

Documentation

Step 1) Create a Docker file, which is a set of instructions used to build docker container image.

```
FROM python:3.12 #Docker Base image for container
LABEL Maintainer = "Najam" #Maintainer of image
WORKDIR /App #Sets working directory inside container to /App
COPY requirements.txt ./ #Copys .txt file to working directory inside container
RUN pip3 install -r requirements.txt #Runs python command inside the container
COPY / . #Copies all files to working directory inside the container
EXPOSE 80 # Maps to container port 80
CMD ["py", "App.py"] #Executes following command when container starts up
```

Step 2) Create the configuration files including Deployment and service.

```
apiVersion: apps/v1 #Use Kubernetes version 1 for deployment
kind: Deployment
metadata:
  name: appdeploy #Specify name of the deployment
spec:
  replicas: 2 #Number of instances of application (here it is two)
  selector: #How pod should be selected for deployment
    matchLabels:
      app: myapp #Key-Value pair for pod management
  template: #Pod template for deployment
    metadata:
      labels:
        app: myapp
    spec:
      containers: #Specify container name and image to use
```

```

- name: your-container-name # Provide a name for your container
  image: your-image-name:tag # Provide the name of your Docker image along with the tag
  ports:
    - containerPort: 80 #Expose to container port 80
---
apiVersion: v1 #Kubernetes version 1 for service
kind: Service
metadata:
  name: appserv #Name of service
spec:
  selector: #Pod the service should target
    app: myapp
  ports:
    - protocol: "TCP" #Uses Transmission Control Protocol
      port: 80 # Port on which service will be available (Internal cluster port)
      targetPort: 80 #Port maps 1-1 with container port
  type: LoadBalancer #Uses loadbalancer to expose service externally

```

Step 3) Create and set up a new Azure DevOps Account, for CICD pipeline.

Step 4) Create a New Project

Step 5) Set up a new Azure Repository and push source code from the local repository to Azure Repos using the following Git commands.

```

git init – initialize working directory
git clone <Azure repository-url> - Clone the Azure Repository url to local repository
git add . - Stage all files
git commit -m "Your descriptive commit message here" - Commit stage changes with message
git push origin <branch-name> - Push commit changes to Azure Repos branch

```

Step 6) Set up CICD Pipeline

- A) Go to Pipeline -> New Pipeline
- B) Select Azure Repos to get the latest code
- C) Select the Repository
- D) Configure Build and Deploy Pipeline (i.e. Build & Deploy to AKS)
- E) Set up variables and service connections with ACR, Docker Registry etc...
- F) Build Artifacts produced by build pipeline are used in Deploy stage

```
trigger: # trigger the main branch
```

```
- main
```

```
resources:
```

```
- repo: self
```

```
variables: # define variables to be used throughout the pipeline
```

```
  tag: '$(Build.BuildId)'
```

```
  repository: 'app-2'
```

```
stages: #Logical Boundary in Azure Pipeline
```

```
- stage: Build # Build Stage
```

```
  displayName: Build Stage
```

```
  jobs: # Consists of one or more steps
```

```
  - job: Build
```

```
    displayName: Build #Build job
```

```
    pool:
```

```
      vmImage: ubuntu-latest # Microsoft hosted agent to run the CICD pipeline
```

```
  steps: # Contains one or more task
```

```
- task: Docker@2 #Docker task to build & Push image to ACR
```

```
  displayName: Build & Push to ACR
```

```
  inputs:
```

```
    containerRegistry: 'Azure Container Registry'
```

```
    repository: '$(repository)' #Repository in ACR to host image
```

```
    command: 'buildAndPush'
```

```
    Dockerfile: '**/Dockerfile'
```

```
    container: 'container1'
```

```
    tags: | #Specify latest tags for the image
```

```
      $(tag)
```

```
      latest
```

```
- task: PublishBuildArtifacts@1 #Build Artifacts produced from build stage
```

```
  inputs:
```

```
PathToPublish: '$(System.ArtifactsDirectory)'
ArtifactName: 'manifests'
publishLocation: 'Container'
```

- stage: SecurityScan #Implementing Security stage for source code practice/compliance
displayName: Static Application Security Testing
dependsOn: Build #Depends on Build Stage
jobs:
 - job: BanditScan #Analyis tool for python code
displayName: Run Bandit SAST
pool:
vmImage: ubuntu-latest
steps:
 - task: UsePythonVersion@0 #Uses python version for the task
inputs:
versionSpec: '3.x'
addToPath: true
 - script: |
pip install bandit #Installs bandit tool
bandit -r \$(Build.SourcesDirectory) #Runs bandit on source code in Repos
displayName: Run Bandit SAST
- stage: Deploy #Deploy stage
displayName: Deploy
dependsOn:
 - SecurityScan
jobs:
 - job: Deploy
displayName: Deploy to AKS
pool:
vmImage: ubuntu-latest
steps:
 - task: Kubernetes@1 #Uses Kubernetes v1 to deploy application to AKS
inputs:
connectionType: 'Kubernetes Service Connection'
kubernetesServiceEndpoint: 'Kubernetes Service Connection'
namespace: 'default'
command: 'apply' # applies the config files using kubectl apply
useConfigurationFile: true
configuration: 'Deployment.yml'
secretType: 'dockerRegistry'
containerRegistryType: 'Azure Container Registry'

Step 7) Save and commit to master branch in-order to trigger the pipeline