Document : Kubernetes Deployment to AKS

Task: Deployment of the containerised application to the Kubernetes cluster that's been created. To achieve this, the essential Kubernetes manifest must be created and provisioned.

Steps:

1. Started by creating a Kubernetes manifest file, named application-manifest.yaml. Inside this file, began by defining the necessary Deployment resource, which will help deploy the containerized web application onto the Terraform-provisioned AKS cluster. The manifest included the following:

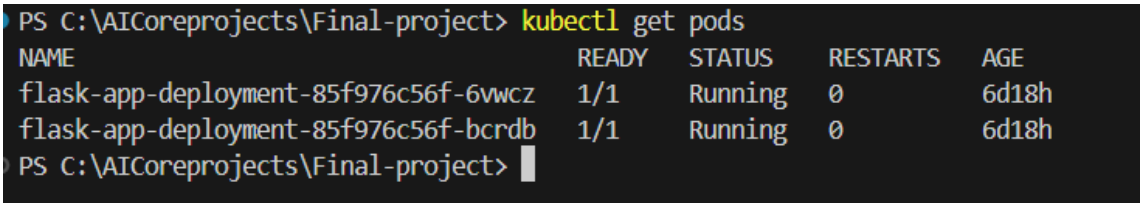
    a. Defined a Deployment named flask-app-deployment that acts as a central reference for managing the containerized application.
    b. Specified that the application should concurrently run on two replicas within the AKS cluster, allowing for scalability and high availability.
    c. Within the selector field, used the matchLabels section to define a label that uniquely identifies the application. Used the label app: flask-app. This label will allow Kubernetes to identify which pods the Deployment should manage.
    d. In the metadata section, defined labels for the pod template. Reused the label app: flask-app. This label is used to mark the pods created by the Deployment, establishing a clear connection between the pods and the application being managed.
    e. Configured the manifest to point to the specific container housing the application, which should be already hosted on Docker Hub (whule/flask-app:latest). This ensures that the correct container image is utilized for deployment.
    f. Exposed port 5000 for communication within the AKS cluster. This port servers as the gateway for accessing the application's user interface, as defined in the application code.
    g. Implemented the Rolling Updates deployment strategy, to facilitate seamless application updates. Ensured that, during updates, a maximum of one pod deploys while one pod becomes temporarily unavailable, maintaining application availability.
2. Added a Kubernetes Service manifest to the existing application-manifest.yaml to facilitate internal communication within the AKS cluster using the --- operator in between the deployment and services configurations sections of the manifest. This manifest would achieve the following key objectives:

    a. Defined a service named flask-app-service to act as a reference for routing internal communication.
    b. Ensured that the selector matches the labels (app: flask-app) of the previously defined pods in the Deployment manifest. This alignment guarantees that the traffic is efficiently directed to the appropriate pods, maintaining seamless internal communication within the AKS cluster.
    c. Configured the service to use TCP protocol on port 80 for internal communication within the cluster. The targetPort was set to 5000, which corresponds to the port exposed by the container.
    d. Set the service type to ClusterIP, designating it as an internal service within the AKS cluster.
3. Deployed the Kubernetes manifest to the AKS cluster making sure to be in the correct context, namely the Terraform-provisioned AKS cluster. This is important as the context

specifies the cluster and user details for your AKS cluster, ensuring the deployment occurs in the intended environment.

a. With the correct context set, proceed with the deployment of the Kubernetes manifests. Apply the manifest and monitor the command-line interface for feedback on the deployment process. Us this command to apply the manifest: kubectl apply -f path/to/manifest.yaml

b. After the deployment has successfully completed, verified the status and details of the deployed pods and services to ensure the pods are running. Use the following command to get the pods status: kubectl get pods



4. Tested and validated the deployment to the AKS cluster. Since the application in development is an internal tool designed for the company's employees and is not intended for external users, given its internal nature, could only assess the deployment by performing port forwarding to a local machine following these steps:

   a. Verified the status and details of the pods and services within the AKS cluster as in the previous step. Ensuring that the pods are running, and the services are correctly exposed within the cluster.
   b. Next, initiated port forwarding using the kubectl port-forward <pod-name> 5000:5000 command. Replaced <pod-name> with the actual name of the pods to connect to. This command establishes a secure channel to the application, allowing for a possibility to interact with it locally.
   c. With port forwarding in place, the web application hosted within the AKS cluster was accessed locally at http://127.0.0.1:5000
   d. Visited this local address and thoroughly tested the functionality of the web application paying particular attention to the orders table, ensuring that it was displayed correctly and was able to use the Add Order functionality successfully.
5. Finally, added the terraform state files to .gitignore and pushed the latest IaC file to github. First while on the main project directory (aks-terraform-main) added the files to git (git add <file name>), then commit the files (git commit -m <"description of action">). And git push command to push to git hub.