CASE STUDY- INTRODUCTION TO KUBERNETES

You have just joined a startup Ventura Software as a DevOps Lead Engineer. The company relies on a Monolithic Architecture for its product. Recently, the senior management was hired. The new CTO insists on having a Microservice Architecture. The Development Team is working on breaking the Monolith. Meanwhile, you have been asked to host a Test Application on Kubernetes, to understand how it works.

The following things must be implemented:

1. **Deploy an Apache2 deployment of 2 replicas**
2. **Sample code has been checked in at the following Git-Hub repo:**

   https://github.com/hshar/website.git.

   You must containerize this code and push it to Docker Hub. Once done, deploy it on

   Kubernetes with 2 replicas
3. **Deploy Ingress with the following rules:**

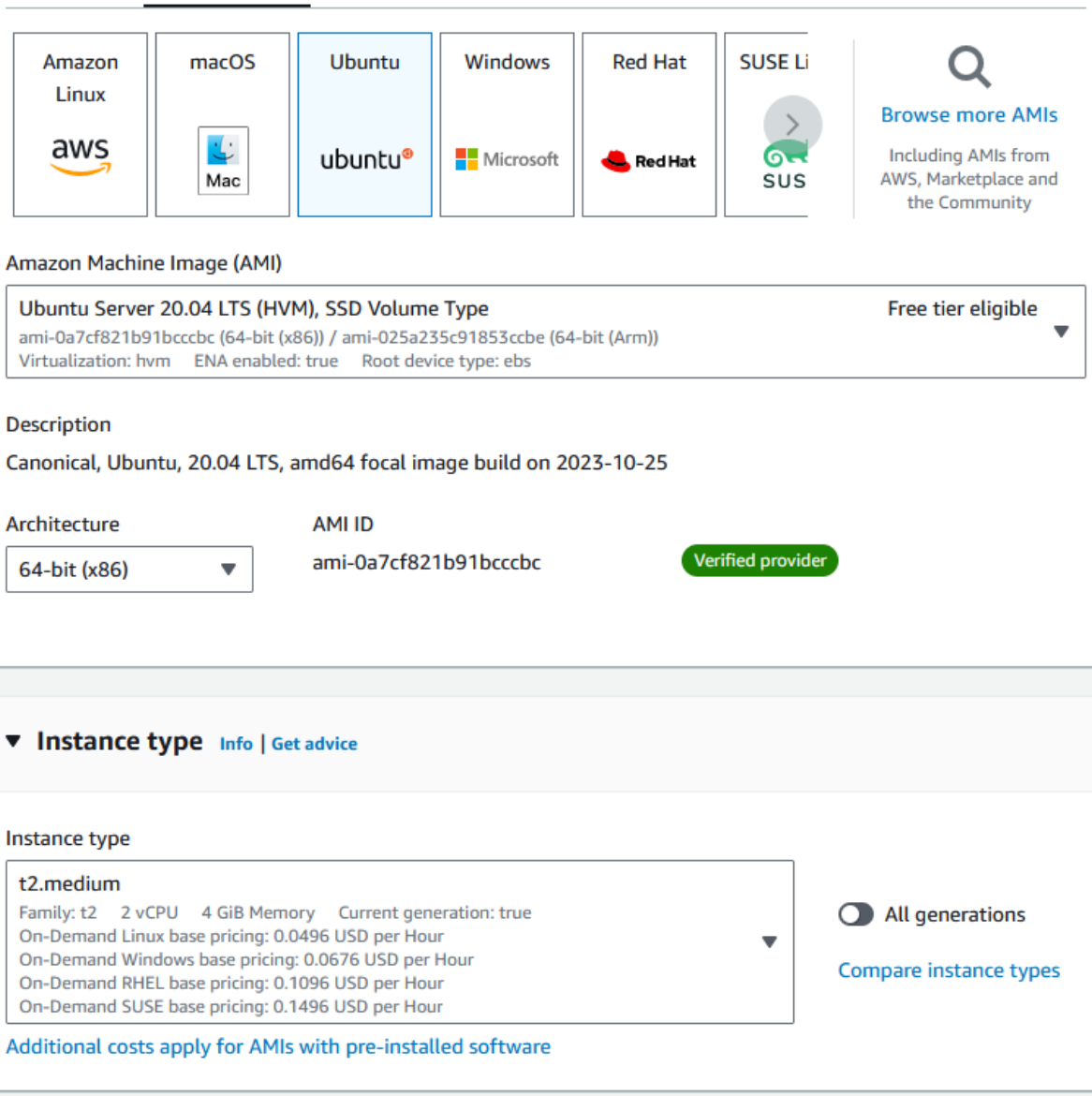
   i) */Apache* should point to the Apache pods

   ii) */custom* should point to the GitHub application

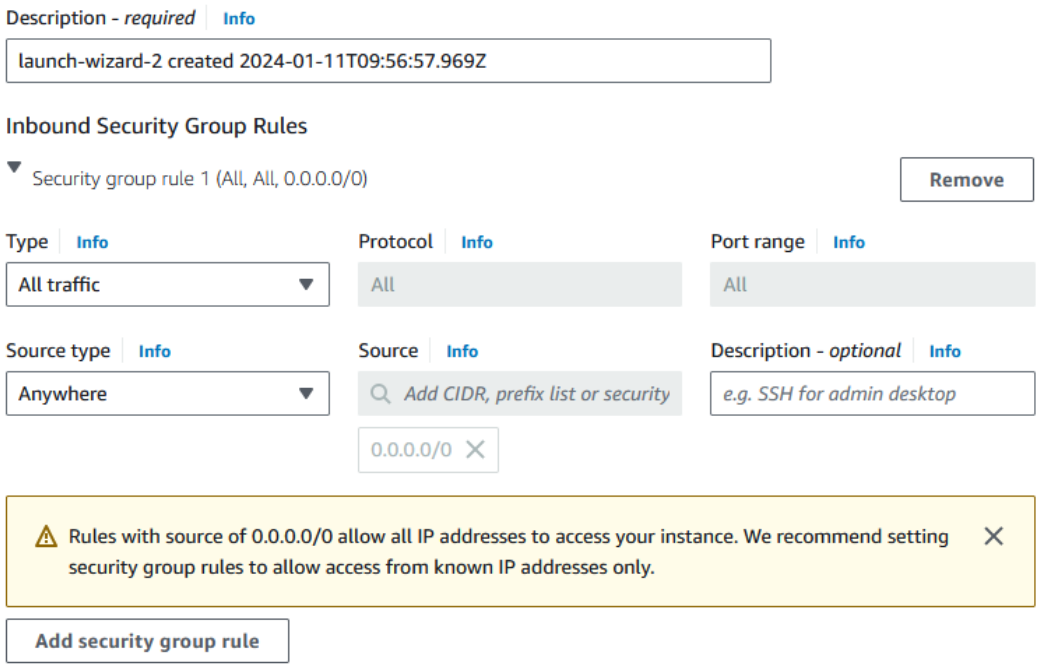
Answer:

As I am using AWS resources to create a Kubernetes environment, I am using AWS EC2 instances Ubuntu 20.04 LTS, launching t2.medium.

Login to your AWS console and navigate to EC2 service → Click on Launch Instance



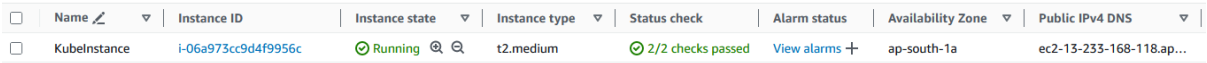
- Select OS: Ubuntu 20.04LTS Version
- Instance Type: t2.medium (Cost will incur)
- Select Keypair (If you do not have a key pair please create one)
- Network Settings Allow all the traffic to avoid issues



Example like above.

Then click on Launch Instance.

Wait for a few mins since it will take 3 to 5 mins to prep the instance.



Once you see the Instance Checks are green then try to access the instance.

Now in the Linux terminal type the below commands to prep the environment.

```
$ sudo apt-get update
$ sudo apt-get install docker.io -y
$ curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube_l
atest_amd64.deb
$ sudo dpkg -i minikube_latest_amd64.deb

$ sudo chmod 777 /var/run/docker.sock

$ sudo snap install kubectl -classic

kubectl 1.28.5 from Canonical✓ installed

$ minikube start
```

We have installed and Kubernetes.

```
ubuntu@ip-172-31-27-96:~$ minikube start
* minikube v1.32.0 on Ubuntu 20.04 (xen/amd64)
* Automatically selected the docker driver. Other choices: ssh, none
* Using Docker driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.28.3 preload ...
    > preloaded-images-k8s-v18-v1...:  403.35 MiB / 403.35 MiB  100.00% 15.66 M
    > gcr.io/k8s-minikube/kicbase...:  453.90 MiB / 453.90 MiB  100.00% 14.89 M
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Verifying Kubernetes components...
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Now developing the deployment config

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: apache-deployment
 labels:
   app: apache
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: ubuntu/apache2
        ports:
        - containerPort: 80
```

Let’s Deployment

```
$ kubectl create -f apache.yaml
deployment.apps/apache-deployment created
```

Check the deployment.

```
$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
apache-deployment   2/2     2            2           4m16s
```

```
$ kubectl expose deployment apache-deployment --type NodePort

$ kubectl get svc
NAME                 TYPE         CLUSTER-IP      EXTERNAL-IP
PORT(S)    AGE
apache-deployment    ClusterIP    10.104.65.184   <none>
80/TCP     3m22s
kubernetes           ClusterIP    10.96.0.1       <none>
443/TCP    26m
```

First Deployment has been completed.

```
$ git clone https://github.com/hshar/website.git
Cloning into 'website'...
remote: Enumerating objects: 8, done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 8
Unpacking objects: 100% (8/8), 82.67 KiB | 7.52 MiB/s, done.

$ cd website
$ sudo nano Dockerfile
FROM ubuntu
RUN apt-get update
RUN apt-get install apache2 -y
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND

$ sudo docker build . -t arkitcoin/k8scasestudy
Successfully built e14c9796d7e5
Successfully tagged arkitcoin/k8scasestudy:latest
```

```
ubuntu@ip-172-31-27-96:~/website$ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED              SIZE
arkitcoin/k8scasestudy      latest    e14c9796d7e5   About a minute ago   233MB
ubuntu                      latest    174c8c134b2a   4 weeks ago          77.9MB
gcr.io/k8s-minikube/kicbase v0.0.42   dbc648475405   2 months ago         1.2GB
ubuntu@ip-172-31-27-96:~/website$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: arkitcoin
Password:
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@ip-172-31-27-96:~/website$ docker push arkitcoin/k8scasestudy
Using default tag: latest
The push refers to repository [docker.io/arkitcoin/k8scasestudy]
0563947fab86: Pushed
b19e0668c55e: Pushed
119db9557aa8: Pushed
a1360aae5271: Mounted from library/ubuntu
latest: digest: sha256:d8cd4d8034a5b6eb300bd68b57c1ad3aadc925507c72557f2d6a62ad20646874 size: 1163
```

Create a custom deployment file

```
ubuntu@ip-172-31-27-96:~/website$ cat customdeploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: custom-deployment
 labels:
   app: custom
spec:
```

```
  replicas: 2
  selector:
    matchLabels:
      app: custom
  template:
    metadata:
      labels:
        app: custom
    spec:
      containers:
      - name: custom
        image: arkitcoin/k8scasestudy
        ports:
        - containerPort: 80
```

```
$ kubectl create -f customdeploy.yaml
deployment.apps/custom-deployment created
```

```
$ kubectl expose deployment custom-deployment --type NodePort
service/custom-deployment exposed
```

If you want to check the website is working using Minikube just forward the port and access the website.

```
$ kubectl port-forward service/custom-deployment --address
0.0.0.0 :80
```

Deploy Ingress

```
$ kubectl get ingress
No resources found in default namespace.

$ minikube addons enable ingress
* ingress is an addon maintained by Kubernetes. For any
concerns contact minikube on GitHub.
You can view the list of minikube maintainers at:
https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-
certgen:v20231011-8b53cabe0
  - Using image registry.k8s.io/ingress-
nginx/controller:v1.9.4
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-
certgen:v20231011-8b53cabe0
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

Lets define ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress
 annotations:
   nginx.ingress.kubernetes.io/rewrite-targeti: /
spec:
 ingressClassName: nginx
 rules:
 - http:
    paths:
    - path: /apache
      pathType: Prefix
      backend:
        service:
          name: apache-deployment
          port:
            number: 80
    - path: /custom
      pathType: Prefix
      backend:
        service:
          name: custom-deployment
          port:
            number: 80
```

```
$ kubectl create -f ingress.yaml

$ kubectl get ingress
NAME      CLASS   HOSTS   ADDRESS         PORTS   AGE
ingress   nginx   *       192.168.49.2    80      2m37s
```

```
$ kubectl get svc -A
NAMESPACE       NAME                                 TYPE
CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
default         apache-deployment
ClusterIP   10.104.65.184   <none>        80/TCP
57m
default         custom-deployment                    NodePort
10.96.184.59    <none>        80:31900/TCP
27m
default         kubernetes
ClusterIP   10.96.0.1       <none>        443/TCP
80m
ingress-nginx   ingress-nginx-controller             NodePort
10.108.87.183   <none>        80:30997/TCP,443:31386/TCP
7m43s
```

```
ingress-nginx   ingress-nginx-controller-admission
ClusterIP   10.99.101.192   <none>        443/TCP
7m43s
kube-system     kube-dns
ClusterIP   10.96.0.10      <none>
53/UDP,53/TCP,9153/TCP       80m
```

```
$ kubectl port-forward service/ingress-nginx-controller -n
ingress-nginx --address 0.0.0.0 :80
Forwarding from 0.0.0.0:32773 -> 80
```

Successfully Access the Website hosted on Kubernetes.