Performance Optimization of Serverless Edge Computing for Machine Learning Workloads in Distributed Edge Environments

MSc Research Project
Cloud Computing

Akash Anil Sane
Student ID: 21220956

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akash Anil Sane
Student ID:	21220956
Programme:	Cloud Computing
Year:	2023/2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	31/01/2024
Project Title:	Performance Optimization of Serverless Edge Computing for Machine Learning Workloads in Distributed Edge Environments
Word Count:	1522
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Akash Anil Sane
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Akash Anil Sane
21220956

1 Setting Up a Serverless ML Infrastructure for Edge Computing

This manual provides a comprehensive explanation of how to set up a serverless computing infrastructure specifically designed for machine learning tasks at the edge layers. Begin by installing Python 3.11.5., a crucial component for scripting serverless functions and ML models. Subsequently, configure Visual Studio Code with Python and Docker extensions to facilitate development and install all the required requirement from requirements.txt. Docker for Desktop is essential for containerising serverless functions within containers. In addition, you will require Kubernetes and KIND for the purpose of managing clusters. Familiarise yourself with Knative, an open-source platform for deploying and scaling serverless applications. Install and set up TensorFlow and Keras through PIP command within your project to construct convolutional neural network models. Make sure that MySQL is set up for database administration and that Google Cloud Buckets are available for model storage. To do load testing using the K6 tool. Ensure that all installs are verified by conducting proper version checks and setups as you progress.

2 Docker for Desktop and Kubernetese Cluster setup with KIND

[Instructions for setting up Docker for Desktop and Kubernetes Cluster with KIND will follow here.]

1. Install Docker for Desktop and update WSL2: To install Docker for Desktop, start by downloading the software with the latest version from the official website[1]. To set up the required environment it is required to update the Windows Subsystem for Linux available in the Windows 11 operating system for the docker application to run smoothly and enable creation of cluster. To prepare your system for Docker for Desktop with Kubernetes, begin with updating and verifying Windows Subsystem for Linux 2 (WSL2) as seen in figure1. For updating WSL 2 installation Open PowerShell or Command Prompt and run:

```
>wsl --update
```

[1] `https://docs.docker.com/desktop/install/windows-install/`

```
C:\Users\akash>wsl --update
Checking for updates.
Updating Windows Subsystem for Linux to version: 2.0.14.
The requested operation requires elevation.
Checking for updates.
Updating Windows Subsystem for Linux to version: 2.0.14.
```

Figure 1: Updating WSL2

Once the WSL2 system is updated restart the Machine for the configurations to apply. When the system is again up and running verify the installation with below command.

```
>wsl --list --verbose
```

```
C:\Users\akash>wsl --list --verbose
  NAME                   STATE           VERSION
* docker-desktop-data    Stopped         2
```

Figure 2: Update verification

When the WSL2 is set up then access the Docker for desktop and enable the Kubernetes option from settings as shown in figure3 so its available for cluster creation and configuration which will now allow to use the kubectl command.

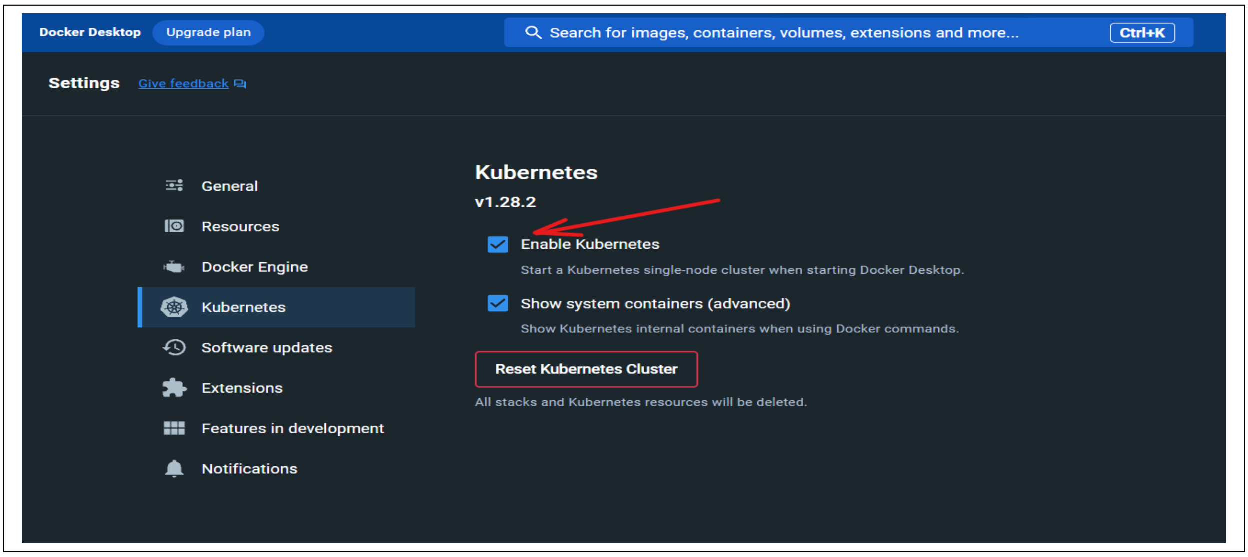


Figure 3: Enable Kubernetese on Docker For Desktop

2. Install Kubernetes with KIND: KIND, acronym for Kubernetes IN Docker, is a utility designed to execute Kubernetes clusters within Docker containers. Follow the

instructions provided on the KIND[2] and follow chocolatey[3] for windows installation to install it on the system.

3. Create a Multi Node Kubernetes Cluster with KIND: Use the KIND tool to establish a Kubernetes cluster consisting of 4 nodes in which one node will act as a master node and other three nodes will be available as worker nodes and these worker nodes will act as edge clients. For this cluster configuration setup use the my-cluster.yaml available in the GitHub project and execute it by following below command.

```
>kind create cluster --name <yourclustername> --config=my-cluster.yaml
```

When the command is successfully executed verify the cluster installation and node setup with below commands.

```
>kind get clusters
```

```
PS C:\Users\akash> kind get clusters
multi-node-cluster
PS C:\Users\akash>
```

Figure 4: Cluster Creation Verification

```
>kubectl get nodes
```

```
C:\Users\akash>kubectl get nodes
NAME                               STATUS   ROLES           AGE   VERSION
multi-node-cluster-control-plane   Ready    control-plane   46d   v1.27.3
multi-node-cluster-worker          Ready    <none>          46d   v1.27.3
multi-node-cluster-worker2         Ready    <none>          46d   v1.27.3
multi-node-cluster-worker3         Ready    <none>          46d   v1.27.3
```

Figure 5: Cluster Nodes

[2] https://kind.sigs.k8s.io/docs/user/quick-start/#installation
[3] https://community.chocolatey.org/packages/kind

3 Serverless Environment Configuration

3.1 Knative Installation

Knative is a crucial element for the deployment and management of serverless workloads on Kubernetes. It streamlines the procedure of building, deploying, and overseeing scalable, serverless services. To deploy Knative, execute instruction available on the official website[4] or follow below:

1. **Install Knative Serving:** To facilitate the deployment and serving of serverless applications and processes, Knative Serving builds upon Kubernetes. Use the following command to install Knative Serving:

```
>kubectl apply -f https://github.com/knative/serving/
    releases/download/knative-v1.12.2/serving-crds.yaml

>kubectl apply -f https://github.com/knative/serving/
    releases/download/knative-v1.12.2/serving-core.yaml
```

2. **Configure Networking Layer:** For Knative serving, select the Istio networking layer. Use these steps to install Istio on your network:

```
>kubectl apply -l knative.dev/crd-install=true -f https
    ://github.com/knative/net-istio/releases/download/
    knative-v1.12.0/istio.yaml
```

The Knative Istio controller can be installed by running the following command:

```
>kubectl apply -f https://github.com/knative/net-istio/
    releases/download/knative-v1.12.0/net-istio.yaml
```

3. **Verify Installation:** Please ensure that each component are in the correct location and are working properly. Virify the pods by using below command in the 'knative-serving' namespace to check its status as running:

```
>kubectl get pods --namespace knative-serving
```

```
PS C:\Users\akash>  kubectl get pods --namespace knative-serving
NAME                                    READY   STATUS      RESTARTS
activator-8d86d5bf4-7rv6k               1/1     Running     48 (75s ago)
autoscaler-557b48445-dpf2s              1/1     Running     64 (2m43s ago)
controller-6656dd8559-75f4n             1/1     Running     96 (17m ago)
default-domain-dmj9m                    0/1     Completed   0
net-istio-controller-5764678bd4-w4c59   1/1     Running     93 (56s ago)
net-istio-webhook-586c8df865-28ds5      1/1     Running     35 (56s ago)
webhook-5cf7fd9584-6ldt6                1/1     Running     96 (2m43s ago)
```

Figure 6: Knative installation verification

[4] https://knative.dev/docs/install/yaml-install/serving/install-serving-with-yaml/

4. **Configure Domain Name:** By default, Knative Serving will utilise the sslip.io DNS suffix, due to a Kubernetes Job called default-domain. Run the following command to add sslip.io as a domain:

```
>kubectl apply -f https://github.com/knative/serving/
    releases/download/knative-v1.12.2/serving-default-
    domain.yaml
```

4 Database and Storage Setup

4.1 MySQL Database Setup on Master Node

To set up the MySQL database on the master node of your Kubernetes cluster, use the ‘mysql-deployment.yaml‘ file available in the services folder in the GitHub repository. Once the file is downloaded update your Master Node name in the file so that the database will be only installed and available on the specified node. Below are the steps for execution are the steps:

1. Once the project is cloned locate the ’mysql-deployment.yaml’ file which has the necessary configuration. This file defines the MySQL deployment, including the container image, environment variables (such as the MySQL root password), storage volumes, and other required settings.

2. Apply the ’mysql-deployment.yaml’ file to your Kubernetes cluster to create the MySQL deployment using the following command:

```
>kubectl apply -f mysql-deployment.yaml
```

3. Verify that the MySQL pod is running correctly:

```
>kubectl get pods
```

   Look for the MySQL pod in the output and ensure it’s in the ’Running’ state as seen in the figure7.

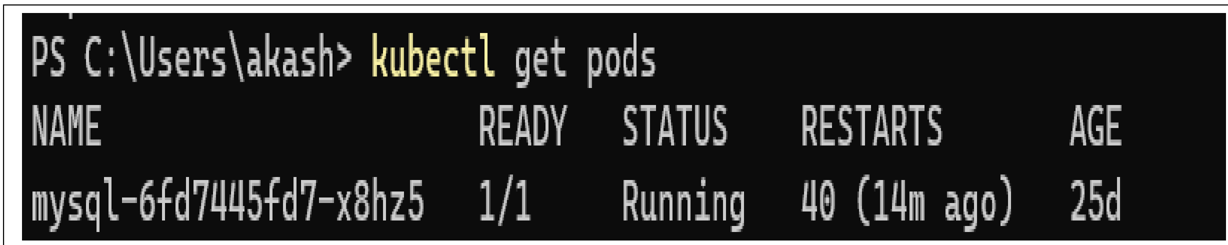


Figure 7: MySQL pod created and running

4. Once the MySQL pod is up and running, you can access the MySQL database from within the cluster. Use the MySQL command-line tool or any other MySQL client to connect to the database.

5. When the deployment is available and pod is in running state, create the necessary database schema and tables for the application. This can be done by connecting to the MySQL pod and executing SQL commands as below:

```
>kubectl port-forward pod/[your-sql-pod-name] [port
  number]
>mysql -u root -p -h [ip-of-you-sql-service] --port=[
  portnumber]
>CREATE DATABASE model_db;
>CREATE TABLE models (id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL, dataset_name VARCHAR
  (255) NOT NULL, architecture VARCHAR(255) NOT NULL,
  accuracy FLOAT NOT NULL, loss FLOAT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  model_data LONGBLOB, model_path MEDIUMTEXT);
```

Once the Database is created you can verify with below commands:

```
>SHOW DATABASES;
```

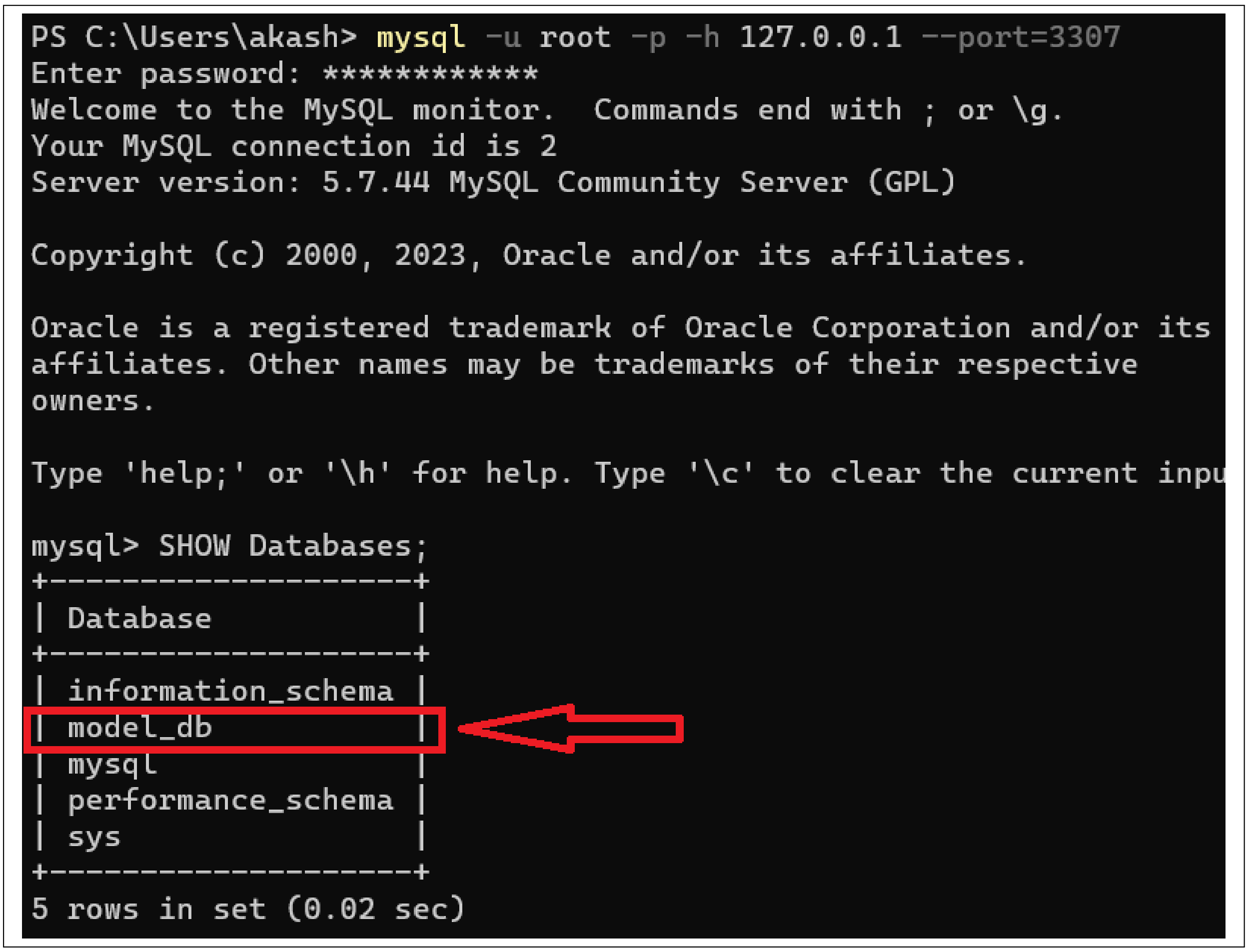
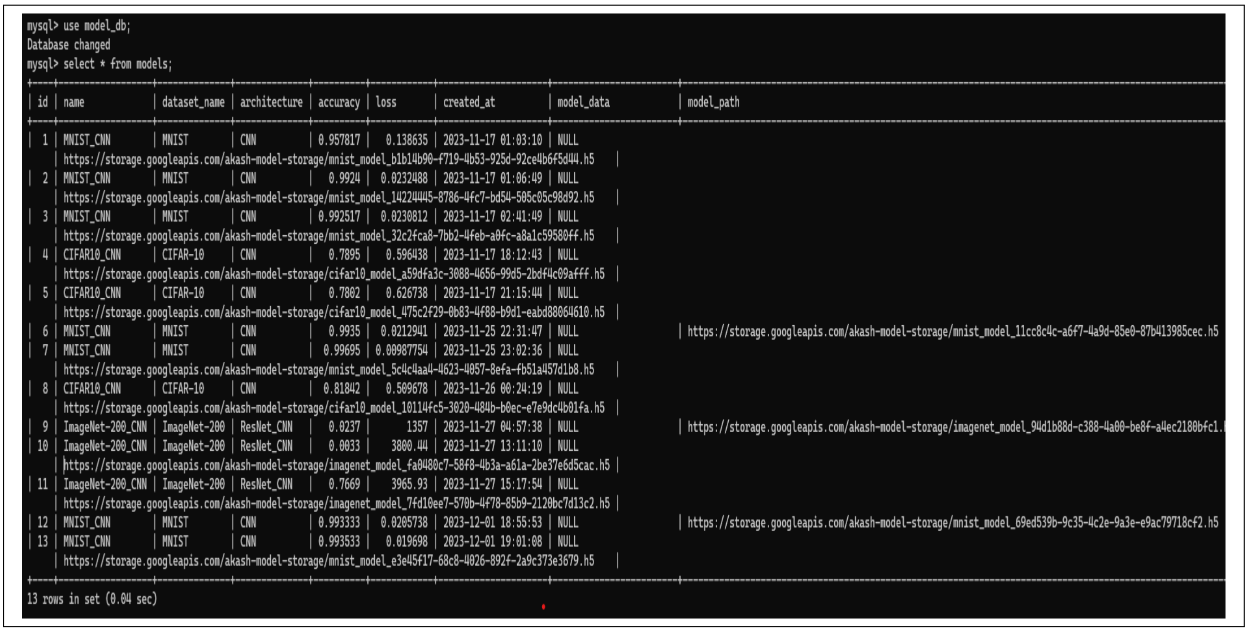


Figure 8: Database Created in MySQL

```
>SELECT * from models;
```



```
mysql> use model_db;
Database changed
mysql> select * from models;
```

id	name	dataset_name	architecture	accuracy	loss	created_at	model_data	model_path
1	MNIST_CNN	MNIST	CNN	0.957817	0.138635	2023-11-17 01:03:10	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_b1b14b90-f719-4b53-925d-92ce4b6f5d44.h5
2	MNIST_CNN	MNIST	CNN	0.9924	0.0232488	2023-11-17 01:06:49	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_14224445-8786-4fc7-bd54-505c05c98d92.h5
3	MNIST_CNN	MNIST	CNN	0.992517	0.0230812	2023-11-17 02:41:49	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_32c2fca8-7bb2-4feb-a0fc-a8a1c59580ff.h5
4	CIFAR10_CNN	CIFAR-10	CNN	0.7895	0.596438	2023-11-17 18:12:43	NULL	https://storage.googleapis.com/akash-model-storage/cifar10_model_a59dfa3c-3088-4656-99d5-2bdf4c09afff.h5
5	CIFAR10_CNN	CIFAR-10	CNN	0.7802	0.626738	2023-11-17 21:15:44	NULL	https://storage.googleapis.com/akash-model-storage/cifar10_model_475c2f29-0b83-4f88-b9d1-eabd88064610.h5
6	MNIST_CNN	MNIST	CNN	0.9935	0.0212941	2023-11-25 22:31:47	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_11cc8c4c-a6f7-4a9d-85e0-87b413985cec.h5
7	MNIST_CNN	MNIST	CNN	0.99695	0.00987754	2023-11-25 23:02:36	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_5c4c4aa4-4623-4057-8efa-fb51a457d1b8.h5
8	CIFAR10_CNN	CIFAR-10	CNN	0.81842	0.509678	2023-11-26 00:24:19	NULL	https://storage.googleapis.com/akash-model-storage/cifar10_model_10114fc5-3020-484b-b0ec-e7e9dc4b01fa.h5
9	ImageNet-200_CNN	ImageNet-200	ResNet_CNN	0.0237	1357	2023-11-27 04:57:38	NULL	https://storage.googleapis.com/akash-model-storage/imagenet_model_94d1b88d-c388-4a00-be8f-a4ec2180bfc1.
10	ImageNet-200_CNN	ImageNet-200	ResNet_CNN	0.0033	3800.44	2023-11-27 13:11:10	NULL	https://storage.googleapis.com/akash-model-storage/imagenet_model_fa0480c7-58f8-4b3a-a61a-2be37e6d5cac.h5
11	ImageNet-200_CNN	ImageNet-200	ResNet_CNN	0.7669	3965.93	2023-11-27 15:17:54	NULL	https://storage.googleapis.com/akash-model-storage/imagenet_model_7fd10ee7-570b-4f78-85b9-2120bc7d13c2.h5
12	MNIST_CNN	MNIST	CNN	0.993333	0.0205738	2023-12-01 18:55:53	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_69ed539b-9c35-4c2e-9a3e-e9ac79718cf2.h5
13	MNIST_CNN	MNIST	CNN	0.993533	0.019698	2023-12-01 19:01:08	NULL	https://storage.googleapis.com/akash-model-storage/mnist_model_e3e45f17-68c8-4026-892f-2a9c373e3679.h5

```
13 rows in set (0.04 sec)
```

Figure 9: Database table Models oberview

4.2 Google Cloud Bucket Setup

Google Cloud Buckets serve as repositories for storing trained models. Below are the steps to setup the bucket:

(a) First, ensure you have a Google Cloud account and have set up a project.

(b) Create a new Google cloud storage bucket: When the project is created you can set up a new cloud bucket by clicking on create bucket and by giving it appropriate name, Location type, storage class and access.

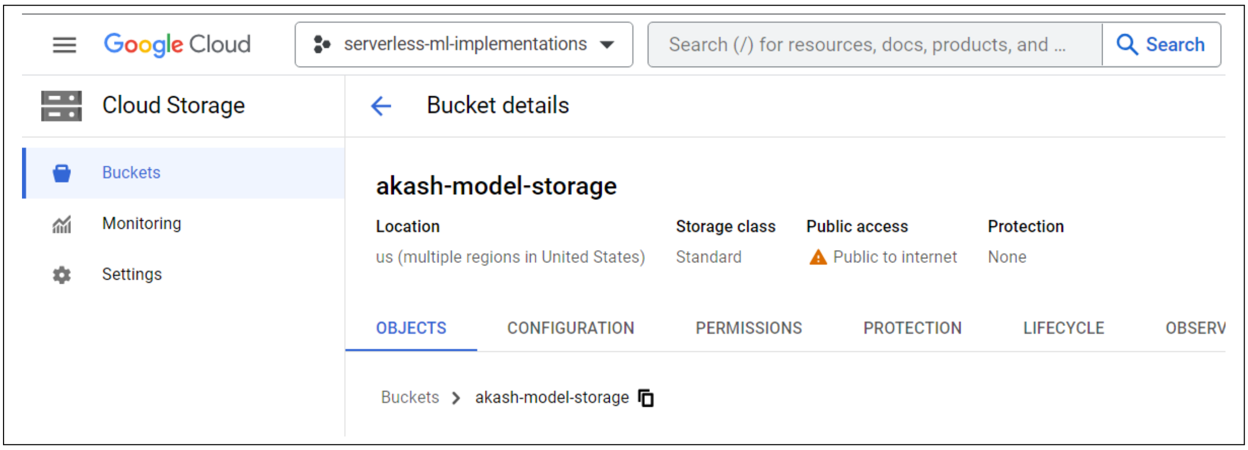


Figure 10: Google Cloud Storage Bucket

(c) To provide public access to the bucket using the code, you must make the bucket public for testing the system.

(d) Post bucket creation, obtain the JSON key file for your Google Cloud account—this file houses the credentials for programmatic access to the bucket.

(e) Securely store the JSON key file and establish an environment variable directing to the file's location, facilitating your application's authentication with Google Cloud services. Insert the path into your environment variables as illustrated:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/your
    /keyfile.json"
```

(f) Deploy Google Cloud dependencies within your Python application to enable interactions with the GCS. The following pip command installs the necessary packages:

```
pip install google-cloud-storage
```

(g) Use the Google Cloud libraries within the application's code to connect to the GCS bucket and facilitate operations such as model upload and retrieval. The 'GOOGLE_APPLICATION_CREDENTIALS' environment variable will authenticate your requests to GCS.

(h) To upload and download models from your application, use the Google Cloud Storage client libraries in your Python scripts. Also, a config map needs to be created to store the JSON file downloaded as secret for accessing the bucket seamlessly.

5 Deploying Serverless Functions from GitHub and DockerHub

How to import code from GitHub[5] repository, package it up, and use it in the Knative environment as serverless functions is explained in this part. Another option is for users to use pre-built files from the DockerHub Image Registry as shown in figure12 and 13. Within the serverless system, YAML files are used for various services during the deployment process.

[5] https://github.com/Akash-Sane/Serverless-Edge-ML-System

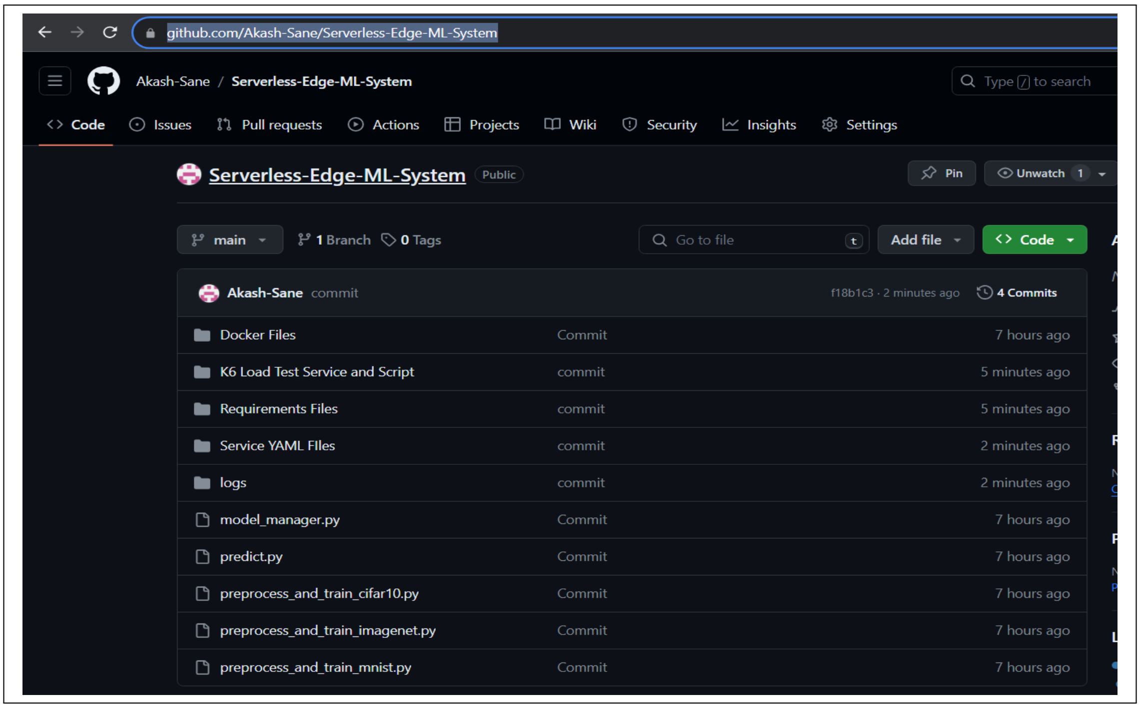


Figure 11: GitHUb Repository

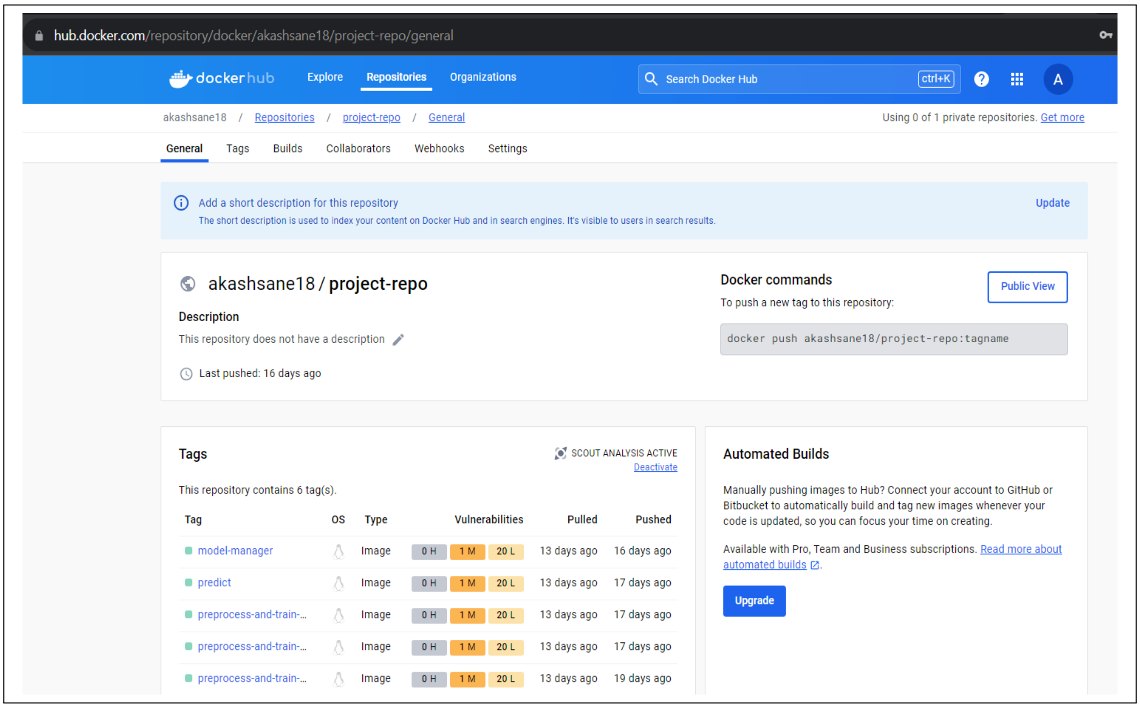


Figure 12: Docker Image Registry

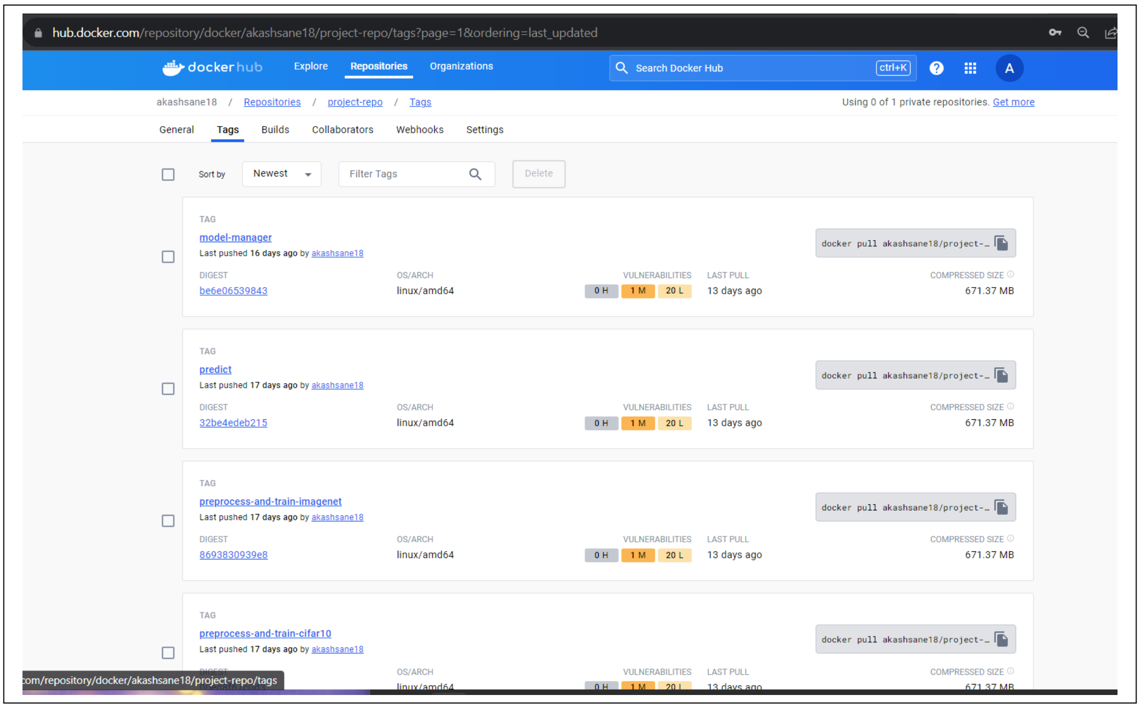


Figure 13: Docker Images

1. **Cloning the Repository:** Begin by cloning the necessary code repositories from GitHub. Use the following command to clone a repository:

```
>git clone <repository-url>
```

2. **Containerizing the Code:** After cloning, navigate to the repository directory and build Docker images for your serverless functions also the required function code will be available files with ".py" extension. Use the Docker CLI to build images but make sure the requirement.txt is in the same location as the docker file so it can pick the required libraries during the build phase of the image:

```
>docker build -t <your-image-name> .
```

   Alternatively, you can pull pre-built images from DockerHub[6] as they are made available for replicating the system.

3. **Pushing Images to DockerHub:** Push your newly built images to DockerHub (or another container registry of your choice):

```
>docker push <your-image-name>
```

4. **Deploying Serverless Functions:** Once the images are ready in the registry, deploy them as serverless functions using Knative. Apply the YAML files for each service using 'kubectl' but make sure that the requirement.txt files is in the same location as these YAML files:

[6] `https://hub.docker.com/repository/docker/akashsane18/project-repo/general`

```
> kubectl apply -f service-mnist.yaml
> kubectl apply -f service-cifar10.yaml
> kubectl apply -f service-imagenet.yaml
> kubectl apply -f service-predict.yaml
> kubectl apply -f service-model.yaml
```

Figure 14: Deployment of Serverless Function

5. **Verifying the Deployment:** Ensure that the services are running correctly in your Knative environment. Check the status of the deployments using:

```
>kubectl get pods
```

```
model-manager-00001-deployment-65c79676cb-8c5gb                      3/3    Running
mysql-6fd7445fd7-x8hz5                                               1/1    Running
predict-00001-deployment-769b4dd5d7-8sgrp                            3/3    Running
preprocess-and-train-cifar10-00001-deployment-f87599c85-5fmq7        3/3    Running
preprocess-and-train-imagenet-00001-deployment-54d99c4df9-xlvk7      3/3    Running
preprocess-and-train-mnist-00001-deployment-6f7c76f74d-f9dhq         3/3    Running
```

Figure 15: Verifying the Pods from Once the Service is created

```
>kubectl get all
```

```
NAME                                                               LATESTCREATED                         LATESTREADY                           READY   REASONconfiguration.serving.knative.de
v/model-manager                       model-manager-00001                model-manager-00001                True
configuration.serving.knative.dev/predict                          predict-00001                         predict-00001                         True
configuration.serving.knative.dev/preprocess-and-train-cifar10     preprocess-and-train-cifar10-00001    preprocess-and-train-cifar10-00001    True
configuration.serving.knative.dev/preprocess-and-train-imagenet    preprocess-and-train-imagenet-00001   preprocess-and-train-imagenet-00001   True
configuration.serving.knative.dev/preprocess-and-train-mnist       preprocess-and-train-mnist-00001      preprocess-and-train-mnist-00001      True

NAME                                                                CONFIG NAME                      K8S SERVICE NAME   GENERATION   READY   REASON   ACTUAL REPLICAS   DESIRED REPLICAS
revision.serving.knative.dev/model-manager-00001                    model-manager                                       1            True             0                 0
revision.serving.knative.dev/predict-00001                          predict                                             1            True             0                 0
revision.serving.knative.dev/preprocess-and-train-cifar10-00001     preprocess-and-train-cifar10                        1            True             0                 0
revision.serving.knative.dev/preprocess-and-train-imagenet-00001    preprocess-and-train-imagenet                       1            True             0                 0
revision.serving.knative.dev/preprocess-and-train-mnist-00001       preprocess-and-train-mnist                          1            True             0                 0

NAME                                                       URL                                                                  READY   REASON
route.serving.knative.dev/model-manager                    http://model-manager.default.172.18.1.101.sslip.io                   True
route.serving.knative.dev/predict                          http://predict.default.172.18.1.101.sslip.io                         True
route.serving.knative.dev/preprocess-and-train-cifar10     http://preprocess-and-train-cifar10.default.172.18.1.101.sslip.io    True
route.serving.knative.dev/preprocess-and-train-imagenet    http://preprocess-and-train-imagenet.default.172.18.1.101.sslip.io   True
route.serving.knative.dev/preprocess-and-train-mnist       http://preprocess-and-train-mnist.default.172.18.1.101.sslip.io      True

NAME                                                         URL                                                                  LATESTCREATED
           LATESTREADY                                 READY   REASON
service.serving.knative.dev/model-manager                    http://model-manager.default.172.18.1.101.sslip.io                   model-manager-00001                   model-manager-00001
                 True
service.serving.knative.dev/predict                          http://predict.default.172.18.1.101.sslip.io                         predict-00001
           predict-00001                               True
service.serving.knative.dev/preprocess-and-train-cifar10     http://preprocess-and-train-cifar10.default.172.18.1.101.sslip.io    preprocess-and-train-cifar10-00001    preprocess-and-train-
cifar10-00001    True
service.serving.knative.dev/preprocess-and-train-imagenet    http://preprocess-and-train-imagenet.default.172.18.1.101.sslip.io   preprocess-and-train-imagenet-00001   preprocess-and-train-
imagenet-00001   True
service.serving.knative.dev/preprocess-and-train-mnist       http://preprocess-and-train-mnist.default.172.18.1.101.sslip.io      preprocess-and-train-mnist-00001      preprocess-and-train-
mnist-00001      True
```

Figure 16: Deployed Services Running In Serverless Environment

This method will it easy to deploy serverless functions from GitHub repositories or directly from DockerHub. This makes sure that your machine learning apps are set up correctly in a Knative environment with multiple nodes.

6 Load Testing with k6 and InfluxDB

Using the k6 tool with InfluxDB in a Kubernetes environment for load testing is explained in this part. InfluxDB will be used to store the logs from the test cases for later analysis. The objective is to test the serverless functions under different load conditions.

6.1 Configuring k6 Job in Kubernetes

1. **Job Definition:** Create a Kubernetes job to conduct the k6 load test. The job configuration will use the 'loadimpact/k6' image and provide the script and InfluxDB details for storing the job's logs. The following is the YAML setup for the k6 job:

6.2 Configuring k6 Job in Kubernetes

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: k6-load-test
spec:
  template:
    spec:
      containers:
      - name: k6
        image: loadimpact/k6
        command: ["k6", "run", "--out", "influxdb=http://influxdb.monitoring.svc.cluster.local:8086/myk6db", "/scripts/loadtest.js"]
        volumeMounts:
        - name: script-volume
          mountPath: /scripts
      volumes:
      - name: script-volume
        configMap:
          name: loadtest-script
      restartPolicy: Never
  backoffLimit: 4
```

Figure 17: k6 Job Configuration

2. **Load Test Script Preparation:** Prepare your k6 load test script, for instance, 'loadtest.js', which contains your load testing logic or the current script of loadtest.js available in the repository can be used and updated as per the testing requirements.

3. **Creating ConfigMap for Load Test Script:** Create a Kubernetes ConfigMap for storing the load test script. This allows the script to be easily accessed within

the cluster. Use the following command to generate a ConfigMap from your script file:

```
>kubectl create configmap loadtest-script --from-file=loadtest.js
```

4. **Creating the Job:** Apply the above YAML file to your Kubernetes cluster using the following command:

```
>kubectl apply -f k6-job.yaml
```

5. **Monitoring the Job:** Monitor the execution of the job using:

```
>kubectl get jobs
```

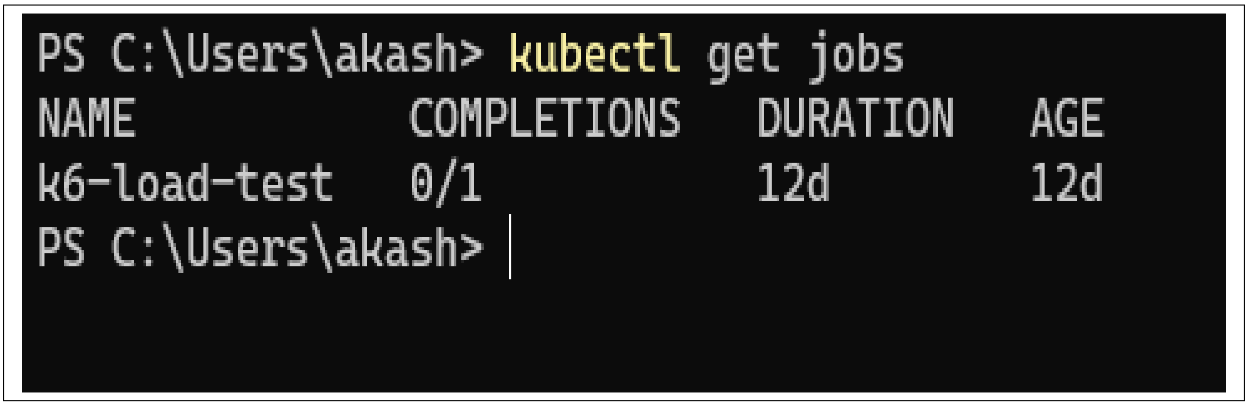
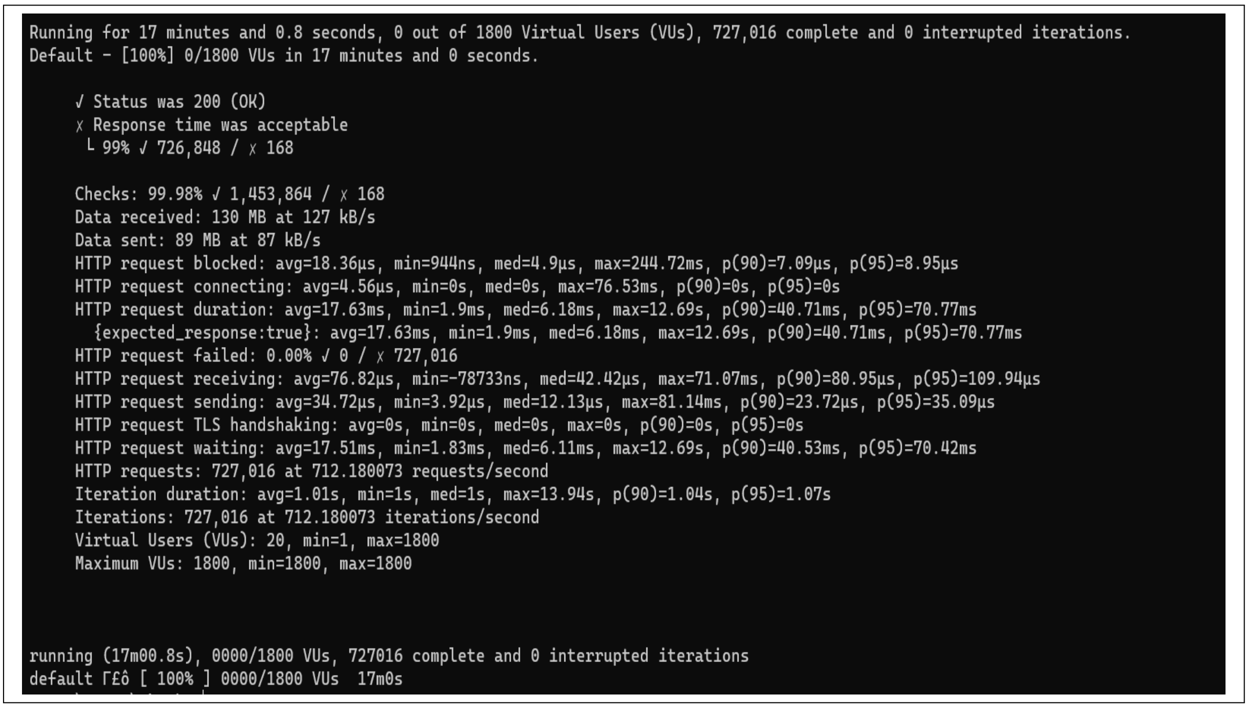


Figure 18: K8 Job Deployment

Figure 19: k6 job pod running while executing the job

6. **K6 Job Verification:** The Job execution can be found by viewing the k6 pod logs by executing the below command as seen in the figure20.

```
>kubectl logs pod <K6 pod Name>
```



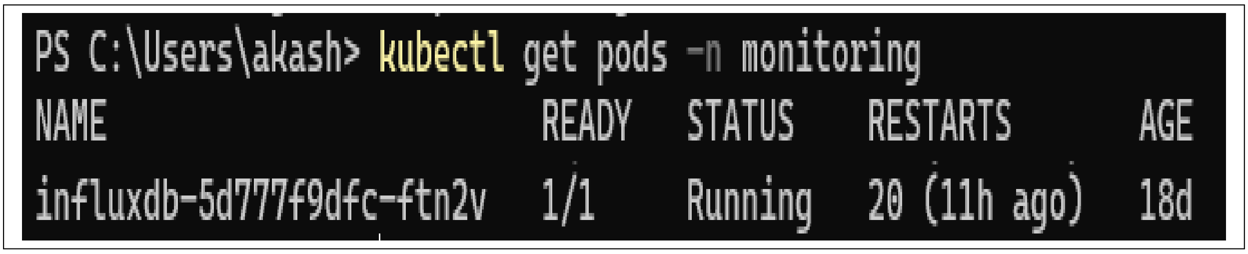
```
Running for 17 minutes and 0.8 seconds, 0 out of 1800 Virtual Users (VUs), 727,016 complete and 0 interrupted iterations.
Default - [100%] 0/1800 VUs in 17 minutes and 0 seconds.

     √ Status was 200 (OK)
     ✗ Response time was acceptable
      └ 99% √ 726,848 / ✗ 168

     Checks: 99.98% √ 1,453,864 / ✗ 168
     Data received: 130 MB at 127 kB/s
     Data sent: 89 MB at 87 kB/s
     HTTP request blocked: avg=18.36µs, min=944ns, med=4.9µs, max=244.72ms, p(90)=7.09µs, p(95)=8.95µs
     HTTP request connecting: avg=4.56µs, min=0s, med=0s, max=76.53ms, p(90)=0s, p(95)=0s
     HTTP request duration: avg=17.63ms, min=1.9ms, med=6.18ms, max=12.69s, p(90)=40.71ms, p(95)=70.77ms
       {expected_response:true}: avg=17.63ms, min=1.9ms, med=6.18ms, max=12.69s, p(90)=40.71ms, p(95)=70.77ms
     HTTP request failed: 0.00% √ 0 / ✗ 727,016
     HTTP request receiving: avg=76.82µs, min=-78733ns, med=42.42µs, max=71.07ms, p(90)=80.95µs, p(95)=109.94µs
     HTTP request sending: avg=34.72µs, min=3.92µs, med=12.13µs, max=81.14ms, p(90)=23.72µs, p(95)=35.09µs
     HTTP request TLS handshaking: avg=0s, min=0s, med=0s, max=0s, p(90)=0s, p(95)=0s
     HTTP request waiting: avg=17.51ms, min=1.83ms, med=6.11ms, max=12.69s, p(90)=40.53ms, p(95)=70.42ms
     HTTP requests: 727,016 at 712.180073 requests/second
     Iteration duration: avg=1.01s, min=1s, med=1s, max=13.94s, p(90)=1.04s, p(95)=1.07s
     Iterations: 727,016 at 712.180073 iterations/second
     Virtual Users (VUs): 20, min=1, max=1800
     Maximum VUs: 1800, min=1800, max=1800



running (17m00.8s), 0000/1800 VUs, 727016 complete and 0 interrupted iterations
default Γ£ô [ 100% ] 0000/1800 VUs  17m0s
```

Figure 20: k6 pod log showing completed job executing

6.3 Setting Up InfluxDB

InfluxDB is used to store and analyze the results of load tests. Follow these steps to configure InfluxDB:

1. **InfluxDB Installation:** Deploy InfluxDB of version 1.8 within your Kubernetes cluster. To use the official InfluxDB Docker image, you can set up a deployment within Kubernetes or us the available file named influxdb.yaml which is can be seen in figure22:



```
PS C:\Users\akash> kubectl get pods -n monitoring
NAME                        READY   STATUS    RESTARTS       AGE
influxdb-5d777f9dfc-ftn2v   1/1     Running   20 (11h ago)   18d
```

Figure 21: InfluxDB POD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: influxdb
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: influxdb
  template:
    metadata:
      labels:
        app: influxdb
    spec:
      containers:
      - name: influxdb
        image: influxdb:1.8
        ports:
        - containerPort: 8086
---
apiVersion: v1
kind: Service
metadata:
  name: influxdb
  namespace: monitoring
spec:
  type: ClusterIP
  ports:
    - port: 8086
  selector:
    app: influxdb
```

Figure 22: InfluxDB Config Configuration

2. **Logging Database Configuration:** Once the InfluxDB and its service is running create a database named 'myk6db' in InfluxDB for storing the k6 test results.

3. **Accessing Data:** Access and verify at the logs using InfluxDB's Web UI or CLI. The database can also be access locally by port-forwarding the service to available ports. Querying is an option to get useful information from the load test results, which are made available after the k6 job is finished.

Note: Verify that the InfluxDB service is properly exposed and can be accessed from the k6 job within the Kubernetes cluster, or assign the necessary port for port forwarding to access the database.

7 Monitoring and Performance Evaluation

7.1 System Monitoring

- **Kubernetes Dashboard:** The Kubernetes Dashboard lets you keep tabs on your Kubernetes cluster's health and performance as a whole. To install Kubernetes, just follow the steps given on the website[7]. It gives useful details about how resources are being used, the status of pods, and system events.

- **Accessing the Dashboard:** Use the 'kubectl proxy' command to access the Kubernetes Dashboard23. Navigate to the provided URL to view the cluster's

[7]https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

status. Also, the kubernetese dashboard visualization is available from Docker for desktop as seen in below figure24.

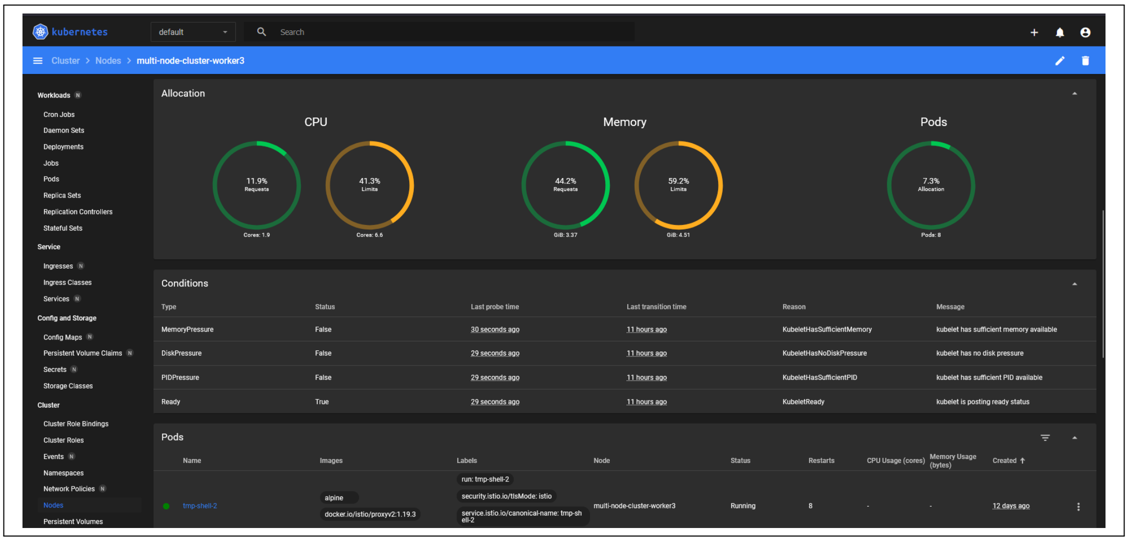


Figure 23: Kubernetes Dashboard

Figure 24: Docker For Desktop K8s Dashboard for Master Node

7.2 Performance Metrics

- **Evaluating Test Results:** Analyze the load test results stored in InfluxDB. Focus on key performance indicators such as response times, error rates, and throughput to evaluate the resilience and scalability of your serverless functions. The scalability of this system can be analyzed from the Knative serving logs for how the pods are getting scaled based on the scaling metric defined in the services deployed in the environment. autoscaling.knative.dev/metric: "concurrency": This annotation seen in the below service deployment7.2 gives the Knative autoscaler's measure for scaling the application. Using "concurrency" as the metric . This indicates the autoscaler will count concurrent requests per application instance. Concurrency-based scaling is utilised when a pod's ability to handle many requests is a better load indicator than CPU or memory utilisation. autoscaling.knative.dev/target: "200": Sets pod average concurrency. Knative should aim for 200 concurrent requests per pod with "200". Knative creates more pods to accommodate the load when the average number of concurrent requests exceeds this threshold. Knative may limit pods if concurrent requests consistently fall below this target.

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: preprocess-and-train-mnist
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/metric: "concurrency"
        autoscaling.knative.dev/target: "200"
    spec:
      containers:
      - image: akashsane18/project-repo:preprocess-and-train-mnist
        ports:
        - containerPort: 80
        env:
        - name: GOOGLE_APPLICATION_CREDENTIALS
          value: /var/secrets/google/key.json
        readinessProbe:
          httpGet:
            path: /health
        volumeMounts:
        - mountPath: /var/secrets/google
          name: google-credentials-volume
          readOnly: true
      volumes:
      - name: google-credentials-volume
        secret:
          secretName: google-credentials
```

Figure 25: Autoscaler Configuration in Service

- **Optimization Based on Data:** Utilize the insights from the performance data to optimize the serverless environment. Adjust configurations, scale resources, and refine serverless functions as needed.

Note: The process of load testing and performance evaluation is defined by iteration. Consistent monitoring and review are crucial for the purpose of analysis and producing insightful information.