Coursework

Cloud Computing

Lakshmanan Balasubramanian
MSc Advanced Computer Science
220237815

INDEX

Task 1: Deploy a web application component in Docker Environment.

1. Pull and run the docker image "nclcloudcomputing/javabenchmarkapp" from Docker hub in the virtual machine running on your laptop/PC. Perform these tasks using the Command Line Interface (CLI).
   - The docker image has been pulled from the docker hub using the following command.
     - "*docker pull nclcloudcomputing/javabenchmarkapp*"
   - the image is pulled and ready in the docker. The below image (*Fig 1*) represents the docker image that has been pulled.

```
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing$ docker pull ncl
cloudcomputing/javabenchmarkapp
Using default tag: latest
latest: Pulling from nclcloudcomputing/javabenchmarkapp
b56ae66c2937: Pull complete
81cebc5bcaf8: Pull complete
9f7678525069: Pull complete
0710a15130c1: Pull complete
Digest: sha256:2fe4006cfe2b15fd09463201f8cbcd36566a309f036327191b8fa9b51bde9502
Status: Downloaded newer image for nclcloudcomputing/javabenchmarkapp:latest
docker.io/nclcloudcomputing/javabenchmarkapp:latest
```

Figure 1 docker image of nclcloudcomputing/javabenchmarkapp has been pulled

2. Verify whether the web application is working by pointing your web browser to (http://localhost:8080/primecheck). The page should display the time taken to perform a prime number checking calculation.
   - The following command is executed to run the docker image that was pulled earlier.
     - "*docker run -dp 8080:8080 nclcloudcomputing/javabenchmarkapp*"
   - In the browser with the following URL (*http://localhost:8080*), the web page is displayed. It displays the time taken to perform the prime number checking calculation.
   - The below image (*Fig 1.1*) represents the docker image is executed.

```
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing$ docker run -dp 8080:8080 nclcloudcomputing/javabenchmarkapp
f3980922064cc3f1bdf5dd6b06814fd5177eb0da6bb582f0b423f4cfe38f7897
```

Figure 1.1 Docker run nclcloudcomputing/javabenchmarkapp

   - The below image (*Fig 1.2*) represents the web page being displayed in the browser.

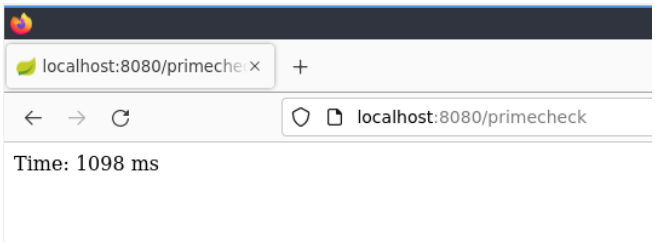


Figure 1.2 Web page displayed in the browser

Task 2: Deploy a complex web application stack in Docker Environment.

1. Pull the following images from the Docker hub.
   - The docker image is not pulled manually or one by one. Instead, I have mentioned the images in the docker-compose file under section (2), which will pull the image from the docker-hub while executing the compose file.
2. Define a Docker compose file which contains all necessary configurations and instructions for deploying and instantiating the above docker images (web server, locust, MongoDB, cAdvisor, Docker swarm Visualizer).
   - The docker-compose file is defined with all the mentioned images.
   - The docker-compose file can be referred below.
   - Docker-compose.yml

```yaml
version: "3"
services:
  benchmarkapp:
    image: nclcloudcomputing/javabenchmarkapp
    deploy:
      resources:
        limits:
          cpus: "0.5"
          memory: 100M
        reservations:
          cpus: "0.25"
          memory: 75M
    ports:
      - "8081:8080"
    environment:
      ENV_KEY: ENV_VALUE
  locust:
    image: locustio/locust
    ports:
      - "8089:8089"
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py -host=http://192.168.56.101:8081
  mongodb:
    image: mongo:4.4.6
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=example
    ports:
      - "27017:27017"
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:latest
    volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
```

```
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
    ports:
      - "8080:8080"
  visualizer:
    image: dockersamples/visualizer:latest
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    ports:
      - "5000:8080"
```

- Locustfile.py

```
from locust import HttpUser, task

class HelloWorldUser(HttpUser):
    @task
    def hello_world(self):
        self.client.get("/primecheck")
```

3. Create a web application topology on a single docker swarm node by using the above docker compose file.
   - To create a web application topology on a single docker swarm, a docker swarm is initiated first.
   - The docker swarm can be initiated using the following command.
     - *"docker swarm init"*
   - Some system might ask to initiate ip address, in that case the following command can be used.
     - *"docker swarm init –advertise-addr [IP ADDRESS]"*
   - The below image (*Fig 2*) shows that the docker swarm node is initiated.

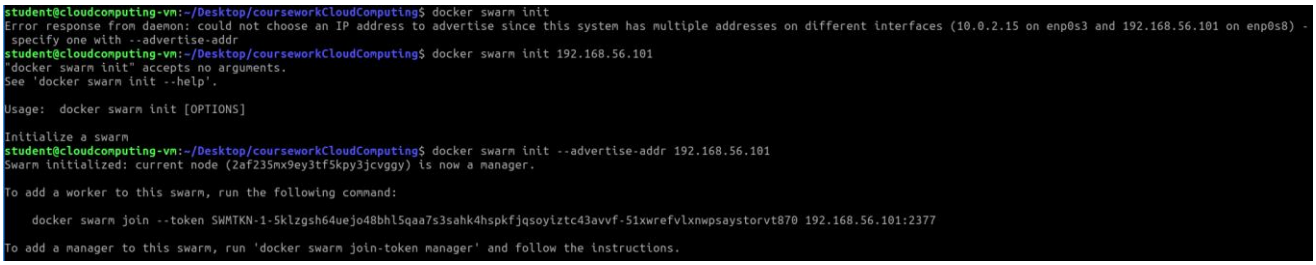


Figure 2 docker swarm initialization

   - Then, the images are stacked and deployed into the swarm node by the following command.
     - *"docker stack deploy -c docker-compose.yml [service_name]"*
   - The below image (*Fig 2.1*) shows the images been deployed.

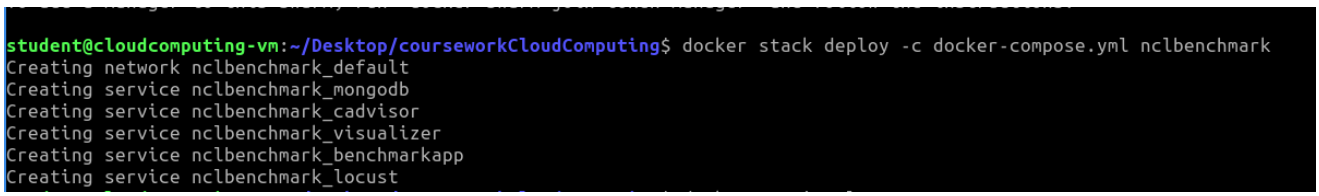


Figure 2.1 images been deployed

- The status of the image can be checked using the following command.
  - "*docker service ls*"
- The below image (*Fig 2.2*) shows the status of each image.

```
student@ML-RefVm-691229:~/Desktop/courseworkCloudComputing$ docker service ls
ID             NAME                      MODE         REPLICAS   IMAGE                                          PORTS
h7ncofkz4mgx   nclbenchmark_benchmarkapp  replicated   1/1        nclcloudcomputing/javabenchmarkapp:latest      *:8081->8080/tcp
zllvou4ezcin   nclbenchmark_cadvisor      replicated   1/1        gcr.io/cadvisor/cadvisor:v0.45.0               *:8080->8080/tcp
ptjgnzkfqo1i   nclbenchmark_locust        replicated   1/1        locustio/locust:latest                         *:8089->8089/tcp
vqmmfblerimg   nclbenchmark_mongodb       replicated   1/1        mongo:4.4.6                                    *:27017->27017/tcp
vpjuwg01ecee   nclbenchmark_visualizer    replicated   1/1        dockersamples/visualizer:latest                *:5000->8080/tcp
student@ML-RefVm-691229:~/Desktop/courseworkCloudComputing$
```

Figure 2.2 status of the images

- Here, the "Replicas" represents the status. If the replicas are "1/1" then the image is working which can be seen using the port mentioned. If the replicas are "0/1" then the image is starting or stopped.
- The following images represents the working of all the images.

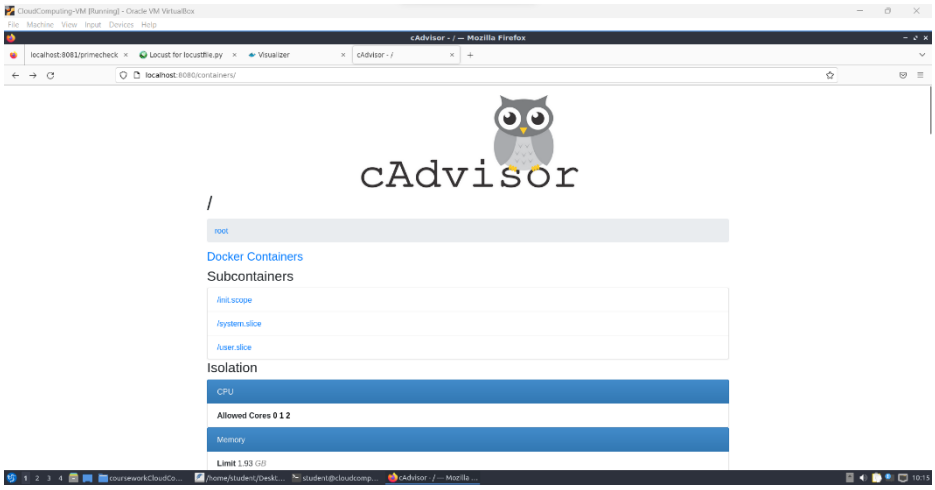


Figure 2.3 cadvisor (192.168.56.101:8080/containers)

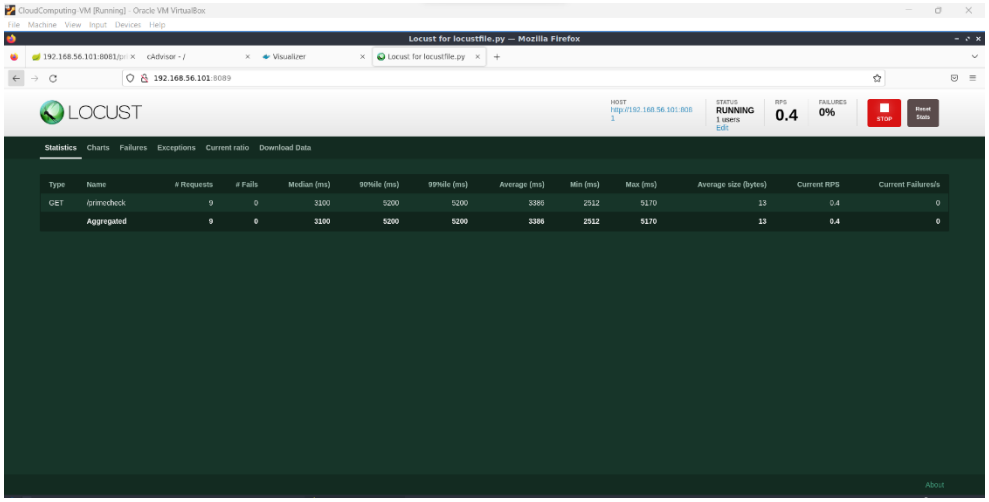


Figure 2.4 locust (192.168.56.101:8089)

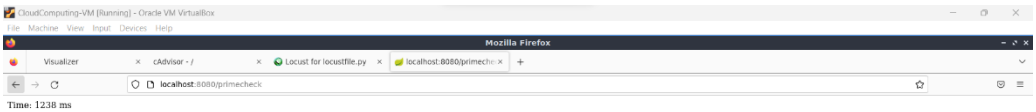


Figure 2.5 java app (192.168.56.101:8080/primecheck)

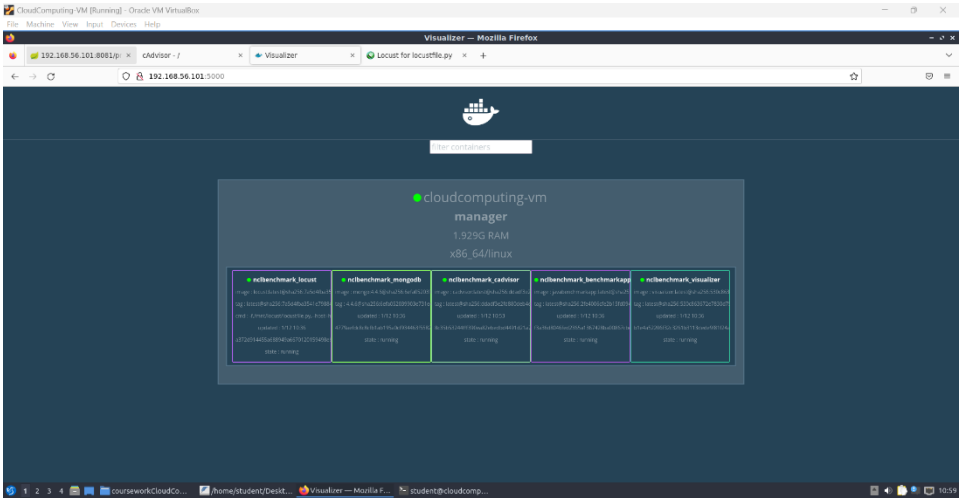


Figure 2.6 visualizer (192.168.56.101:5000)

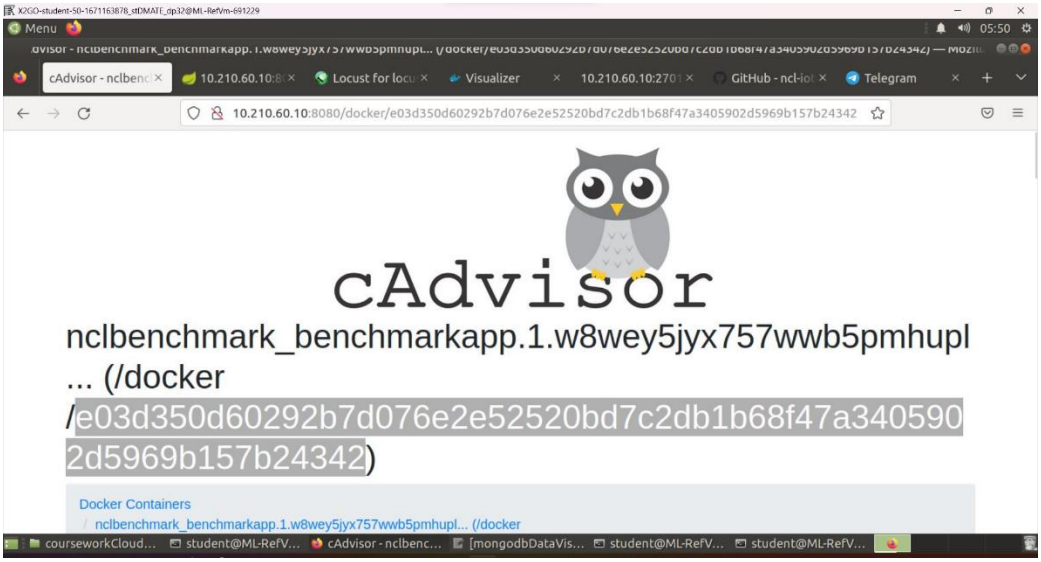
Task 3: Deploy a complex web application stack in Docker Environment.

1. Build a Docker image of a Java program. Download the source code from the GitHub. The application implements how to extract the monitoring information from the Docker cAdvisor container and how to insert this monitoring information into a MongoDB docker container.
   - The code is pulled from the GitHub using the following command.
     - *"git clone <url>"*
   - The below image (*Fig 3*) shows the code is pulled.

```
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing/cadvisor-scrapper$ git clone https://github.com/ncl-iot-team/cadvisor-scraper.git
Cloning into 'cadvisor-scraper'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 38 (delta 4), reused 38 (delta 4), pack-reused 0
Receiving objects: 100% (38/38), 20.66 KiB | 1.72 MiB/s, done.
Resolving deltas: 100% (4/4), done.
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing/cadvisor-scrapper$ ls
cadvisor-scraper
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing/cadvisor-scrapper$ cd cadvisor-scraper
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing/cadvisor-scrapper/cadvisor-scraper$ ls
'cAdvisor scrape.iml'   pom.xml   README.md   response.json   src
student@cloudcomputing-vm:~/Desktop/courseworkCloudComputing/cadvisor-scrapper/cadvisor-scraper$
```

Figure 3 Code pulled from GitHub

2. Edit the provided Java code using the IntelliJ editor. The instructor in the Java code that need editing include container id, mongoDB addresss and the endpoint address.
   - The container id can be found in the cAdvisor. The container id of the JavaBenchmark app container is noted and edited in the Main.java file. This container id is unique and changes every time when the container is initiated. Thus, it has to be modified every time.
   - The below image (*Fig 3.1*) shows the container id of benchmarkapp from cAdvisor.



   - The mongoDB address and the endpoint address is the host ip address in which swarm gets initiated. That can be obtained by typing the following command.
     - *"docker node ls"* to get the node id.
     - *"docker node inspect <node id>"*

- The below image (*Fig 3.2*) shows the ip address that the node is established.

Figure 3.2 Ip address of swarm is obtained

- Now that the ip address is obtained, the mongoDb address and the endpoint address can be modified as the following.
  - "*mongodb://root:example@192.168.56.101:27017*"
    - Here the username is "*root*" and password is "*example*" and the port is "*27017*"
  - "*http://192.168.56.101:8080/api/v1.3/container*"
- The below image (*Fig 3.3*) shows the modified MongoDb address and endpoint address.

```
public class Main {

    //Container ID to monitor - You have to find the container ID via the web frontend of cAdvisor or via Docker command line!
    //Change the ID HERE!
    private static final String containerId = "a7c2d692d90cf5bd9979362def6abd7bb957315ee9d2e15a9cfa2e4f5af2853d";

    //MongoDB address
    //Change the host IP HERE!
    private static final String mongodbAddress = "mongodb://root:example@192.168.56.101:27017";

    //cAdvisor API endpoint
    //Change the host IP HERE!
    private static final String endpoint = "http://192.168.56.101:8080/api/v1.3/docker/";
```

Figure 3.3 MongoDb address and endpoint address

3. After the code modification, pack the program as a standalone docker image and push it to the docker hub. Name the image as cadvisor-scrapper.
   - After the modification is done, the project is built as a docker image with the help of Dockerfile.
   - Dockerfile

```
FROM openjdk:8-jdk-alpine
RUN addgroup -S spring && adduser -S spring -G spring
USER spring:spring
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} cadvisor-scrape-1.0-jar-with-dependencies.jar
ENTRYPOINT ["java", "-jar", "/cadvisor-scrape-1.0-jar-with-
dependencies.jar"]
```

- The following image shows (*Fig 3.4*) the cadvisor-scraper has been built as a Docker image.

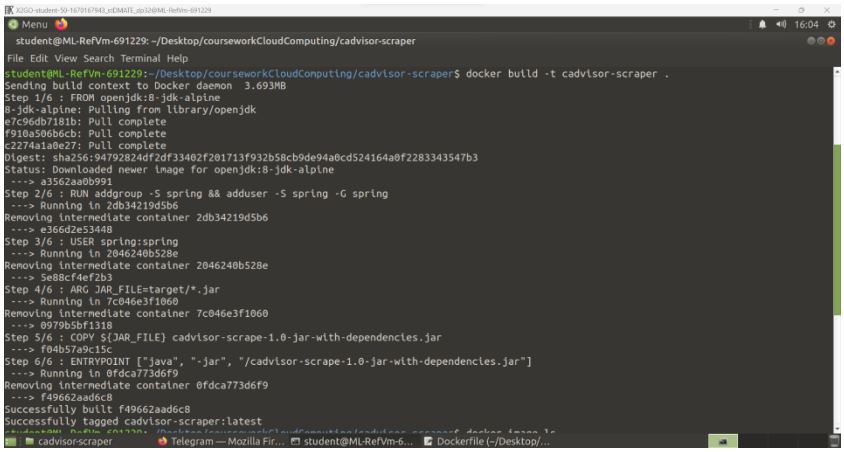


Figure 3.4 cadvisor-scraper as a docker image

- Then, the image is executed to check if it is working. The below image (Fig 3.5) shows that the image is executed.

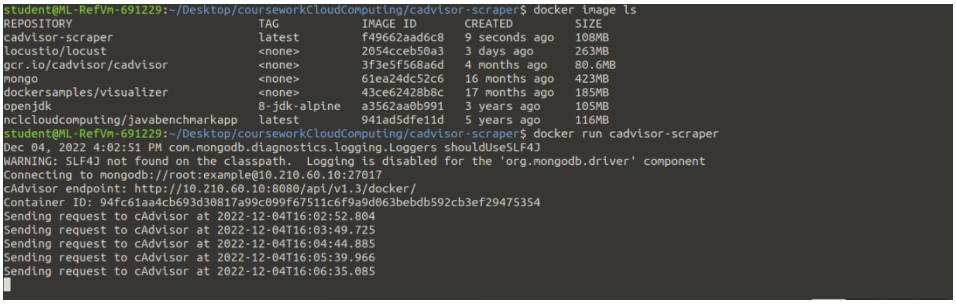


Figure 3.5 cadvisor-scraper image is running

- Now, the image is pushed to the docker hub. Before pushing the image, we are authenticating into the docker hub account by providing the following command.
    - "*docker login*"
- After providing the username and password, the authentication is successful.
- Then, duplicating the "cadvisor-scraper" image with the name as "username/cadvisor-scraper".
- Now, there are two images of cadvisor-scraper with different image name.
- Finally, the duplicated image is pushed to the docker hub using the following command.
    - "docker push <username>/cadvisor-scraper"
- The below image (Fig 3.6) shows the cadvisor-scraper image is pushed to the docker hub.

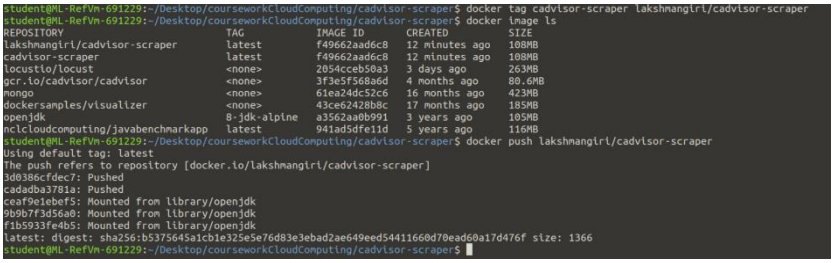


Figure 3.6 cadvisor-scraper image is pushed

Task 4: Fully deploy and run the complex web application stack and undertake performance benchmarking activities.

1. Pull the "cadvisor-scraper" image created in task 3, back to the virtual machine on your laptop/pc.
   - The image can be pulled from the docker hub using the following command.
     - "*docker pull <username>/cadvisor-scraper*"
   - The below image (Fig 4) shows that the image is being pulled.

```
student@ML-RefVm-691229:~/Desktop/courseworkCloudComputing/cadvisor-scraper$ docker pull lakshmangiri/cadvisor-scraper
Using default tag: latest
latest: Pulling from lakshmangiri/cadvisor-scraper
e7c96db7181b: Already exists
f910a506b6cb: Already exists
c2274a1a0e27: Already exists
501c6635ea52: Already exists
e8f84d03039b: Already exists
Digest: sha256:b5375645a1cb1e325e5e76d83e3ebad2ae649eed54411660d70ead60a17d476f
Status: Downloaded newer image for lakshmangiri/cadvisor-scraper:latest
docker.io/lakshmangiri/cadvisor-scraper:latest
student@ML-RefVm-691229:~/Desktop/courseworkCloudComputing/cadvisor-scraper$
```

Figure 4 cadvisor-scraper pulled from docker hub

2. Make necessary changes to cadvisor-scraper so that it monitors the swarm.
   - The containers and ports are already updated and pulled the image to the docker hub.
3. Launch cAdvisor-scraper as a container separately outside of the swarm.
   - The image "*cadvisor-scraper*" is executed using the following command.
     - "*docker run cadvisor-scraper*"
   - The following image (*Fig 4.1*) shows the cadvisor-scraper is running.

```
student@ML-RefVm-691229:~/Desktop/courseworkCloudComputing/cadvisor-scraper$ docker run lakshmangiri/cadvisor-scraper
Dec 04, 2022 4:47:40 PM com.mongodb.diagnostics.logging.Loggers shouldUseSLF4J
WARNING: SLF4J not found on the classpath.  Logging is disabled for the 'org.mongodb.driver' component
Connecting to mongodb://root:example@10.210.60.10:27017
cAdvisor endpoint: http://10.210.60.10:8080/api/v1.3/docker/
Container ID: 94fc61aa4cb693d30817a99c099f67511c6f9a9d063bebdb592cb3ef29475354
Sending request to cAdvisor at 2022-12-04T16:47:42.183
```

Figure 4.1 cadvisor-scraper is running

4. Send dynamic workload (eg., http requests) to the web application image instance by using the load generator instance.
   - The dynamic workload is sent using the locust.
   - The following image (*Fig 4.2*) shows the locust is generating the loads.

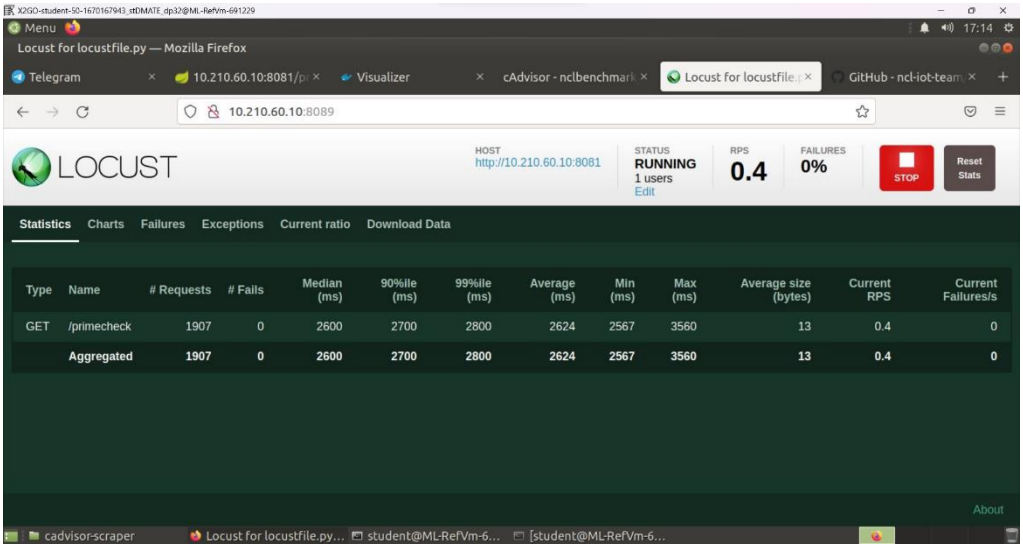


Figure 4.2 Load generator instance

5. Verify that cAdvisor-scraper instance is logging the run-time performance statistics to the MongoDB instance.
   - The mongoDb is executed and its data are inspected.
   - The following commands are used to inspect the MongoDB.
     - "*docker container ls*" => to get the mongoDB container id.
     - "*docker exec -it <mongodb_containerID> mongo 'mongdb://root:example@10.210.60.10:27017/'*" => is used to inspect the mongodb shell.
     - "*show dbs*" => to inspect the available database.
     - "*use <dbname>*" => to inspect into the mentioned database.
     - "*show collections*" => to inspect the collections in the database.
     - "*db["<collectionName>"].find()*" => to inspect the data in the collection.
   - The following image (*Fig 4.3*) shows that the memory data are inspected in the MongoDB shell.

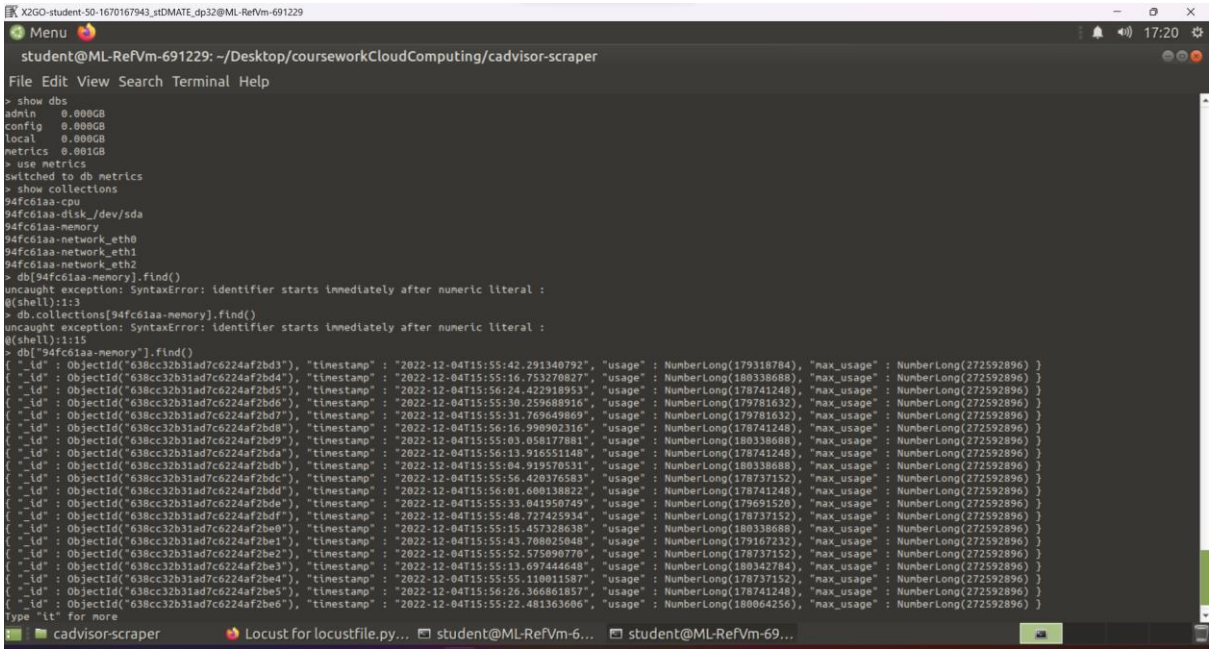


Figure 4.3 Memory data inspected in the Mongoshell

Task 5: Deploy Kubernetes using Terraform and deploy a microservice.

1. Enable the Azure subscription.
   - The Azure subscription is activated by signing into the Azure cloud.
   - The below image (*Fig 5*) shows the Azure cloud.

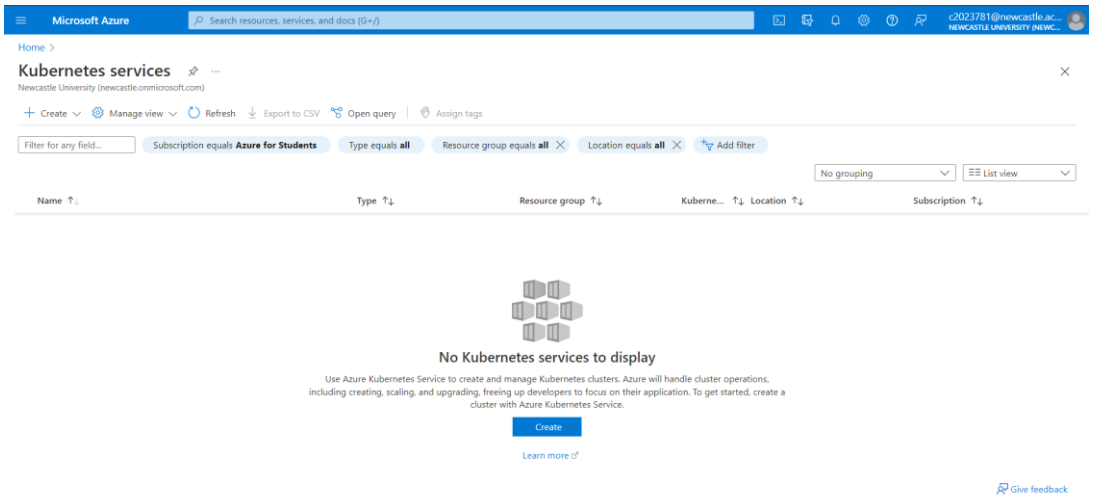


Figure 5 Azure cloud

   - The azure subscription is enabled using the Azure CLI. The following command is used to enable the subscription.
     - *"az account set --subscription 'Azure for Students'"*
   - This will make the subscription enabled and assigned as default.
   - It can be verified using the following command.
     - *"az account show"*
   - The below image (*Fig 5.1*) shows that the subscription is enabled.

```
C:\Users\Lenovo>az account show
{
  "environmentName": "AzureCloud",
  "homeTenantId": "9c5012c9-b616-44c2-a917-66814fbe3e87",
  "id": "4f1dced2-6be4-49d5-aaef-2258e06c3849",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Azure for Students",
  "state": "Enabled",
  "tenantId": "9c5012c9-b616-44c2-a917-66814fbe3e87",
  "user": {
    "name": "c2023781@newcastle.ac.uk",
    "type": "user"
  }
}

C:\Users\Lenovo>
```

Figure 5.1 Azure subscription enabled

2. Create an Azure Kubernetes cluster named "csc8110" using HashiCorp language (HCL)
   - Azure Kubernetes cluster are created using terraform. The terraform is built using the HashiCorp language. This terraform is simply an infrastructure as Code used to create infrastructures like Kubernetes.

- The following code creates Kubernetes cluster.
- Providers.tf

```
# providers.tf
# This initiates Azure platform and its version

provider "azurerm" {
  features {}
}

terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "2.39.0"
    }
  }
}
```

- Main.tf

```
# This main.tf creates Azure resource group and Kubernetes cluster using the
variables from terraform.tfvars
resource "azurerm_resource_group" "aks-rg" {
  name     = var.resource_group_name
  location = var.location
}

resource "azurerm_role_assignment" "role_acrpull" {
  scope                            = azurerm_container_registry.acr.id
  role_definition_name             = "AcrPull"
  principal_id                     =
azurerm_kubernetes_cluster.aks.kubelet_identity.0.object_id
  skip_service_principal_aad_check = true
}

resource "azurerm_container_registry" "acr" {
  name                = var.acr_name
  resource_group_name = azurerm_resource_group.aks-rg.name
  location            = var.location
  sku                 = "Standard"
  admin_enabled       = false
}

resource "azurerm_kubernetes_cluster" "aks" {
  name                = var.cluster_name
  kubernetes_version  = var.kubernetes_version
  location            = var.location
  resource_group_name = azurerm_resource_group.aks-rg.name
  dns_prefix          = var.cluster_name
```

```
  default_node_pool {
    name                = "system"
    node_count          = var.system_node_count
    vm_size             = "Standard_D2_v2"
  }

  identity {
    type = "SystemAssigned"
  }

  network_profile {
    load_balancer_sku = "Standard"
    network_plugin    = "kubenet"
  }
}
```

- Variable.tf

```
# This variable.tf is used to define the variable for creating the cluster

variable "resource_group_name" {
  type        = string
  description = "RG name in Azure"
}
variable "location" {
  type        = string
  description = "Resources location in Azure"
}
variable "cluster_name" {
  type        = string
  description = "AKS name in Azure"
}
variable "kubernetes_version" {
  type        = string
  description = "Kubernetes version"
}
variable "system_node_count" {
  type        = number
  description = "Number of AKS worker nodes"
}
variable "acr_name" {
  type        = string
  description = "ACR name"
}
```

- Terraform.tfvars

```
# This terraform.tfvars is used to declare the values to the variable
initiated.
# main.tf uses these values to create resource group and kubernetes cluster
```

```
resource_group_name = "csc8110_group"
location            = "EastUS"
cluster_name        = "csc8110Cluster"
kubernetes_version  = "1.25.2"
system_node_count   = 1
acr_name            = "cscCourse15"
```

- Output.tf

```
# This output.tf file is used to print the details of the resource group and
cluster details after creation

output "aks_id" {
  value = azurerm_kubernetes_cluster.aks.id
}

output "aks_fqdn" {
  value = azurerm_kubernetes_cluster.aks.fqdn
}

output "aks_node_rg" {
  value = azurerm_kubernetes_cluster.aks.node_resource_group
}

output "acr_id" {
  value = azurerm_container_registry.acr.id
}

output "acr_login_server" {
  value = azurerm_container_registry.acr.login_server
}

resource "local_file" "kubeconfig" {
  depends_on   = [azurerm_kubernetes_cluster.aks]
  filename     = "kubeconfig"
  content      = azurerm_kubernetes_cluster.aks.kube_config_raw
}
```

- The cluster is then created using the terraform commands.
- The following command will initiate a terraform instance.
    - *"terraform init"*
- The following image (*Fig 5.2*) shows that terraform is initiated.

```
PS D:\Newcastle University\Advanced CS\CSC8110 - Cloud Computing\CourseworkCloudComputing\LakshmananB\T5\kubernetes-cluster> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/local from the dependency lock file
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Using previously-installed hashicorp/local v2.2.3
- Using previously-installed hashicorp/azurerm v2.39.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Figure 5.2 Terraform initiated

- Then the terraform creates an execution plan to create the cluster.
- The following command is used to create a terraform plan.
    - "*terraform plan*"
- The following image (*Fig 5.3*) shows that terraform execution plan is created.

```
PS D:\Newcastle University\Advanced CS\CSC8110 - Cloud Computing\CourseworkCloudComputing\LakshmananB\T5\kubernetes-cluster> terraform plan
azurerm_resource_group.aks-rg: Refreshing state... [id=/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourceGroups/csc8110_group]
azurerm_container_registry.acr: Refreshing state... [id=/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourceGroups/csc8110_group/providers/Microsoft.Con
tainerRegistry/registries/cscCourse15]
azurerm_kubernetes_cluster.aks: Refreshing state... [id=/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourcegroups/csc8110_group/providers/Microsoft.Con
tainerService/managedClusters/csc8110Cluster]
azurerm_role_assignment.role_acrpull: Refreshing state... [id=/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourceGroups/csc8110_group/providers/Microso
ft.ContainerRegistry/registries/cscCourse15/providers/Microsoft.Authorization/roleAssignments/ce72a525-9a10-952d-265c-c1dc189ad6ae]
local_file.kubeconfig: Refreshing state... [id=2f94eb9de9b3bb7d5f492349b144597bd6eacb90]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply" which may have affected this plan:

  # azurerm_container_registry.acr has been deleted
  - resource "azurerm_container_registry" "acr" {
      - id                  = "/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourceGroups/csc8110_group/providers/Microsoft.ContainerRegistry/registries
/cscCourse15" -> null
      - login_server        = "csccourse15.azurecr.io" -> null
        name                = "cscCourse15"
        # (7 unchanged attributes hidden)
    }

  # azurerm_kubernetes_cluster.aks has been deleted
  - resource "azurerm_kubernetes_cluster" "aks" {
      - fqdn                      = "csc8110cluster-3abc1178.hcp.eastus.azmk8s.io" -> null
      - id                        = "/subscriptions/4f1dced2-6be4-49d5-aaef-2258e06c3849/resourcegroups/csc8110_group/providers/Microsoft.ContainerService/mana
gedClusters/csc8110Cluster" -> null
      - kube_config_raw           = (sensitive value)
      ~ kubelet_identity          = [
          ~ {
              ~ object_id               = "d991b5ab-884a-44d6-af3b-bde640d38146" -> null
                # (2 unchanged elements hidden)
            },
        ]
        name                      = "csc8110Cluster"
      - node_resource_group       = "MC_csc8110_group_csc8110Cluster_eastus" -> null
        # (10 unchanged attributes hidden)

        # (5 unchanged blocks hidden)
```

Figure 7 terraform execution plan

- Then, the terraform executes the action of plan that was created.
- The following command is used to apply the action of plan.
    - "terraform apply"
- The following image (Fig 5.4) shows the action of plan being applied.

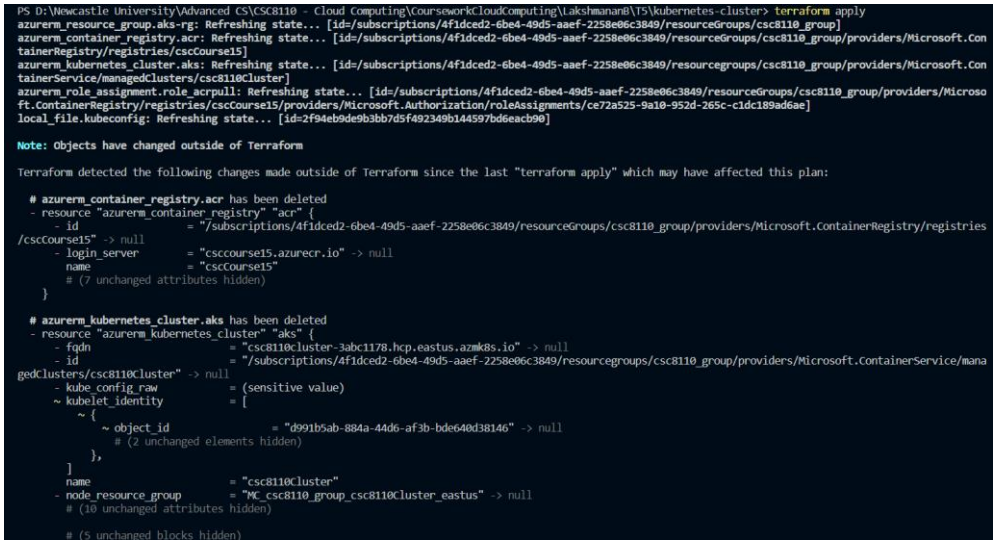


Figure 5.4 terraform plan of action being applied

- Finally, the Kubernetes cluster creation is verified using the following command.
  - "az aks show"
- This lists the available cluster.
- The following image (*Fig 5.5*) shows the current active cluster.

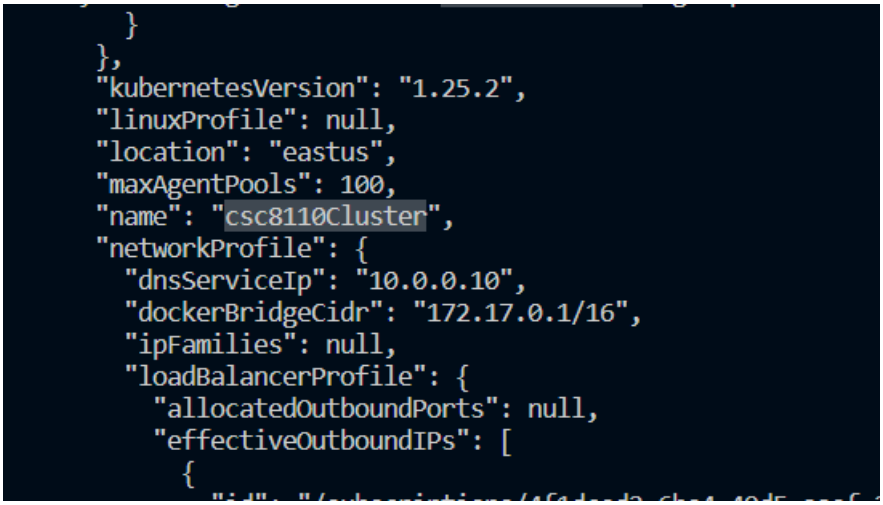


Figure 5.5 active cluster is shown

3. Deploy any sample microservice application once the cluster is ready.
   - With reference to the link [https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-cli], a microservice application is deployed on the cluster.
   - An YAML file is created to deploy the application in the Kubernetes.
   - The YAML file has deployments and service that is used to deploy the application in the Kubernetes cluster.
   - Deployment.yml file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
```

```yaml
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
```

```yaml
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-front
        image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

- Then the application is deployed using the Kubernetes CLI.
- The following command is used to get the active Kubernetes cluster node.
    - *"kubectl get nodes"*
- This will show the current active node. The following image (*Fig 5.6*) shows the active node.

```
PS D:\Newcastle University\Advanced CS\CSC8110 - Cloud Computing\CourseworkCloudComputing\LakshmananB\T5\benchmark-app> kubectl get nodes
NAME                                STATUS   ROLES   AGE    VERSION
aks-system-42203183-vmss000000   Ready    agent   131m   v1.25.2
```

Figure 5.6 active node in the kubernetes cluster

- The yaml code is deployed using the following command.
    - *"kubectl apply -f deployment.yml"*
- This will deploy the yaml file to the Kubernetes cluster.
- The deployment and service are created.
- The deployed service can be seen using the following command.

- *"kubectl get pods"*

- The following image (*Fig 5.7*) shows the pods.

Figure 5.7 shows the pods

- The ip address of the deployed application can be seen using the following command.
  - *"kubectl get services"*
- The following image (*Fig 5.8*) shows the ip address.

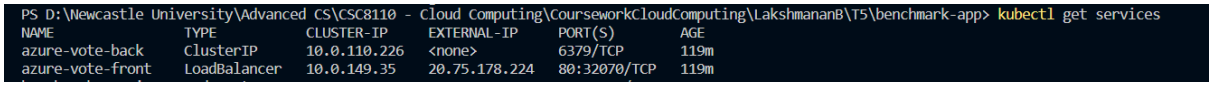


Figure 5.8 Ip address

- The application can be inspected by using the external-IP address.
- The following image (*Fig 5.9*) shows the application.

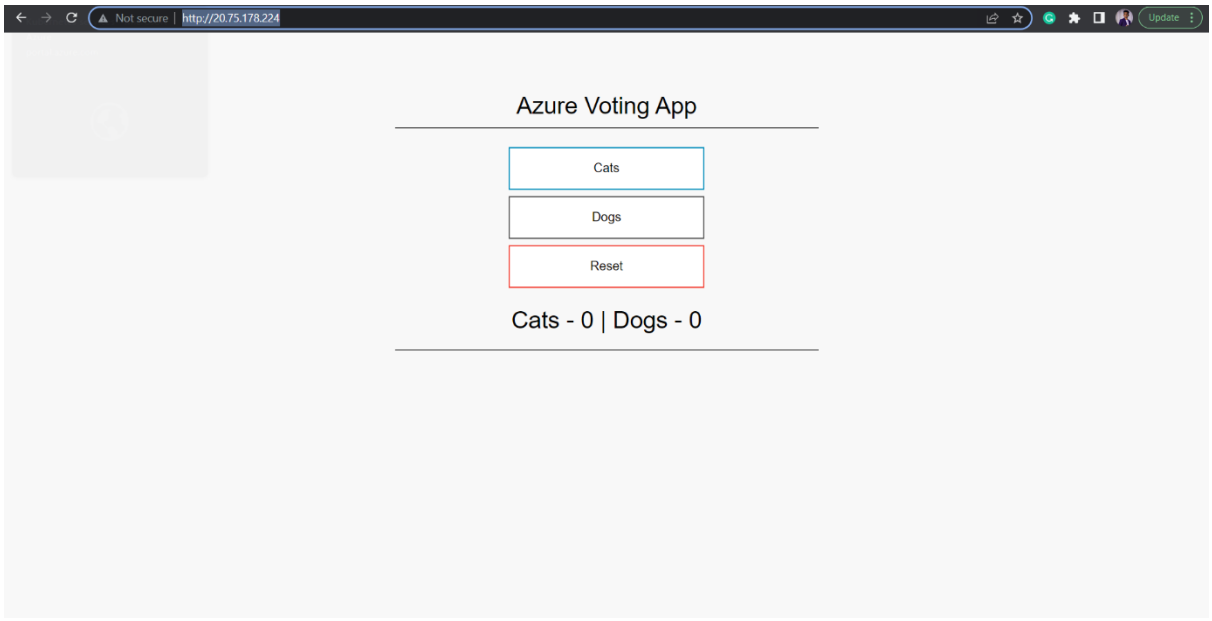
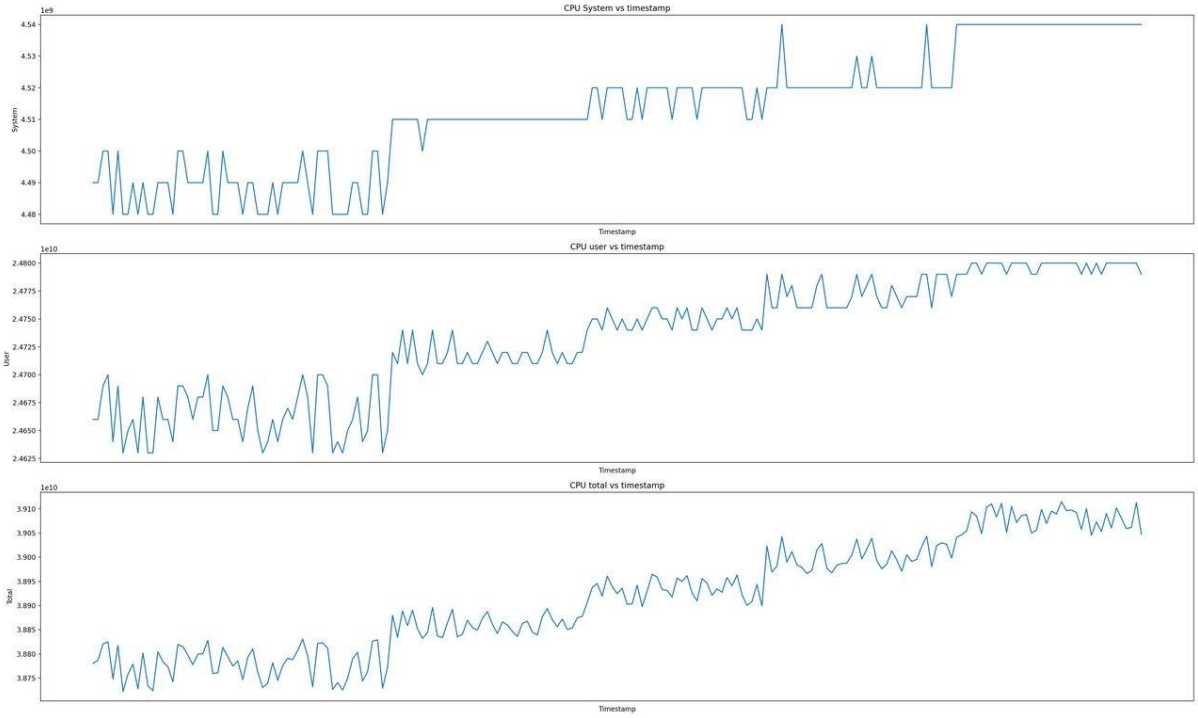


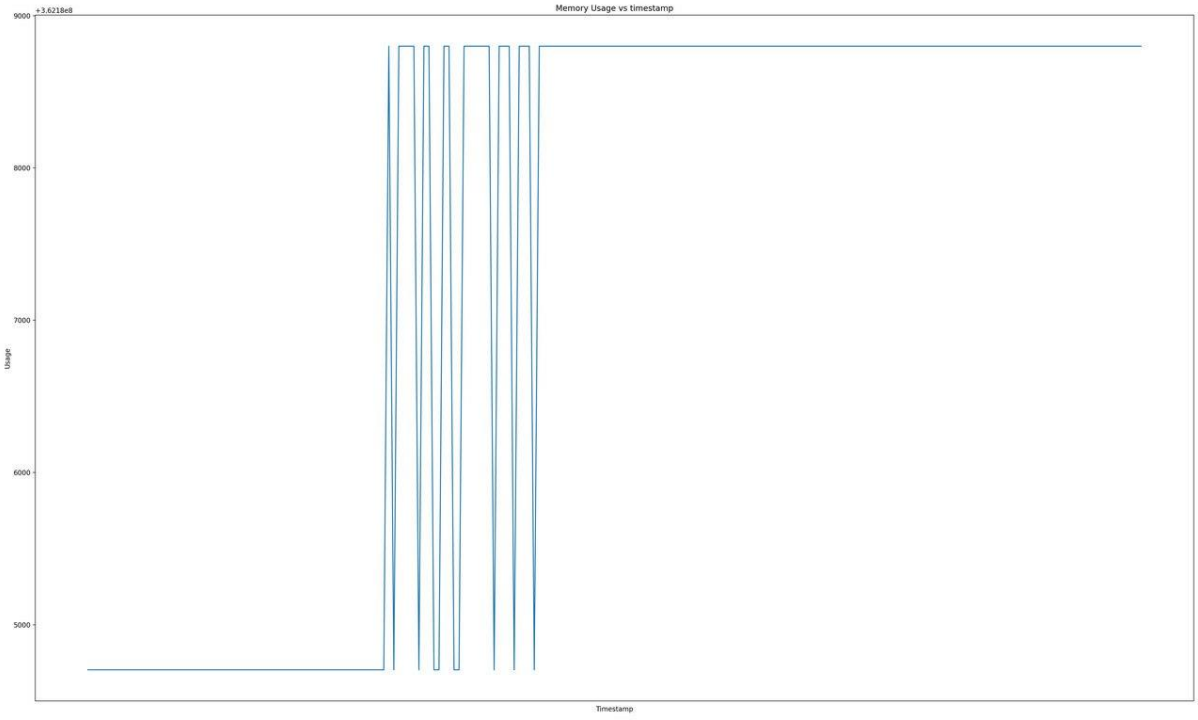
Figure 5.9 Deployed application

Task 6: Report

1. Plotting of Benchmarking report:
    - With inspecting the data, we could see that CPU and Memory from the collections. Taking those data and interpreting it with Matplotlib, the data was able to be plotted in the line chart.
    - The below image shows that CPU data are plotted.



- From the image, it is seen that, the CPU consumption increases when the time increases. Since, every time the loads are generated and thus the CPU consumption also increases.
- The below image visualizes the memory data plotted in the Line chart.



Demonstration video:

1) The video is uploaded to YouTube as an unlisted one. The link can be found here.

References:

1) https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-cli
2) https://www.youtube.com/watch?v=yYIm_swpCoY
3) https://kubernetes.io/docs/tutorials/stateless-application/expose-external-ip-address/