

# How to build an application and deploy it on Kubernetes

-By:Maha

## What we are doing-

1. How to create a monitoring application in Python using flask
2. Start by building the application and then containerize it using Docker
3. Create a docker file, build the image, and run the container locally
4. Then create an ECR using the Python boto3 module, in ECR, we will push the docker image to store, retrieve, and use the docker images in a secure and efficient manner
5. Next in the deployment phase we will create an elastic Kubernetes cluster with nodes and deploy the application on Kubernetes, we will create deployment and service using Python, so that our application can be accessed from the internet that is deployed in Kubernetes.

Tools: Python, Kubernetes, Docker, Amazon ECR, Python+boto3

Lets start!

## Prerequisites:

- ✓ AWS account  
**Username:**  
**Password:**
- ✓ Programmatic access and AWS configured with CLI  
In IAM-> select/create a user with Administrator permission, login from the user, IAM->select you user->security credentials, and then click on Access keys to get programmatic access

**Login root account/Alias:**

**IAM user:**

**Password:**

**Access key ID:**

**Secret access key:**

To configure AWS: ( do it in VS code)

**aws configure**

enter Access key, secret access key, and region as us-east-1, output none as default, and enter, to cross-check just run the list IAM command

**aws iam list-users**

- ✓ Python3 installed: to check if you have Python installed enter:  
**python3 --version**  
you should give the version as output: Python 3.8.10; if you don't have it installed, then just check for the version in Google and install it
- ✓ Docker and Kubectl installed  
You can download docker and kubectl from internet, for windows you can download dockerdesktop but ensure to enable bios svm(virtualmachine) so that your docker engine can run.  
Note: kubectl will default to be downloaded in downloads, so after it is downloaded, go to cmd in downloads and run the below commands to check.  
**kubectl --help**  
**kubectl version**  
**kubectl version --client=true**

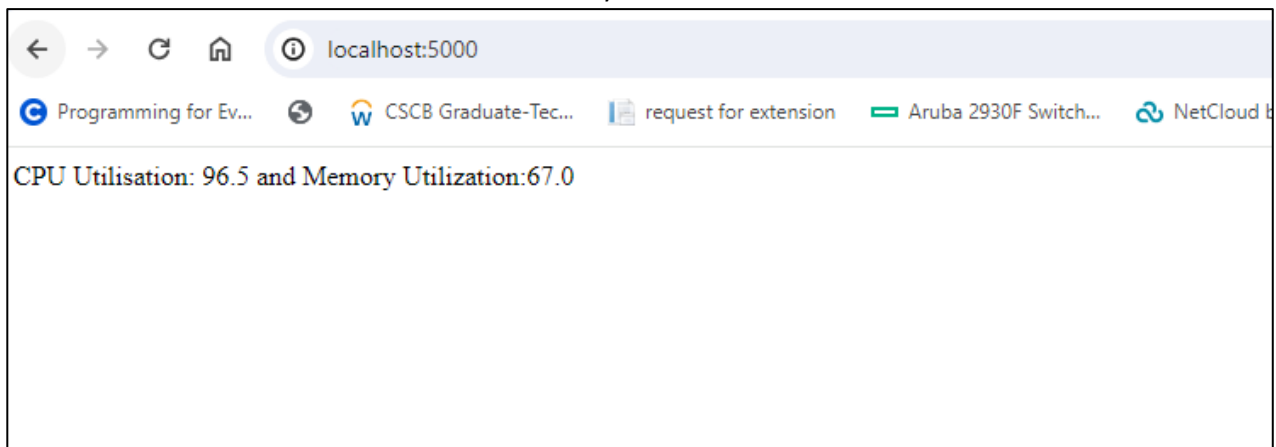
Once you check it move it to c folder

- ✓ Code editor (vscode)  
Download vscode

Now that we have the prerequisites, lets reaaally start! 😊

Building application and running in local machine:

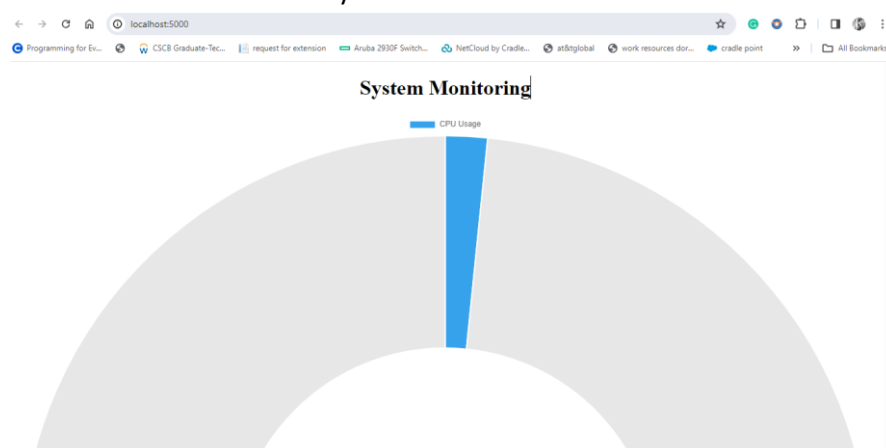
1. Create a folder called "cloud\_native\_monitoring\_app" in your home directory. We will be using this folder to create all your files here.
2. Now open vs code, and open this folder in vs code
3. So the first file that we are going to create is the "app.py" file. So click on the folder on the left-hand side and select the icon for the new file. Name the file as "app.py" as mentioned.
4. The app that we are creating is a monitoring app that is going to get the CPU and memory metric. In Python, you have different modules to do some tasks. If you want to get CPU and metric, then we need to import "psutil". This is a cross-platform library for retrieving information on running processors and system utilization (CPU, memory, disk, network sensor) in Python
5. Next, we need to import the flask module, this is need to create the application itself
6. Now to create the app, we use the flask and give the `"__name__"`
7. Next, we set the `@app.route` as `"/"` which means, the app will run when the user is at the home path, the application will run.
8. Now about what it should show, lets create a function here called index
9. In the function, we are going to define 3 variables.: `cpu_percent` ( holds the value of CPU usage)and `mem_percent`( holds virtual memory in the form of a percentage) and we are giving the message as none.
10. But let's include a functionality here such is memory or CPU is higher than 80%, then we should get the message that says, memory is high or CPU is high.
11. This function is not returning anything, so lets get a return statement as well.
12. Let's say if name of app is equal to main, then then run the app, so set `debug=True` and give the host as `'0.0.0.0'`, so that it can run in our local machine.
13. Now if you try to run the file you will get an error saying module not found, so for that we just need to give the pip3 install and then mention the module as shown below  
`pip3 install psutil`  
`pip3 install Flask`
14. so instead for sitting and running each of theses modules. Just create a requirements.txt" file and paste the below modules as shown below:
15. then you can just run one command  
`pip3 install -r requirements.txt`
16. now run the python file  
`python3 app.py`
17. you will see the app running on port 5000 in local host. So go to chrome and type localhost:5000. You will be able to see the CPU utilisation and memory utilisation:



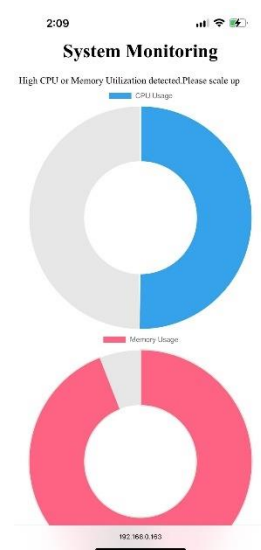
And every time you run , it or say refresh the chrome page, you will get the log in the terminal of vs code like below

```
PS D:\maha\my documents\material\Devops Projects\cloud_native_monitoring_app> python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.163:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 983-852-039
127.0.0.1 - - [31/Dec/2023 03:43:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:43:57] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [31/Dec/2023 03:46:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:48] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:48] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 03:46:48] "GET / HTTP/1.1" 200 -
```

18. Now to beautify the file let's use a html doc. So in our app.py, we need to make a few changes.
19. First, we need to import render\_template along with flask and then we need to change the return statement to point to the html file, along with the CPU and memory value.
20. Then create a new folder called “templates” and inside that folder create an index.html and paste the code given below. If you see the html file, we can see that we are setting the style as a gauge or indicator to show the CPU and memory utilization.



21. You can also see it from any device entering the ip 192.168.0.163:5000



Now we have our application running in the local machine, do we need to containerize this application now.

1. So create a file called docker file, so that it can create an image and run the application in the form of container.
2. Since we are running python application here, we will need a python image, if you go to the docker hub and type python, you will find the official python image and inside that there are multiple versions, we will be using the image with version "3.9-slim-buster". So use FROM to set the base image
3. Next we need to set the working directory where the container will be running so set the WORKDIR as /app. All the commands will be run in this directory
4. Now lets copy everything in this directory to the /app directory mentioned here, so you can just give COPY . /app
5. RUN is the parameter to execute any command on top of the current image. So use it to run the pip3 install command to install the dependencies in the requirement.txt with no cache.
6. So use ENV command to set an environment variable NAME as World
7. Now we need to expose the port 5000. So, give the command EXPOSE 5000.
8. Now we need to command to be running so for that give the command CMD ["python", "app.py"], to run the python file app.py as soon as the container starts.
9. Now we need to build an image out of it to run it.
10. So for that run the below command in vs code  
**docker build -t my-flask-app .**
11. Here we are creating an image name my-flask-app and the "." to use the docker file present inside the "cloud\_native\_monitoring\_app" directory
12. Next we need to run the docker run command to create the docker container, so here we define the port as 5000:5000 which basically means when the container runs, it should run the app in your localhost port 5000 as well, then we mention the image name or the image ID which we can get when we type the command docker images. You will get the monitor application web page  
**docker images**  
**docker run -p 5000:5000 <imageID>**

Now we go ahead and push the image to docker hub/ECR and then deploy it as a ECR Kubernetes cluster

1. Go to amazon ECR, you can easily create repository in the ECR page, by just clicking create repository and giving name, but that is not done, so let's go and create in VS code using python.
2. So we will use python+boto3 module
3. So when we are going to first use boto3, we need to import boto3 and then mention ECR as the client as shown below
4. `import boto3`
5. `client = boto3.client( 'ecr' )`
6. you can find more information about this and the available API actions in Python+ boto3, in the boto3 documentation provided by AWS
7. <https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/index.html>
8. So create a python file called "ecr.py" and import boto3 and set client as 'ecr'
9. Give a name to the repository "my-cloud-native-repo"
10. Then we need to set response to run the create repository API and give the repository name as attribute
11. Then we need the repository URI, so use the response to get the repository uri and then print it out
12. Then you can run the python file and you will see the uri as the output, in AWS you will also be able to see a Repository created successfully
13. `python3 ecr.py`
14. now in your amazon console in ecr, if you select your repo, you will find it empty, but on the side, you can see the option that says view push command
15. Retrieve an authentication token and authenticate your Docker client to your registry.  
Use AWS Tools for PowerShell:  
(Get-ECRLoginCommand).Password | docker login --username AWS --password-stdin  
279646452254.dkr.ecr.us-east-1.amazonaws.com

This command didn't work for me, needs some AWS tool package installed...however and linux cli command worked for my windows terminal, weird!!

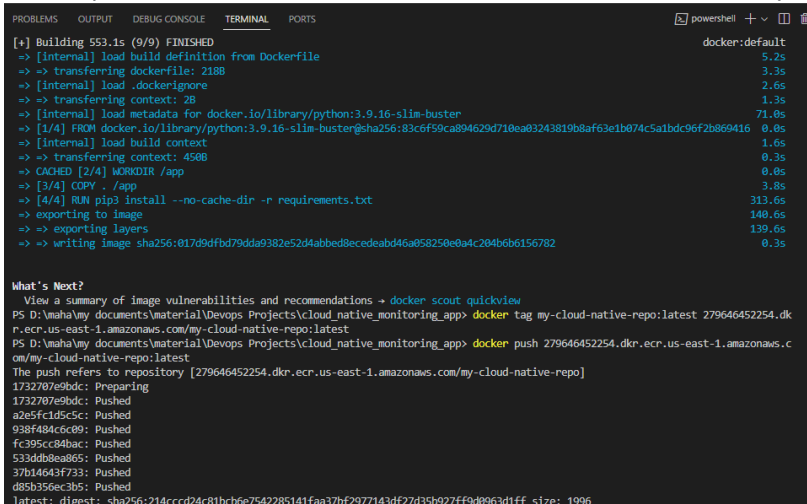
```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 279646452254.dkr.ecr.us-east-1.amazonaws.com
```

16. Build your Docker image using the following command. This will build an image from your docker file that you have created

```
docker build -t my-cloud-native-repo .
```

17. After the build completes, tag your image so you can push the image to this repository:  
docker tag my-cloud-native-repo:latest 279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-native-repo:latest

18. Run the following command to push this image to your newly created AWS repository:  
docker push 279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-native-repo:latest

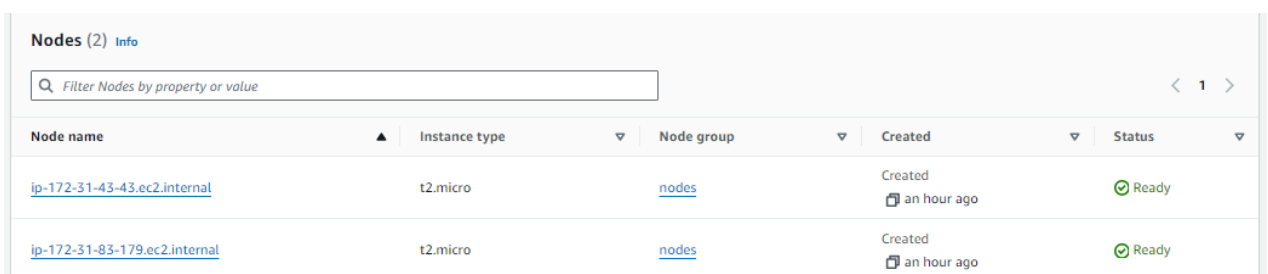


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[+] Building 553.1s (9/9) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 218B
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [internal] load metadata for docker.io/library/python:3.9.16-slim-buster
-> [1/4] FROM docker.io/library/python:3.9.16-slim-buster@sha256:83c6f59ca894629d710ea83243819b8af63e1b074c5a1bdc96f2b869416
-> [internal] load build context
-> => transferring context: 450B
-> CACHED [2/4] WORKDIR /app
-> [3/4] COPY . /app
-> [4/4] RUN pip3 install --no-cache-dir -r requirements.txt
-> exporting to image
-> exporting layers
-> writing image sha256:017d9dfbd79dda9382e52d4abbed8ecedeabd46a058250e8a4c204b6b6156702
What's Next?
View a summary of image vulnerabilities and recommendations -> docker scout quickview
PS D:\maha\my documents\material\Devops Projects\cloud_native_monitoring_app> docker tag my-cloud-native-repo:latest 279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-native-repo:latest
PS D:\maha\my documents\material\Devops Projects\cloud_native_monitoring_app> docker push 279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-native-repo:latest
The push refers to repository [279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-native-repo]
1732707e9bdc: Preparing
1732707e9bdc: Pushed
a2e5fc1d5c5c: Pushed
938f4846c09: Pushed
fc395cc84bac: Pushed
533ddb8ea865: Pushed
37b14643f793: Pushed
085b396ec3b5: Pushed
latest: digest: sha256:214cccd24c81bcb6e7542285141faa37bf2977143df27d35b927ff9d0963d1ff size: 1996
```

19. Once it is done, you should be able to see our image in the repository

Now lets go to EKS to create our Kubernetes cluster

1. Go to Amazon EKS, and click on Create Cluster.
2. Give a name for the cluster "cloud-native-cluster, leave the default 1.28Kubernetete Kubernetes version
3. For the cluster service role create an IAM role for EKS (aws type->EKS cluster -> awsekspolicy->name->create) and select that IAM role as a cluster service role
4. no tags or encryption-> click on next
5. select your VPC, and subnet, and when you give security group, ensure that port 5000 is given
6. click on next, we don't want any loggings or add on -> so click on next ->next and create the cluster.
7. Now if in vs terminal, if you try kubectl get pods -n default, you will get unable to connect to server error, that's because the cluster is not created and after it is created, we need to authenticate and connect using the command eks config
8. Once the cluster is created, select the cluster, compute scroll down and click on create node group
9. Give a name for the node group(nodes)->attack an IAM role (create like before) ->go ahead and click on next
10. In the next, select, AMI,t2.micro, on demand, desiredsize, minimumsize, maximumsize =>2->next
11. Select the subnets->next->create, this will create node group having 2 nodes inside it



Nodes (2) Info					
Filter Nodes by property or value					
Node name	Instance type	Node group	Created	Status	
<a href="#">ip-172-31-43-43.ec2.internal</a>	t2.micro	<a href="#">nodes</a>	Created an hour ago	Ready	
<a href="#">ip-172-31-83-179.ec2.internal</a>	t2.micro	<a href="#">nodes</a>	Created an hour ago	Ready	

Now lets go back to vs code and proceed with Kubernetes deployment and Kubernetes service

1. If you go to Kubernetes deployment in Kubernetes website->you will see the deployment refers to declarative updates for pods and ReplicaSet, so in deployment, we will mention, what image to use, what application name, how many replicas, all those steps, even port number, and Kubernetes will use this deployment to maintain the state.
2. You can do it manually like the one shown in the Kubernetes web page and then run `kubectl apply -f` and the name of this file. but we will do it by integrating python with Kubernetes
3. Similarly, on the left hand side, you can type service, so service is a method for exposing a network application that is running as one or more pods in your cluster
4. So, our application runs on inside the cluster, but for everyone else to have access, it should have a service.
5. So, let's go to vs code=> create a file called "eks.py"
6. To use Kubernetes with python, we need to use something called Kubernetes client(Kubernetes python package), so we will import cluster and config, cluster is used to create deployment and services and config to use the cluster
7. First we need to load the Kubernetes config (the cluster we created), so you have `cd .kube/config` file is there. If you want to do this in terminal you can give the below command  
**`aws eks update-kubeconfig --name cloud-native-cluster`**
8. You can also give the above command if your `.kube/config` file is not updated.
9. Next we need to create an API client.
10. Now we have client, we can start with the deployment
11. Here we will give the name of the app, the number of replicas ,and we are going to use the container image in ECR
12. In terminal, when we need to create a deployment, we give the below command.  
`kubectl apply -f deployment.yaml` and namespace
13. But for python we can give a code to create deployment, so you will day create the deployment in the default name space.
14. If you now enter the below command, it will say no resource found and that the default is empty, but after we run the code, you can find the deployment here  
`kubectl get deployment -n default`
15. Next we need to create service, so here we will give the name of the service and the port number.
16. Once that is run we can run the python file  
**`python3 eks.py`**
17. Now in the terminal, you can run to run in chain watch mode. You can see you flask app created and about to be in read state  
**`kubectl get pods -n default -w.`**
18. You can see the deployment and service being created by  
`kubectl get deployment -n default`  
`kubectl get svc -n default`
19. few additional commands:
20. to check if any issue or errors and details about the pod  
**`kubectl get pods -n default -w`** => to get the name  
**`kubectl describe pods <name> -n default`** =>to get description and status(any error/ issue) of pods  
**`kubectl edit deployment <appname> -n default`** => to open editor in terminal  
**`kubectl port-forward svc/my-flask-service 5000:5000`** => to run the application that is running on port 5000 to run in our localhost at port 5000
21. you will see the application running and that's it,,,we are done!!!

App.py

```
import psutil
from flask import Flask

app = Flask(__name__)
@app.route("/")
def index():
    cpu_percent = psutil.cpu_percent()
    mem_percent = psutil.virtual_memory().percent
    Message = None
    if cpu_percent > 80 or mem_percent > 80:
        Message = " High CPU or Memory Utilization detected.Please scale up"
    return f"CPU Utilisation: {cpu_percent} and Memory Utilization:{mem_percent}"

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

requiremnets.txt:

```
Flask==2.2.3
MarkupSafe==2.1.2
Werkzeug==2.2.3
itsdangerous==2.1.2
psutil==5.8.0
plotly==5.5.0
tenacity==8.0.1
boto3==1.9.148
kubernetes==10.0.1
```

app.py modified:

```
import psutil
from flask import Flask, render_template

app = Flask(__name__)
@app.route("/")
def index():
    cpu_percent = psutil.cpu_percent()
    mem_percent = psutil.virtual_memory().percent
    Message = None
    if cpu_percent > 80 or mem_percent > 80:
        Message = " High CPU or Memory Utilization detected.Please scale up"
    return render_template("index.html", cpu_percent=cpu_percent,
mem_percent=mem_percent, message=Message)

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

index.html

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Indicator Gauge Chart</title>
  <!-- Include Chart.js library from CDN -->
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <div class="container">
    <h1 align="center">System Monitoring</h1>
    {% if message %}
    <div class="alert alert-danger">{{ message }}</div>
    {% endif %}
  </div>

  <!-- Canvas element to render the CPU Usage chart -->
  <canvas id="cpuChart" width="10" height="10"></canvas>

  <!-- Canvas element to render the Memory Usage chart -->
  <canvas id="memChart" width="10" height="10"></canvas>

  <script>
    // Retrieve the CPU and Memory percentage values from the Python code
    var cpuPercent = {{ cpu_percent|default(0) }};
    var memPercent = {{ mem_percent|default(0) }};

    // Get the canvas elements
    var cpuCanvas = document.getElementById('cpuChart');
    var memCanvas = document.getElementById('memChart');

    // Create separate configurations for CPU and Memory charts
    var cpuConfig = {
      type: 'doughnut',
      data: {
        labels: ['CPU Usage'],
        datasets: [{
          data: [cpuPercent, 100 - cpuPercent],
          backgroundColor: ['#36A2EB', '#E7E7E7'],
          hoverBackgroundColor: ['#36A2EB', '#E7E7E7'],
        }]
      },
      options: {
        elements: {
          center: {
            text: cpuPercent + '%',
            color: '#000',
            fontStyle: 'Arial', // Set the font style
            sidePadding: 20 // Adjust the padding
          }
        }
      }
    };
  </script>

```



```

var memConfig = {
  type: 'doughnut',
  data: {
    labels: ['Memory Usage'],
    datasets: [{
      data: [memPercent, 100 - memPercent],
      backgroundColor: ['#FF6384', '#E7E7E7'],
      hoverBackgroundColor: ['#FF6384', '#E7E7E7'],
    }]
  },
  options: {
    elements: {
      center: {
        text: memPercent + '%',
        color: '#000',
        fontStyle: 'Arial', // Set the font style
        sidePadding: 20 // Adjust the padding
      }
    }
  }
}

};

// Create and render the CPU and Memory charts
var cpuChart = new Chart(cpuCanvas.getContext('2d'), cpuConfig);
var memChart = new Chart(memCanvas.getContext('2d'), memConfig);

</script>

</body>
</html>

```

## Dockerfile

```

FROM python:3.9.16-slim-buster

WORKDIR /app

COPY . /app

RUN pip3 install --no-cache-dir -r requirements.txt

ENV NAME World

EXPOSE 5000

CMD ["python", "app.py"]

```

ecr.py

```
import boto3

ecr_client = boto3.client('ecr')

repository_name = "my-cloud-native-repo"
response = ecr_client.create_repository(repositoryName=repository_name)

repository_uri = response['repository']['repositoryUri']
print(repository_uri)
```

eks.py

```
#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-flask-app"),
    spec=client.V1DeploymentSpec(
        replicas=1,
        selector=client.V1LabelSelector(
            match_labels={"app": "my-flask-app"}
        ),
        template=client.V1PodTemplateSpec(
            metadata=client.V1ObjectMeta(
                labels={"app": "my-flask-app"}
            ),
            spec=client.V1PodSpec(
                containers=[
                    client.V1Container(
                        name="my-flask-container",
                        image="279646452254.dkr.ecr.us-east-1.amazonaws.com/my-cloud-
native-repo",
                        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"),
    spec=client.V1ServiceSpec(
        selector={"app": "my-flask-app"},
        ports=[client.V1ServicePort(port=5000)]
    )
)

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