CLOUD-NATIVE PYTHON APP ON AWS EKS

TECHNICAL GUIDE

Prerequisites:

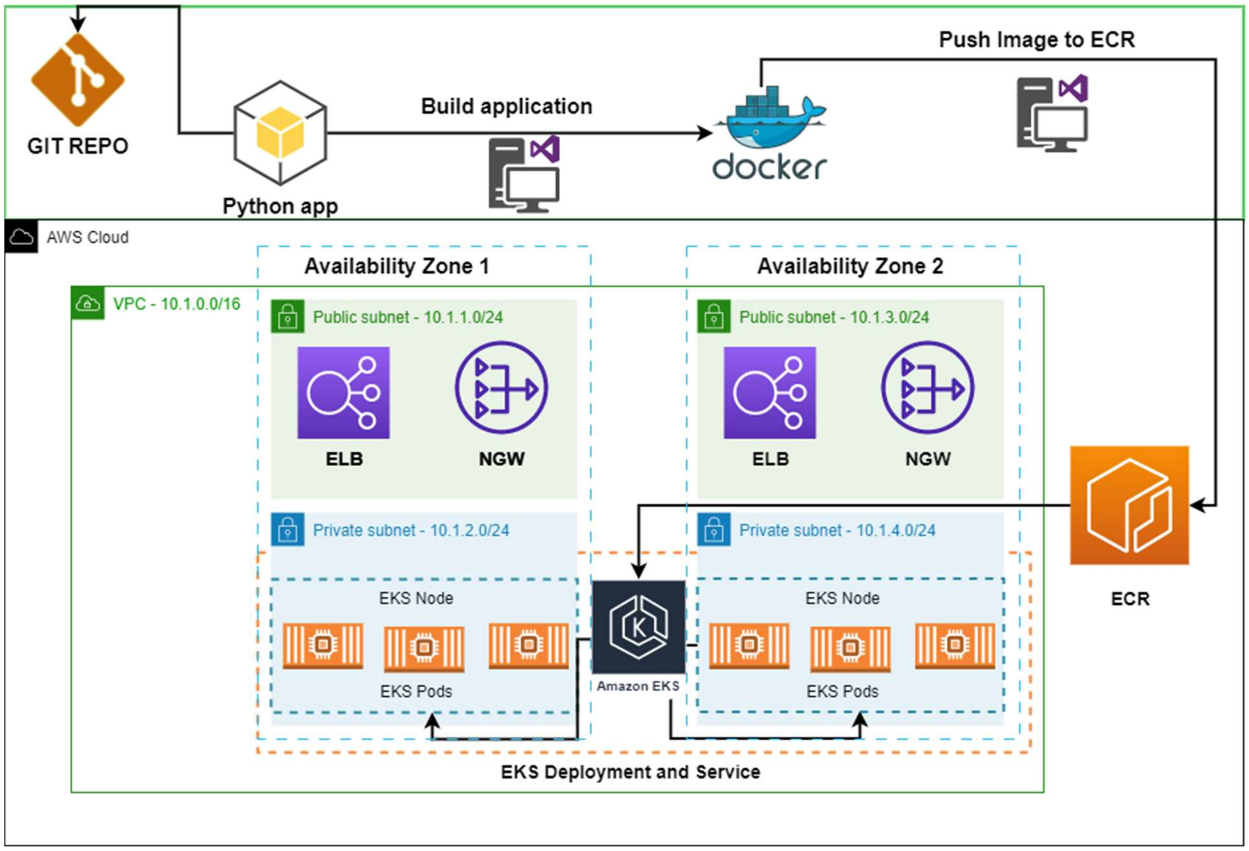
General Requirements:

1. Microsoft Visual Studio Code – Download Visual Studio Code - Mac, Linux, Windows
2. AWS Account with Programmatic Access (Access and Secret Key) – Watch the Video for steps
3. Python Installation - Download Python | Python.org
4. Docker Installation - Install Docker Engine | Docker Documentation

NB: The general requirements apply to all Operating Systems

Windows Users:

1. Install Microsoft C++ Build Tools - Microsoft C++ Build Tools - Visual Studio
2. Enable Virtualization in your BIOS Settings (Needed to run Docker Desktop)
3. RUN Docker Desktop Using WSL (Windows Subsystem for Linux)
4. Open Powershell and run *wsl --update*
5. Then you can launch your Docker Desktop and Start the VSCode Editor.



PYTHON APP DEPLOYMENT ON AWS EKS

STEP 1

To start, create a new folder and open it using VSCode. Create a new file with the name **requirements.txt** and paste the code below-

```
Flask
MarkupSafe
Werkzeug
itsdangerous
psutil
plotly
tenacity
boto3
kubernetes
```

Before running the above script, check the following (Windows User):

```
python --version
Python 3.10.7
pip --version
pip 23.1 from C:\Users\hp\AppData\Local\Programs\Python\Python310\lib\site-packages\pip
(python 3.10)
```

Before running the above script, check the following (Linux and Mac Users):

```
Python3 --version
Python 3.10.6
pip --version
pip 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10)
```

If the above results are returned, then install the requirements file using the following commands:
Windows:

```
pip install -r .\requirements.txt
```

Linux and Mac

```
pip3 install -r requirements.txt
```

STEP 2

Authenticate with your AWS Account using your access and secret key; run the following commands:

```
aws configure
AWS Access Key ID [****************WZHI]: #Replace with your access key
AWS Secret Access Key [****************T+ys]: #Replace with your secret key
Default region name [eu-west-2]: #Choose your own region
Default output format [None]: #You can leave this as none
```

NB: Ensure your IAM Account as an AdministratorsAccess Policy attached. **Do not use your root account.**

Then create a new file within your VSCode named **ecr.py** (to create an Elastic Container Registry for our Docker Image). Copy the script below and run it using **python ecr.py(Windows) or python3(Linux and MacOS) ecr.py in your VSCode terminal**

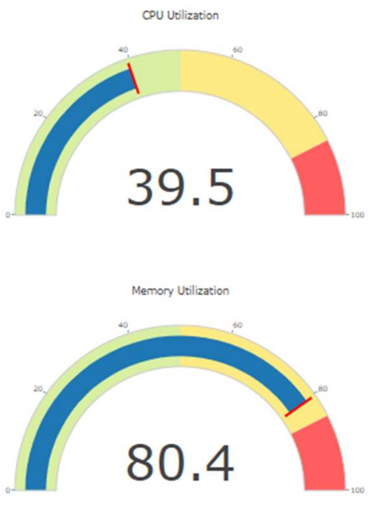
```
import boto3

client = boto3.client('ecr')

response = client.create_repository(
    repositoryName='flask-monitoring-app', # Any name of your choice
    tags=[
        {
            'Key': 'flask-app-docker_image', # Any name of your choice
            'Value': 'flask-monitoring-app'  # Any name of your choice
        },
    ],
    imageScanningConfiguration={
        'scanOnPush': True
    }
)

ecr_uri = response['repository']['repositoryUri']
print(f"ECR URI=: {ecr_uri}")
```

STEP 3

Create a new file named **app.py** copy and run the script below locally to test your application:

```
import psutil
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    cpu_metric = psutil.cpu_percent()
    mem_metric = psutil.virtual_memory().percent

    Message = None
    if cpu_metric > 80 or mem_metric > 80:
        Message = "High CPU or Memory Detected, scale up!!!"
    return render_template("index.html", cpu_metric=cpu_metric, mem_metric=mem_metric,
message=Message)


if __name__=='__main__':
    app.run(debug=True, host = '0.0.0.0')
```

If you have this page, Congratulations!



STEP 4

Package application code into a Docker Image using a **Dockerfile**. Create a new file named **Dockerfile** and copy the below and paste.

```
# Use the official Python image as the base image
FROM python:3.11.3-buster

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file to the working directory
COPY requirements.txt .

# Install the required Python packages
RUN pip3 install --no-cache-dir -r requirements.txt

# Copy the application code to the working directory
COPY . .

# Set the environment variables for the flask app
ENV FLASK_RUN_HOST=0.0.0.0

# Expose the port on which the Flask app will run
```

```
EXPOSE 5000

# Start the Flask app when the container is run
CMD ["flask", "run"]
```

Use the following command to build a Docker image:

```
docker build -t flaskapp .
```

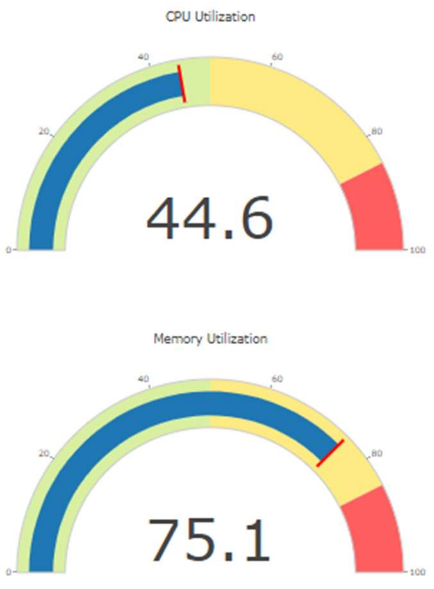
Check the new image using the following command:

```
docker images
```

To run the docker image and confirm if the app is running as it was locally, run the following command:

```
docker run -d -p5000:5000 flaskapp:latest
```

You should have the following result:



Push the Docker image into the ECR repo created in **Step 2**. On your ECR page, click on the repo and ***view push commands;*** it shows you steps to tag your docker image and push it to the repository. You can watch the live demo video for more clarity.

STEP 5

Deploy the application on AWS EKS Cluster.

First, you need to create two IAM Roles.

1. Amazon EKS Cluster IAM Role
   https://docs.aws.amazon.com/eks/latest/userguide/service_IAM_role.html#create-service-role
2. Amazon EKS Node IAM Role

https://docs.aws.amazon.com/eks/latest/userguide/create-node-role.html

Once you have the two roles created, create your EKS Cluster; watch the live demo video for a step-by-step guide. To connect to your EKS cluster, run the following command to check that you installed kubectl correctly.

Note: Before deploying your Kubernetes cluster, ensure your VPC and subnets are appropriately configured. Also, Create a security group and add an incoming rule allowing port 5000.

```
kubectl version
================
Output
=========
WARNING: This version information is deprecated and will be replaced with the output from
kubectl version --short.  Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.4",
GitCommit:"872a965c6c6526caa949f0c6ac028ef7aff3fb78", GitTreeState:"clean", BuildDate:"2022-
11-09T13:36:36Z", GoVersion:"go1.19.3", Compiler:"gc", Platform:"windows/amd64"}
Kustomize Version: v4.5.7
Unable to connect to the server: dial tcp: lookup 37504CB7D5082E9CCCCD44EDF886D54A.yl4.eu-
west-2.eks.amazonaws.com: no such host
```

If you have the output above, then run the following command to connect your **kubectl client** to the AWS EKS cluster:

```
aws eks update-kubeconfig --name flask-app-kube-cluster # Replace with the name of your
cluster
```

Then run the command below again; your output should be different from the previous

```
kubectl version
```

Create your Kubernetes deployment and service using a Python script. First, create a file named **eks.py,** and copy and paste the script below, then run **python eks.py (Windows) or python3 eks.py (Linux and Mac)**

```
#create deployment and service
from kubernetes import client, config

# Load Kubernetes configuration
config.load_kube_config()

# Create a Kubernetes API client
api_client = client.ApiClient()

# Define the deployment
deployment = client.V1Deployment(
    metadata=client.V1ObjectMeta(name="my-flaskapp"), #Change the name
    spec=client.V1DeploymentSpec(
        replicas=1,
        selector=client.V1LabelSelector(
            match_labels={"app": "my-flaskapp"} #name must match metadata name above
        ),
        template=client.V1PodTemplateSpec(
            metadata=client.V1ObjectMeta(
                labels={"app": "my-flaskapp"} #name must match metadata name above
            ),
            spec=client.V1PodSpec(
                containers=[
                    client.V1Container(
                        name="my-flaskcontainer", #name of your container
                        image="392102158411.dkr.ecr.eu-west-2.amazonaws.com/flask-app-ecr-
repo", #URI of your docker image in ecr
                        ports=[client.V1ContainerPort(container_port=5000)]
                    )
                ]
            )
        )
    )
)

# Create the deployment
api_instance = client.AppsV1Api(api_client)
api_instance.create_namespaced_deployment(
    namespace="default",
    body=deployment
)

# Define the service
service = client.V1Service(
    metadata=client.V1ObjectMeta(name="my-flask-service"), #choose your own name
    spec=client.V1ServiceSpec(
        selector={"app": "my-flask-app"}, #choose your own name
        ports=[client.V1ServicePort(port=5000)]
    )
)

# Create the service
api_instance = client.CoreV1Api(api_client)
api_instance.create_namespaced_service(
    namespace="default",
    body=service
)
```

Run the following commands to check your deployments, pods, and service:

```
kubectl get deployments -n default
kubectl get pods -n default
kubectl get svc -n default
```

Deployments – To check if your container ran successfully
Pods – To check if the pods were successfully deployed on your nodes
SVC – To check that your container port is open to network communication.

To communicate with your app on AWS EKS locally, run the following command:

```
kubectl port-forward svc/<nameofservice> <port:port>
```

The above command lets you communicate with your EKS app using your localhost.
NB: When you run ***kubectl get svc -n default*** – you get the service name and port number

Note: The **eks.py** script above allows you to interact with your app locally. To expose your app to the public (internet access), you need to configure the service type to use a loadbalancer. To do that replace the ***Define the service*** section in the **eks.py** code with these:

```
# Define the service
service = client.V1Service(
    metadata=client.V1ObjectMeta(name="my-flaskservice"),
    spec=client.V1ServiceSpec(
        selector={"app": "my-flaskapp"},
        type="LoadBalancer",
        ports=[client.V1ServicePort(
            port=5000,
            target_port=5000)]
    )
)
```

NB: Before running the edited **eks.py** ensure you delete your initial deployment and service; use the following commands:

```
 kubectl delete deployments -n default

kubectl delete svc <servicename> -n default
```

When a deployment is deleted, all associated pods are deleted automatically. You can verify the deletion by using the following commands:

```
kubectl get deployments -n default
kubectl get pods -n default
kubectl get svc -n default
```

If all is clear, rerun **python eks.py** and check the status of your deployment. To get the loadbalancer URL use the command below:

```
kubectl get svc -n default
```