1. Create Spring boot app needed
2. Create dockerfile (dockerfile.txt) for dockerizing the app

|  |
| --- |
| # Start with a base image containing Java runtime  FROM openjdk:8-jdk-alpine  # Add Maintainer Info  LABEL maintainer="amahilmaran@jetblue.com"  # Add a volume pointing to /tmp  VOLUME /tmp  # Make port 8080 available to the world outside this container  EXPOSE 8080  # The application's jar file  ARG JAR\_FILE=target/<application jar name>  # Add the application's jar to the container  ADD ${JAR\_FILE} <application jar name>  # Run the jar file  ENTRYPOINT ["java","-jar","/<application jar name>"] |

1. Execute the docker command to containerize the application
   1. docker build -t <application name> .
2. Run the container image to test the container
   1. docker run -p <Port to run>:<Exposed Tomcat port> < application name >:latest
3. Create Azure Container Registry (ACR) to store the image create
   1. az acr create --name <ACR name> --resource-group <Resource Group Name> --sku <SKU> --location eastus --admin-enabled true
   2. Make a note of the output of this command as this will have the login servername and name (user id for acr)
4. Get the credential of the Azure Container Registry (ACR) to log into it so that we can push the image from local to ACR
   1. az acr credential show --name <ACR name> --query passwords[0]
5. Login to ACR
   1. docker login <Login server name> -u <user id> -p <password>
6. Create tag name (<Login server name>/<application name>) for the docker image
   1. docker tag <application name> <Login server name>/<application name>
7. Push the image to ACR
   1. docker push <tag name>
8. Create a service principal for creating AKS
   1. az ad sp create-for-rbac --name <service principal name> --password <Service Principal password>
   2. Please make a note of the output of this command as we need to use this for creating AKS
9. Create AKS
   1. az aks create --resource-group <Resource group name>  --name <AKS name> --service-principal "<Service principal app id>" --client-secret "<password from service principal>" --no-ssh-key --location eastus --disable-rbac --node-vm-size <VM size>
10. Get AKS credentials in kubectl so that we can execure kubectl commands
    1. az aks get-credentials --name <AKS name> --resource-group <Resource group name>
11. Create secrets to access ACR so that the container images can be pulled.
    1. kubectl create secret docker-registry <secret name> --docker-server <ACR login server name>  --docker-username <ACR user name> --docker-password <ACR password> --docker-email <email id>
    2. The <secret name> create will be used in manifest yaml file used for deployment
12. Create manifest yaml (masterdatasecured.yaml) for deploying the container images to AKS

|  |
| --- |
| apiVersion: apps/v1beta1  kind: Deployment  metadata:  name: <Deployment Name>  labels:  app: <Application name>  spec:  replicas: 1  selector:  matchLabels:  app: <Application name>  template:  metadata:  labels:  app: <Application name>  spec:  containers:  - name: <container name 1>  image: <image tag name 1>  ports:  - containerPort: <port for Container>  env:  - name: server.port  value: <exposed port>  - name: <container name 2>  image: <image tag name 2>  ports:  - containerPort: <port for Container>  env:  - name: server.port  value: <exposed port on which container 1 calls container 2>  - name: <App environment variable>  value: <Environment variable value>  imagePullSecrets:  - name: <Secret name>  --- |

1. Deploy pod using Yaml
   1. Kubectl apply -f <yaml file path> eg “./masterdatasecured.yaml”
2. Expose the pod as a service type loadbalancer (the server will be exposed externally)
   1. Kubectl expose deployment deploymentName –type=LoadBalancer –port=80 –target-port=<exposed port>
      1. In JetBlue subscription this loadbalancer creation will fail as this will expose public external IP which is not allowed.

1. To create ingress controller.
   1. helm install stable/nginx-ingress  --namespace kube-system  -f ingress.yaml --set controller.replicaCount=2 --set rbac.create=false --name ingress-controller
   2. The ingress.yaml file will be as below.

|  |
| --- |
| controller:  service:  loadBalancerIP: <IP in the AKS subnet>  annotations:  service.beta.kubernetes.io/azure-load-balancer-internal: "true" |

1. Check the pod status and service status
   1. kubectl get pods -n kube-system
   2. kubectl get svc -n kube-system
2. Create the ingress rule for the service
   1. kubectl apply -f ingress-rules.yaml

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: <Application name – The one in manifest yaml (masterdatasecured.yaml)>  annotations:  kubernetes.io/ingress.class: nginx  nginx.ingress.kubernetes.io/ssl-redirect: "false"  nginx.ingress.kubernetes.io/rewrite-target: /  spec:  rules:  - http:  paths:  - path: /  backend:  serviceName: <Service name – Deployment name in manifest yaml (masterdatasecured.yaml) >  servicePort: 80 |

1. *When we are connected to the AKS subnet, use the Ingress load balance IP address provided in the ingress.yaml to access the deployed service*
2. *I am doing ingress port forwarding for this demo as I am not on the VDI to connect to the AKS subnet.*
   1. *kubectl port-forward <ingress pod name> <local port name to call from local>:80 -n kube-system*
   2. *We can get the ingress pod name from the command “kubectl get pods -n kube-system”*