Final Project

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

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.

High Level Steps:
1) Build the sample app locally and have it running w/ manual start of Kafka and ActiveMQ
2) Dockerize the build and start of Step 1 (*Dockerfile xml) … test locally*
3) Use Docker-Compose to create a multi container application stack (*docker-compose yml*) SP-KAFKA, SP-ActiveMQ, SP-WEB, a maven repo shared volume, so that the maven artifacts are not downloaded every time *test locally*
4) Register the container stack into Azure Container Registry (ACR)
5) Use Kubernetes Deployment file (*sp-web-all-in-one yml)* to deploy the registered stack into Azure Container Service (AKS)
6) *…Trouble-shoot authentication issues for AKS to talk to ACR*

Data Source: Not Applicable
Hardware Used: Azure Container Service (AKS) binpacking auto-provision
Software Used: Kafka, ActiveMQ, Spring Framework, Maven, Docker and dependencies.
GITHUB: https://github.com/java-stack/sp
YouTube Links:
2 Min: https://youtu.be/NFy2QVOU43w
15 Min: https://youtu.be/jAwrXA96bx0