh config/zookeeper.properties
Run kafka (in kafka installation dir):
bin/kafka-server-start.sh config/server.properties
Run the server (in application dir):
mvn spring:boot run
or
java -jar target/sp-0.0.1-SNAPSHOT.jar

App screenshot



2. Dockerize the build and start of Step 1 (*Dockerfile xml*)

In this step we create the Dockerfile yaml to dockerize the application. Given the kafka and ActiveMQ is manually started as stated above, you can use the following commands to invoke the docker container:

```
docker volume create --name maven-repo
docker build -t sp-web .
docker run -it -v maven-repo:/root/.m2  sp-web
docker run  -p 80:80 -it -v maven-repo:/root/.m2  sp-web
```

3. Use Docker-Compose to create a multi container application stack (*docker-compose yml*)

In this step we use the docker-compose tool to create a multi-container docker stack:
- SP-KAFKA,
- SP-ActiveMQ,
- SP-WEB,
- a maven repo shared volume, so that the maven artifacts are not downloaded every time

You can use the following commands to start the full stack. Note that this does not need Kafka and ActiveMQ to be separately started. **In fact,** the manual installation Kafka and ActiveMQ should be stopped to avoid port conflicts.

```
docker-compose build
docker-compose up
```

4. Register the container stack into Azure Container Registry (ACR)

The following commands will create the resource group and necessary artifacts and register the local sp-web image into AKS.

```
az group create --name shaqsRg --location eastus
az acr create --resource-group shaqsRg --name shaqsAcr --sku Basic
az acr login --name shaqsAcr
docker tag sp-web shaqsacr.azurecr.io/sp-web:v1
docker push shaqsacr.azurecr.io/sp-web:v1
az acr list --resource-group shaqsRg --query "[].{acrLoginServer:loginServer}" --output table
az acr repository show-tags --name shaqsAcr --repository sp-web --output table
```

```
Shaquilles-MacBook-Pro-2:sp shaq$ docker push shaqsacr.azurecr.io/sp-web:v1
The push refers to repository [shaqsacr.azurecr.io/sp-web]
eca4ce341189: Pushed
25066e10e018: Pushed
acac291b0e7d: Pushed
3809593b25e5: Pushed
3332503b7bd2: Pushed
875b1eafb4d0: Pushed
7ce1a454660d: Pushed
d3b195003fcc: Pushed
92bd1433d7c5: Pushed
f0ed7f14cbd1: Pushed
b31411566900: Pushed
06f4de5fefea: Pushed
851f3e348c69: Pushed
e27a10675c56: Pushed
v1: digest: sha256:f8887497e669c3d8e078d689cee3b4b73a12fbc1c6ddbe779853f651590d4295 size: 3251
```

5. Use Kubernetes Deployment file (*sp-web-all-in-one yml)* to deploy the registered stack into Azure Container Service (AKS)

The following commands will create the AKS cluster and and deploy the sp-web stack

az aks create --resource-group shaqsRg --name shaqsAKSCluster --node-count 1 --generate-ssh-keys
az aks get-credentials --resource-group=shaqsRg --name=shaqsAKSCluster
kubectl get nodes
kubectl create -f sp-web-all-in-one.yml
kubectl get service sp-web —watch

The last command will eventually give a PUBLIC EXTERNAL IP to be accessible via the web.

```
Shaquilles-MacBook-Pro-2:sp shaq$ kubectl create -f sp-web-all-in-one.yml
deployment "sp-kafka" created
service "sp-kafka" created
deployment "sp-activemq" created
service "sp-activemq" created
deployment "sp-web" created
service "sp-web" created
Shaquilles-MacBook-Pro-2:sp shaq$ kubectl get service sp-web --watch
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
sp-web    LoadBalancer   10.0.255.227   <pending>       80:30070/TCP   11s
sp-web    LoadBalancer   10.0.255.227   52.170.116.211  80:30070/TCP   1m
Shaquilles-MacBook-Pro-2:sp shaq$ history
```