National College of Ireland

Configuration Manual

MSc. Research Project
Cloud Computing

Ankur Shroff

Student ID: 23426659

School of Computing

National College of Ireland

Supervisor: Mr Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankur Shroff
Student ID:	23426659
Programme:	MSc Cloud Computing **Year:** Jan 2025 - 1
Module:	MSc. Research Project Cloud Computing
Supervisor:	Mr. Abubakr Siddig
Submission Due Date:	11th December 2025
Project Title:	Machine Learning – Driven Auto Scaling in Kubernetes for Cloud Resource Optimization
Word Count:	…… **Page Count: 10**

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: ……

Date: 11th Dec 2025

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

Ankur Shroff
Student ID: 23426659

1 Setting up Simple Django Web Application

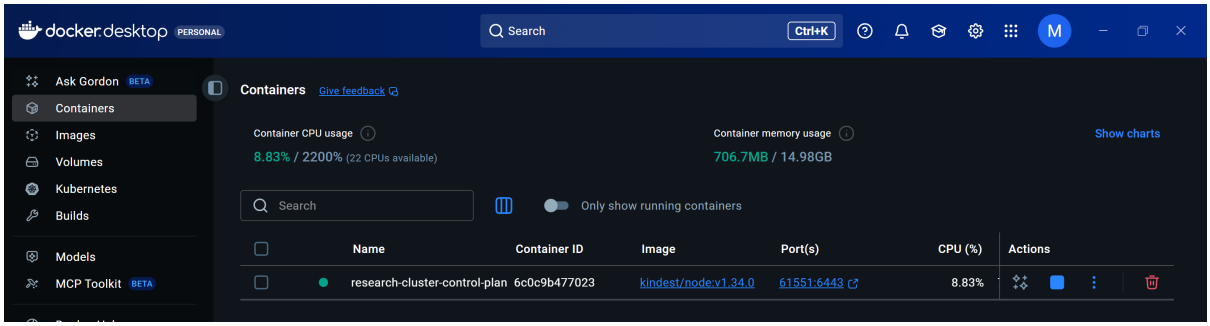
1.1 Introduction

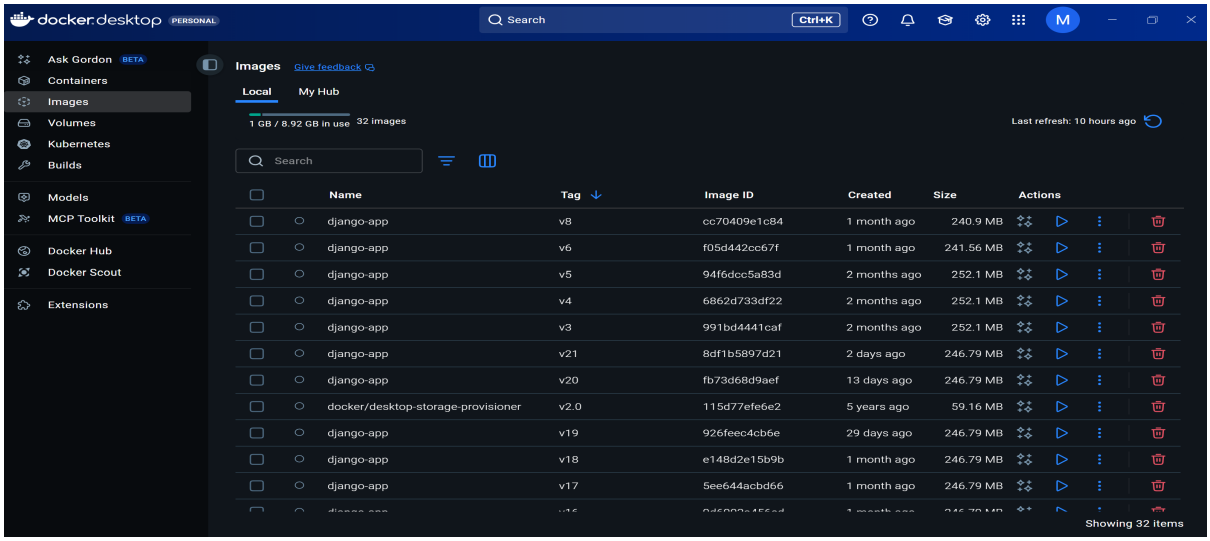
This section provides comprehensive and step-by-step guide to create, containerize, and deploy this simple Django web application used for different phases workload in this research. The application is designed as simple without any state for CPU-intensive service that generates a measurable load within a Kubernetes environment.

1.2 Prerequisites

Before proceeding, ensure the following software is installed and configured on your local workstation:

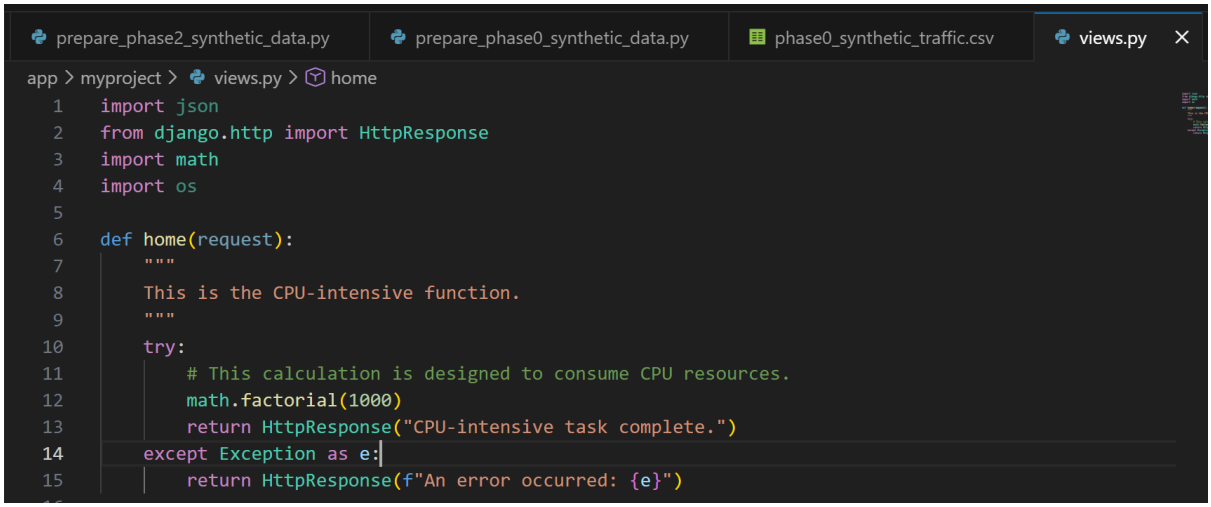
- Python 3.x with pip
- Django app
- Docker Desktop (or another Docker daemon, for this docker desktop is used)
- kubectl configured to communicate with your target Kubernetes cluster

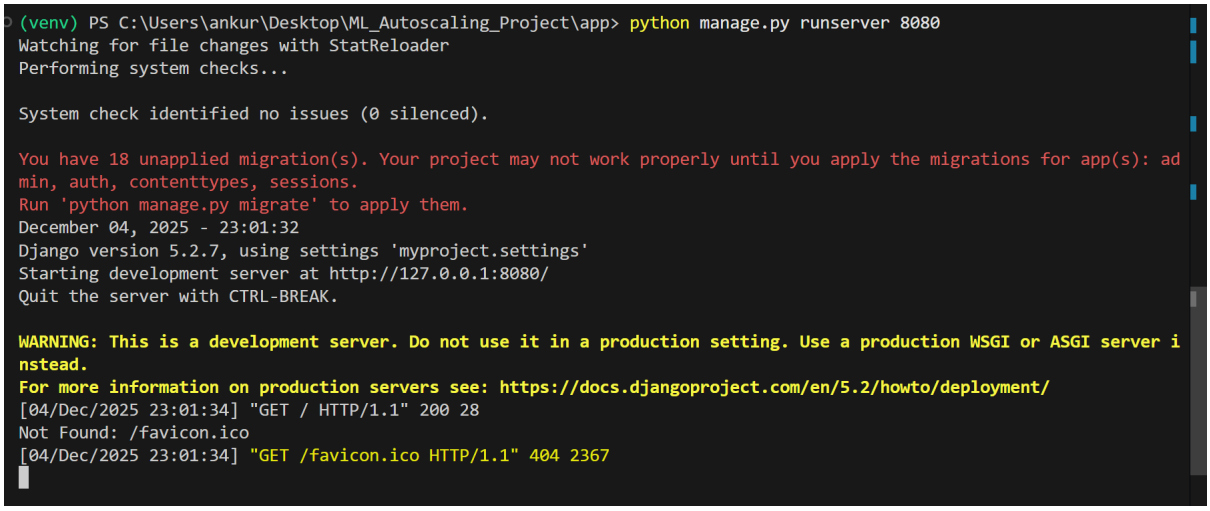


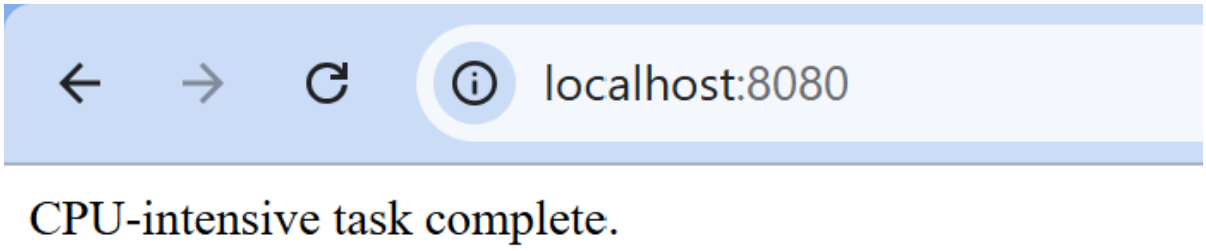


1.3 Building and running python web application

Building and running the simple python CPU intensive app –







2 Building docker image and deploying application

1.1 Create docker file – dockerfile

This file will tell the docker to build the image using these python version, configuration and app files, and to run gunicorn server in port 8000

nthetic_data.py | prepare_phase0_synthetic_data.py | phase0_synthetic_traffic.csv | views.py | dockerfile 1

dockerfile > ...

```
# Use an official lightweight Python image as our base
FROM python:3.9.18-slim-bullseye

# Set environment variables for Python
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

# Set the working directory inside the container
WORKDIR /code

# Copy ONLY the requirements file first
COPY app/requirements.txt /code/

# Install the Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Now copy the rest of the application code
COPY app/ /code/

# Command to run the Gunicorn server, exposing it on port 8000
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "--workers", "4", "--timeout", "120", "myproject.wsgi:app
```

1.2 Building docker image and deploying the django app application

- Run command in terminal –
  Docker build -t djnago-app:21 .

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> docker build -t django-app:v21 .
[+] Building 2.6s (11/11) FINISHED                                                    docker:desktop-linux
 => [internal] load build definition from dockerfile                                                0.1s
 => => transferring dockerfile: 700B                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.9.18-slim-bullseye                      1.3s
 => [auth] library/python:pull token for registry-1.docker.io                                       0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [1/5] FROM docker.io/library/python:3.9.18-slim-bullseye@sha256:9ac27d4ecadc3ef02f980a8e2b37c7e8cdbf24039  0.1s
 => => resolve docker.io/library/python:3.9.18-slim-bullseye@sha256:9ac27d4ecadc3ef02f980a8e2b37c7e8cdbf24039  0.0s
 => [internal] load build context                                                                   0.1s
 => => transferring context: 6.97kB                                                                 0.0s
 => CACHED [2/5] WORKDIR /code                                                                      0.0s
 => CACHED [3/5] COPY app/requirements.txt /code/                                                   0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                 0.0s
 => [5/5] COPY app/ /code/                                                                          0.1s
 => exporting to image                                                                              0.5s
 => => exporting layers                                                                             0.2s
 => => exporting manifest sha256:4434a5e08f2764dabe6712061d0ee784aa13e3a53a05b780492f46ade11ec1ba    0.0s
 => => exporting config sha256:c8d006307509b3bffb5fec9e1ccf86f09d8c6df1ec3381f8dcdc53f3bf02cb3a      0.0s
 => => exporting attestation manifest sha256:8ff919391f913a7979a9051251277e644c13928f041222ebea15a33393d0c22a  0.1s
 => => exporting manifest list sha256:8df1b5897d210fa29dccdda449a4f7385bb414b444b3c5e97949d64d4d5c5d54  0.0s
 => => naming to docker.io/library/django-app:v21                                                   0.0s
 => => unpacking to docker.io/library/django-app:v21                                                0.1s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/f0758uxuz6meckcbhh6qosand
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Check all the docker images, run command in terminal –
  Run following command in terminal
  Docker images

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> docker images
                                                                                                    i Info →  U  In Use
IMAGE                                                                                ID             DISK USAGE   CONTENT SIZE   EXTRA
django-app:v1                                                                        244b15a9af74       238MB         51.8MB
django-app:v12                                                                       0d0e0b425bfc       247MB           54MB
django-app:v14                                                                       907e867e33d6       247MB           54MB
django-app:v15                                                                       1b1a710c1002       247MB           54MB
django-app:v16                                                                       9d6902a456cd       247MB           54MB
django-app:v17                                                                       5ee644acbd66       247MB           54MB
django-app:v18                                                                       e148d2e15b9b       247MB           54MB
django-app:v19                                                                       926feec4cb6e       247MB           54MB
django-app:v20                                                                       fb73d68d9aef       247MB           54MB
django-app:v21                                                                       bcf15797d997       247MB           54MB
django-app:v3                                                                        991bd4441caf       252MB         55.2MB
django-app:v4                                                                        6862d733df22       252MB         55.2MB
django-app:v5                                                                        94f6dcc5a83d       252MB         55.2MB
django-app:v6                                                                        f05d442cc67f       242MB         52.4MB
django-app:v8                                                                        cc70409e1c84       241MB         52.3MB
docker/desktop-cloud-provider-kind:v0.3.0-desktop.3                                  c321a5c92549       582MB          157MB
docker/desktop-containerd-registry-mirror:v0.0.2                                     aadc48ce8960      46.4MB           15MB
docker/desktop-kubernetes:kubernetes-v1.32.2-cni-v1.6.0-critools…                    fdd1722efdcc       596MB          183MB
docker/desktop-storage-provisioner:v2.0                                              115d77efe6e2      59.2MB         17.3MB
docker/desktop-vpnkit-controller:dc331cb22850be0cdd97c84a9cfecaf…                    7ecf567ea070        47MB         10.8MB
docker/welcome-to-docker:latest                                                      c4d56c24da4f      22.2MB         6.03MB
envoyproxy/envoy:v1.32.6                                                             16a413173825       224MB         57.4MB
kindest/node:v1.31.1                                                                 cd224d8da58d      1.49GB          437MB
kindest/node@sha256:7416a61b42b1662ca6ca89f02028ac133a309a2a30ba…                    7416a61b42b1      1.45GB          445MB    U
metrics-api:v1                                                                       c0319735a832       252MB         55.2MB
registry.k8s.io/coredns/coredns:v1.11.3                                              9caabbf6238b      85.1MB         18.6MB
registry.k8s.io/etcd:3.5.16-0                                                        c6a9d11cc5c0       211MB         57.7MB
registry.k8s.io/kube-apiserver:v1.32.2                                               c47449f3e751       129MB         28.7MB
registry.k8s.io/kube-controller-manager:v1.32.2                                      399aa50f4d13       119MB         26.3MB
registry.k8s.io/kube-proxy:v1.32.2                                                   83c025f0faa6       129MB         30.9MB
registry.k8s.io/kube-scheduler:v1.32.2                                               45710d74cfd5      93.5MB         20.7MB
registry.k8s.io/pause:3.10                                                           ee6521f290b2      1.06MB          318kB
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Create kind cluster and load image in cluster
  Run following command in terminal –
  - Kind create cluster –name research-cluster : to create cluster with particular name
  - Kind cluster-info : To check the cluster running or not and in which port
  - Kind get clusters : To check if it exists with the name created
  - Kind load docker-image Django-app:v21 --name research-cluster

```
PROBLEMS 10   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                                    powershell

PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kind create cluster --name research-cluster
Creating cluster "research-cluster" ...
 • Ensuring node image (kindest/node:v1.34.0) 🖼 ...
 ✓ Ensuring node image (kindest/node:v1.34.0) 🖼
 • Preparing nodes 📦   ...
 ✓ Preparing nodes 📦
 • Writing configuration 📜  ...
 ✓ Writing configuration 📜
 • Starting control-plane 🕹️ ...
 ✓ Starting control-plane 🕹️
 • Installing CNI 🔌  ...
 ✓ Installing CNI 🔌
 • Installing StorageClass 💾  ...
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-research-cluster"
You can now use your cluster with:

kubectl cluster-info --context kind-research-cluster

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#community 😊
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:61551
CoreDNS is running at https://127.0.0.1:61551/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kind get clusters
research-cluster
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl load docker-image django-app:v21 --name research-cluster
error: unknown command "load" for "kubectl"

Did you mean this?
        logs
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kind load docker-image django-app:v21 --name research-cluster
Image: "django-app:v21" with ID "sha256:8df1b5897d210fa29dccdda449a4f7385bb414b444b3c5e97949d64d4d5c5d54" not yet prese
control-plane", loading...
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Add helm repo and update helm repo
  Run following command in terminal –
  - Choco install Kubernetes-helm -y
  - helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
  - helm repo update

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>  Choco install Kubernetes-helm -y
Chocolatey v2.4.2
Installing the following packages:
Kubernetes-helm
By installing, you accept licenses for the packages.
kubernetes-helm v3.18.6 already installed.
 Use --force to reinstall, specify a version to install, or try upgrade.

Chocolatey installed 0/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Warnings:
 - kubernetes-helm - kubernetes-helm v3.18.6 already installed.
 Use --force to reinstall, specify a version to install, or try upgrade.

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
 https://chocolatey.org/compare
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo add prometheus-community https:
//prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "prometheus-adapter" chart repository
Update Complete. ⎈Happy Helming!⎈
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Install Prometheus kube stack for monitoring
  - Create file monitoring-values-standard.yaml

monitoring-values-standard.yaml | train_phase3_prophet_model.py | predict_phase3_prophet.py

monitoring-values-standard.yaml > { } prometheus > { } prometheusSpec > [ ] additionalScrapeConfigs > { } 0 > [ ] static_cor

```yaml
# THIS IS THE STANDARD, CLEAN CONFIGURATION FOR PHASES 1, 2, and 3
prometheus-adapter:
  # Explicitly disable the adapter for all other phases to guarantee isolation
  enabled: false

prometheus:
  prometheusSpec:
    additionalScrapeConfigs:
    - job_name: 'pushgateway'
      scrape_interval: 15s
      honor_labels: true
      static_configs:
      - targets: ['pushgateway.default.svc.cluster.local:9091']
```

  - Run command to install the monitoring in terminal –
    helm install prometheus prometheus-community/kube-prometheus-stack --values monitoring-values-standard.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm install prometheus prometheus-commun
 ity/kube-prometheus-stack --values monitoring-values-standard.yaml
I1206 16:08:30.387376   25504 warnings.go:110] "Warning: spec.SessionAffinity is ignored fo
r headless services"
NAME: prometheus
LAST DEPLOYED: Sat Dec  6 16:08:18 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace default get secrets prometheus-grafana -o jsonpath="{.data.admin-pass
word}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace default get pod -l "app.kubernetes.io/name=grafana,
app.kubernetes.io/instance=prometheus" -oname)
  kubectl --namespace default port-forward $POD_NAME 3000

Get your grafana admin user password by running:

  kubectl get secret --namespace default -l app.kubernetes.io/component=admin-secret -o jso
npath="{.items[0].data.admin-password}" | base64 --decode ; echo


Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to cre
ate & configure Alertmanager and Prometheus instances using the Operator.
```

- Fetch Grafana admin password to access the Grafana dashboard
  Run following command in terminal –
  $passwordBytes = kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}';
  [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($passwordBytes))

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>  $passwordBytes = kubectl get secret prom
etheus-grafana -o jsonpath='{.data.admin-password}'; [System.Text.Encoding]::UTF8.GetString
([System.Convert]::FromBase64String($passwordBytes))
Poa0LbZFnVbCV9HGBk0pdd8ByBTIL3cfKM4BXRA7
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Create Deployment, services, pushgatway yaml file for kubernetes
  - Django-app-deployment.yaml

run_phase0_load_generator.sh | monitoring-values-phase0.yaml | django-app-deployment.yaml

django-app-deployment.yaml > { } spec > { } template > { } metadata > { } labels

io.k8s.api.apps.v1.Deployment (v1@deployment.json)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-app-deployment
spec:
  replicas: 3 # HPA will control this, but 3 is a decent starting point
  selector:
    matchLabels:
      app: django-app
  template:
    metadata:
      labels:
        app: django-app
    spec:
      containers:
      - name: django-app-container
        image: django-app:v21
        imagePullPolicy: Never
        ports:
        - containerPort: 8000
        resources:
          requests:
            cpu: "512m"
            memory: "256Mi"
          # "500m" (half a core) to "1000m" (one full core) are common startin
          # for a busy application, depending on the node capacity.
          limits:
            cpu: "600m" # INCREASE THIS - as per the core required
            memory: "512Mi" # Keep memory as is, or increase if logs show OOM
```

- Django-services.yaml

or.sh | monitoring-values-phase0.yaml | django-app-deployment.yaml | *django-service.yaml*

django-service.yaml > { } metadata > { } annotations > abc prometheus.io/path

io.k8s.api.core.v1.Service (v1@service.json)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: django-app-service
  # These annotations are CRITICAL. They tell the Prometheus
  # that is running in our cluster to start "scraping" this service.
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/path:   '/metrics'
    prometheus.io/port:   '8000'
spec:
  selector:
    # This selector is the key. It looks for any Pods
    # that have the label "app: django-app". Our django-deployment.yaml
    # gives our pods this exact label.
    app: django-app
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8000
```

- Pushwgatway.yaml

process_alibaba_data_scaling.py | process_alibaba_data.py | process_data.py | pushgateway.yaml 1

pushgateway.yaml > { } spec > { } template > { } spec > [ ] containers > { } 0

io.k8s.api.core.v1.Service (v1@service.json) | io.k8s.api.apps.v1.Deployment (v1@deployment.json)

```yaml
# ML Experiment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pushgateway
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pushgateway
  template:
    metadata:
      labels:
        app: pushgateway
    spec:
      containers:
      - name: pushgateway
        image: prom/pushgateway:v1.8.0
        # This flag enables the admin API so we can wipe stale data.
        args:
        - "--web.enable-admin-api"
        ports:
        - containerPort: 9091
---
apiVersion: v1
kind: Service
metadata:
  name: pushgateway
  labels:
    app: pushgateway
spec:
  selector:
    app: pushgateway
  ports:
  - name: http
    port: 9091
    targetPort: 9091
```

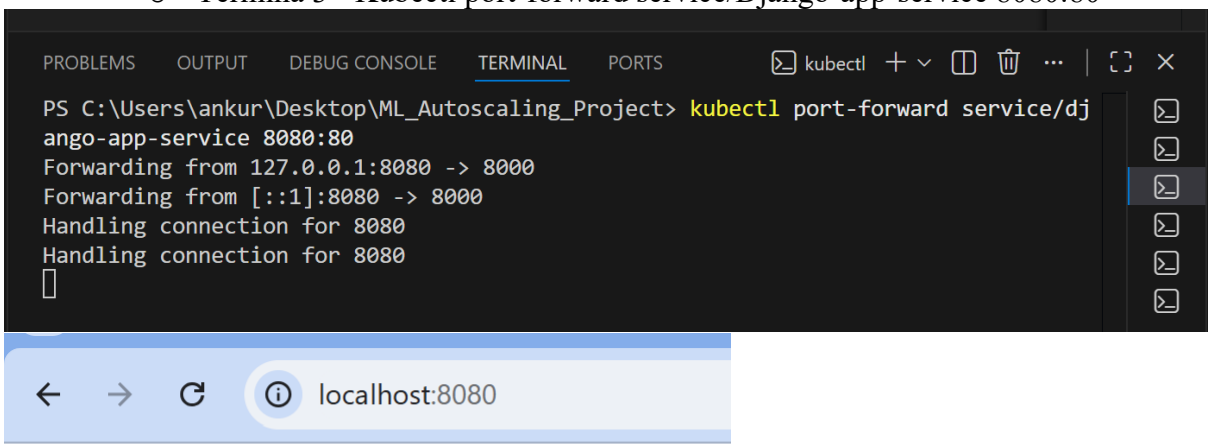
- Apply deployment, service, and pushgatway yaml file

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f django-app-deployment.yaml
deployment.apps/django-app-deployment created
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f django-service.yaml
service/django-app-service created
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f pushgateway.yaml
deployment.apps/pushgateway created
service/pushgateway created
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

- Check all the deployment and services are active and running
  Run the following command to check in new terminal
  Terminal 2 – kubectl get pods -w

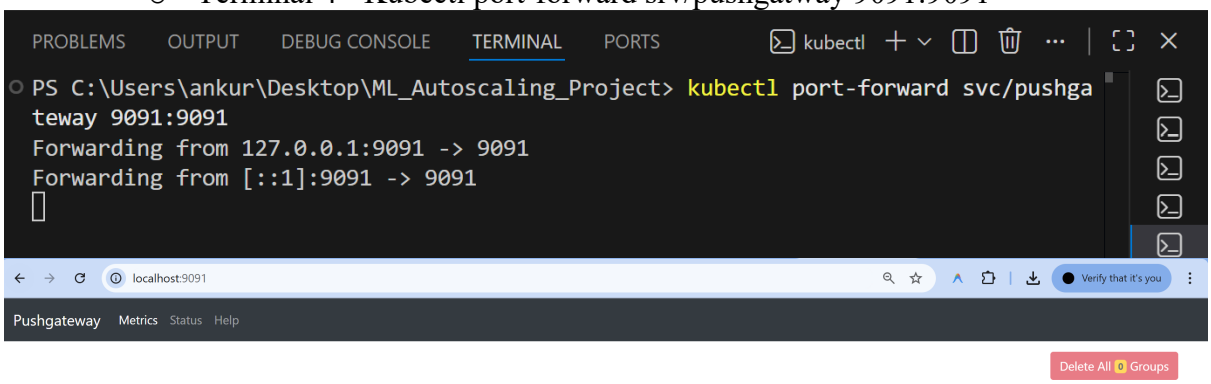
```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl get pods -w
NAME                                                  READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   0          94m
django-app-deployment-7d56c69b48-bh2s8                1/1     Running   0          17m
django-app-deployment-7d56c69b48-bjqg8                1/1     Running   0          17m
django-app-deployment-7d56c69b48-dw5f7                1/1     Running   0          17m
prometheus-grafana-cf65bdf7c-xh4gw                    3/3     Running   0          94m
prometheus-kube-prometheus-operator-6fc45cb74f-cbzbs  1/1     Running   0          94m
prometheus-kube-state-metrics-5d7bdb7c86-mmb6z        1/1     Running   0          94m
prometheus-prometheus-kube-prometheus-prometheus-0    2/2     Running   0          94m
prometheus-prometheus-node-exporter-xchck             1/1     Running   0          94m
pushgateway-78589bf75c-jvrwm                          1/1     Running   0          8m12s
```

- Run application, pushgateway and Grafana in kubernestes
  Run following command in different terminal
  - Termina 3 - Kubectl port-forward service/Django-app-service 8080:80

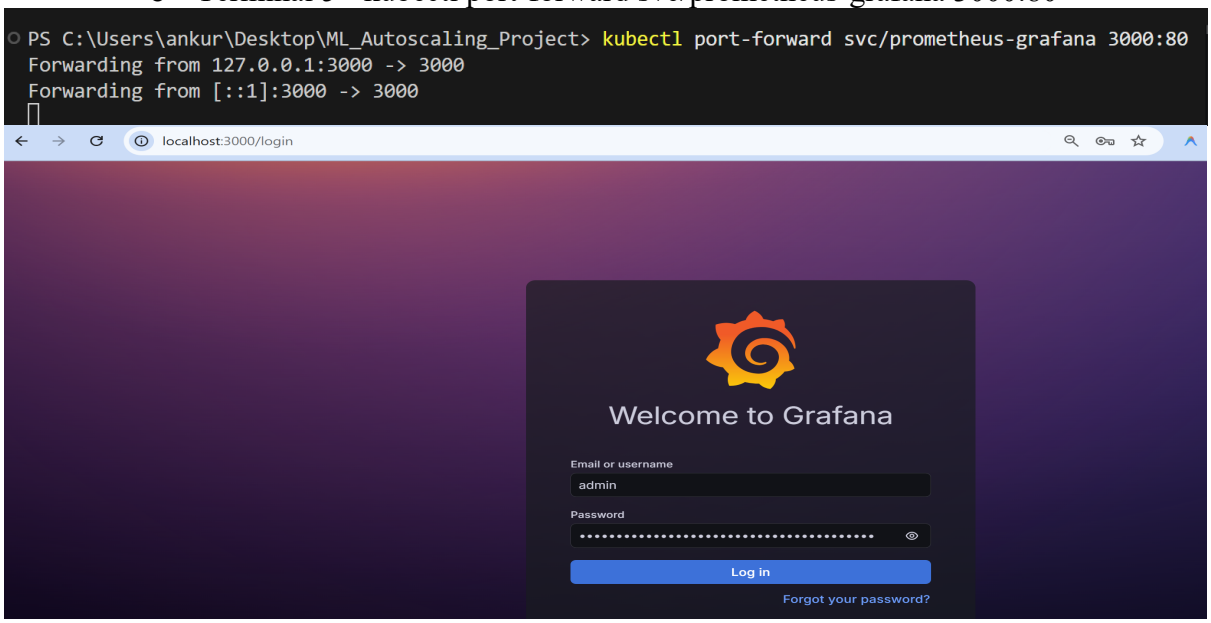


CPU-intensive task complete.

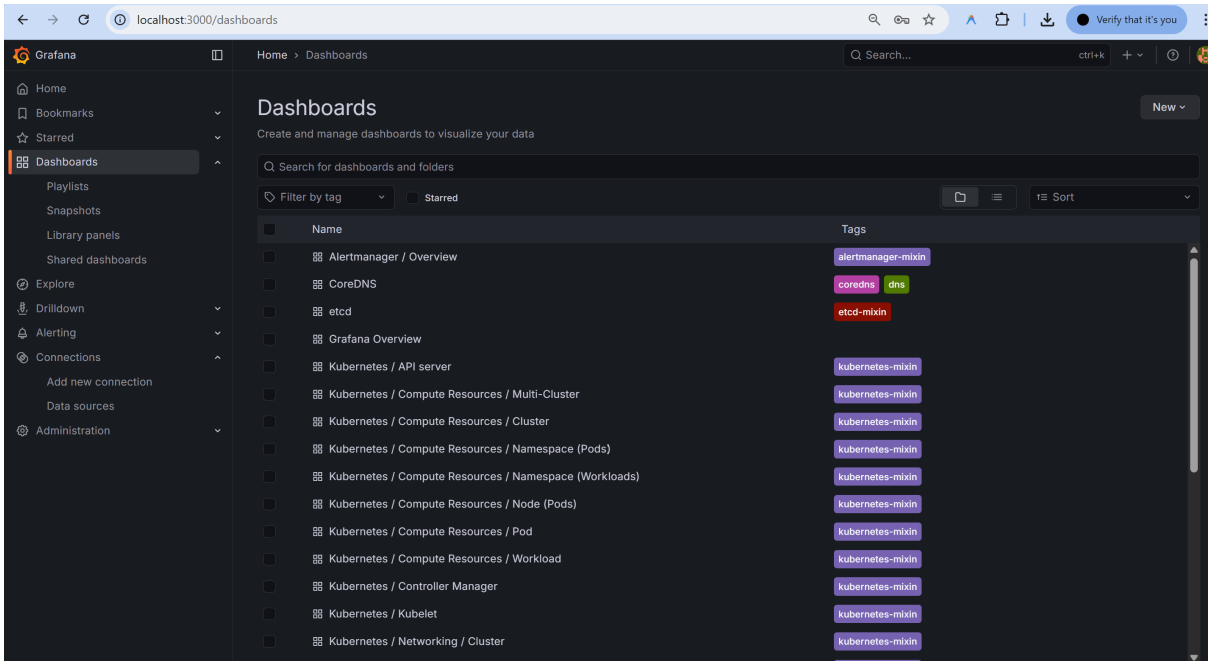
  - Terminal 4 - Kubectl port-forward srv/pushgatway 9091:9091



  - Terminal 5 - kubectl port-forward svc/prometheus-grafana 3000:80



Login using the same password generated using the command



Application, Prometheus pushgateway and Grafana are successfully running in the Kubernetes environment. Now will deploy the different phases to run in this application where the application will have the traffic load using hey and performance will be measured by Prometheus and it will displayed in the Grafana dashboard – showing traffic count, cpu performance, pods scaling up and down based on the different phases configuration deployed.

3 Phase 0 - Baseline Default HPA

3.1 Introduction

This is the baseline of the entire research – the default HPA in Kubernetes environment. The CPU utilization will be observed in this phase and will be recorded for the future sequential phases. This section will provide step by step guidance, configuration and execute for the results to be compared with predicted phases.

3.2 System Prerequisites

Before proceeding for this phase, please ensure the following components are installed and configured properly:

- A running Kubernetes cluster (kind, Docker Desktop, or a cloud-based cluster).
- kubectl command - line tool, configured to communicate with your cluster.
- helm command - line tool, for deploying the monitoring stack.
- The complete project source code, including all YAML files and scripts.
- Do clean the environment specially the Prometheus
  Run the following command in terminal
  - Helm list
  - helm uninstall Prometheus
  - helm uninstall prometheus-stack
  - kubectl delete crd alertmanagers.monitoring.coreos.com `
    podmonitors.monitoring.coreos.com `
    prometheuses.monitoring.coreos.com `

prometheusrules.monitoring.coreos.com `
servicemonitors.monitoring.coreos.com `
thanosrulers.monitoring.coreos.com `
probe.monitoring.coreos.com

3.3 Core Configuration Files for phase 0

This Phase 0 research experiment is defined by a set of the following specific, isolated configuration files:

- django-app-deployment.yaml –
  Defines the web application that will be subjected to the load test and scaled by the HPA – remains the same as of previous section and will be same for all the phases

- load-generator.yaml –
  Defines a utility pod containing the necessary tools (curl, hey) to execute the load test script within the cluster. It will be a separate pod that hits the server for load generation from client side

load-generator.yaml 1 | load-generator-phase1.yaml | predict_phase2_arima.py | monitoring-valu

load-generator.yaml > ...

io.k8s.api.core.v1.Pod (v1@pod.json)

```yaml
# THIS IS A RADICALLY SIMPLIFIED POD DEFINITION FOR PHASE 0 ONLY.
# IT REMOVES ALL PYTHON AND ML DEPENDENCIES TO GUARANTEE A SUCCESSFUL START.
apiVersion: v1
kind: Pod
metadata:
  name: load-generator
spec:
  # Add a volume to hold the traffic data file
  volumes:
  - name: shared-data
    emptyDir: {}
  restartPolicy: "Never"
  containers:
  - name: main
    image: alpine:latest
    command: ["/bin/sh", "-c"]
    args:
      - |
        set -e
        echo "--- Initializing SIMPLIFIED Load Generator for Phase 0 ---"
        # We only need 'curl' to push metrics and 'hey' to generate load.
        # This will build in seconds, not minutes.
        apk add --no-cache curl hey
        echo "--- SETUP COMPLETE. Pod is ready for Phase 0 experiments. ---"
        # Sleep forever, waiting for `kubectl exec` commands.
        sleep infinity
    # Mount the volume so we can copy the CSV file into it.
    volumeMounts:
    - name: shared-data
      mountPath: /data
```

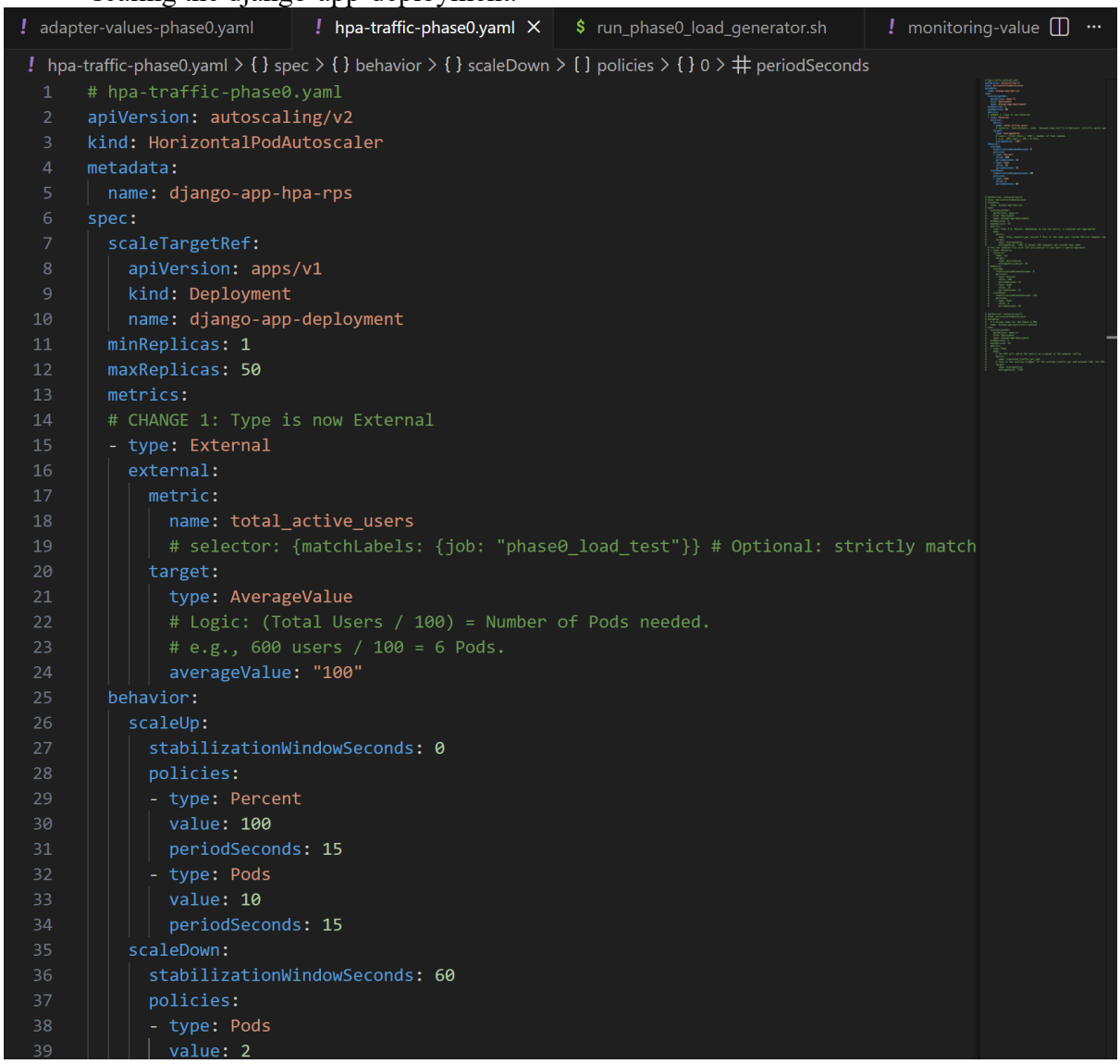
- monitoring-values-phase0.yaml –
  A Helm values file used to configure the kube-prometheus-stack. It is critical for enabling the Prometheus Adapter.

```yaml
# monitoring-values-phase0.yaml
# 1. Configure Pushgateway
pushgateway:
  enabled: true

# 2. Configure Prometheus to scrape Pushgateway
prometheus:
  prometheusSpec:
    serviceMonitorSelectorNilUsesHelmValues: false
    podMonitorSelectorNilUsesHelmValues: false
    additionalScrapeConfigs:
    - job_name: 'pushgateway'
      scrape_interval: 10s
      honor_labels: true
      static_configs:
      - targets: ['pushgateway.default.svc.cluster.local:9091']
# 3. Configure the Adapter (Critical for HPA)
prometheus-adapter:
  enabled: true
  # This tells the adapter where to find Prometheus service created by this chart
  prometheus:
    url: http://prometheus-stack-kube-prom-prometheus.default.svc
    port: 9090

  rules:
    default: false
    custom:
    # This rule looks for the metric "simulated_traffic_users"
    - seriesQuery: '{__name__=~"^simulated_traffic_users$",job="phase0_load_test"}'
      resources:
        template: "<<.Resource>>"
        overrides:
          namespace: {resource: "namespace"}
      name:
        # Renames it to "total_active_users" for the HPA
        matches: "^(.*)_users"
        as: "total_active_users"
      metricsQuery: sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)
```

- adapter-values-phase0.yaml –
  A ConfigMap containing the specific rule for the Prometheus Adapter. This rule translates the simulated_traffic_users metric from Prometheus into a simulated_traffic_per_pod metric that the HPA can understand.

```yaml
# adapter-values-phase0.yaml

prometheus:
  url: http://prometheus-stack-kube-prom-prometheus.default.svc
  port: 9090

rules:
  default: false

  # REMOVE the 'custom' block entirely
  # ADD this 'external' block
  external:
  - seriesQuery: '{__name__=~"^simulated_traffic_users$",job="phase0_load_test"}'
    metricsQuery: sum(<<.Series>>)
    name:
      as: "total_active_users"
    resources:
      overrides:
        # This is boilerplate required for external metrics
        namespace: {resource: "namespace"}
        service: {resource: "service"}
```

- hpa-traffic-phase0.yaml –
  The Horizontal Pod Autoscaler (HPA) resource. It is configured to monitor the simulated_traffic_per_pod metric and maintain an average of 100 users per pod by scaling the django-app-deployment.



- run_phase0_load_generator.sh -
  The shell script that reads a traffic pattern from a CSV file and generates the corresponding user load against the application.

- grafana-dashboard-phase0.json –
  A pre-configured Grafana dashboard definition for visualizing the results of the experiment.

3.4 Step-by-Step Execution Procedure

Execute the following commands in sequence from the root directory of the project in separate terminal.

- Step 1 : once cleaned the environment to start fresh with the phase0
  Run Following command to clean in the terminal –

- Helm list
- helm uninstall Prometheus
- helm uninstall prometheus-stack
- kubectl delete crd alertmanagers.monitoring.coreos.com `
  podmonitors.monitoring.coreos.com `
  prometheuses.monitoring.coreos.com `
  prometheusrules.monitoring.coreos.com `
  servicemonitors.monitoring.coreos.com `
  thanosrulers.monitoring.coreos.com `
  probe.monitoring.coreos.com
- kubectl delete -f load-generator.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm list
NAME                    NAMESPACE       REVISION        UPDATED                                 ST
ATUS    CHART                           APP VERSION
prometheus-adapter      default         2               2025-12-06 20:05:49.8004503 +0000 UTC   de
ployed  prometheus-adapter-5.2.0        v0.12.0
prometheus-stack        default         1               2025-12-06 19:40:55.6365769 +0000 UTC   de
ployed  kube-prometheus-stack-79.12.0   v0.86.2
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm uninstall prometheus
Error: uninstall: Release not loaded: prometheus: release: not found
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm uninstall prometheus-stack
release "prometheus-stack" uninstalled
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl delete crd alertmanagers.monitoring.core
os.com `
>> podmonitors.monitoring.coreos.com `
>> prometheuses.monitoring.coreos.com `
>> prometheusrules.monitoring.coreos.com `
>> servicemonitors.monitoring.coreos.com `
>> thanosrulers.monitoring.coreos.com `
>> probe.monitoring.coreos.com
>>
customresourcedefinition.apiextensions.k8s.io "alertmanagers.monitoring.coreos.com" deleted
customresourcedefinition.apiextensions.k8s.io "podmonitors.monitoring.coreos.com" deleted
customresourcedefinition.apiextensions.k8s.io "prometheuses.monitoring.coreos.com" deleted
customresourcedefinition.apiextensions.k8s.io "prometheusrules.monitoring.coreos.com" deleted
customresourcedefinition.apiextensions.k8s.io "servicemonitors.monitoring.coreos.com" deleted
customresourcedefinition.apiextensions.k8s.io "thanosrulers.monitoring.coreos.com" deleted
Error from server (NotFound): customresourcedefinitions.apiextensions.k8s.io "probe.monitoring.cor
eos.com" not found
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>
```

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl delete -f load-generator.yaml
pod "load-generator" deleted from default namespace
```

- Step 2 : Install and Configure the Monitoring Stack
  The Prometheus monitoring stack, including the crucial Prometheus Adapter, must be deployed first for the execution of phase 0.
  Terminal 1 –
  - helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
  - helm repo update
  - helm install prometheus prometheus-community/kube-prometheus-stack --values monitoring-values-phase0.yaml
  - $passwordBytes = kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}'; [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($passwordBytes)) – To fetch Grafana password

- helm upgrade prometheus-adapter prometheus-community/prometheus-adapter -f adapter-values-phase0.yaml
- kubectl rollout restart deployment prometheus-adapter

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-adapter" chart repository
Update Complete. ⎈Happy Helming!⎈
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm install prometheus prometheus-community/kube-prometheus-stack --values monitoring-values-phase0.yaml
I1207 14:52:11.504044   23944 warnings.go:110] "Warning: unrecognized format \"int64\""
I1207 14:52:26.534309   23944 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
NAME: prometheus
LAST DEPLOYED: Sun Dec  7 14:52:12 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace default get secrets prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace default get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=prometheus" -oname)
  kubectl --namespace default port-forward $POD_NAME 3000

Get your grafana admin user password by running:

  kubectl get secret --namespace default -l app.kubernetes.io/component=admin-secret -o jsonpath="{.items[0].data.admin-password}" | base64 --decode ; echo


Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project>  $passwordBytes = kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}'; [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($passwordBytes))
MLLzEky1eOAmGxQ5HbOAoznedvag01T5dKy7G4ne
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm upgrade prometheus-adapter prometheus-community/prometheus-adapter -f adapter-values-phase0.yaml
Release "prometheus-adapter" has been upgraded. Happy Helming!
NAME: prometheus-adapter
LAST DEPLOYED: Sun Dec  7 14:53:52 2025
NAMESPACE: default
STATUS: deployed
REVISION: 3
TEST SUITE: None
NOTES:
prometheus-adapter has been deployed.
In a few minutes you should be able to list metrics using the following command(s):

  kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1

  kubectl get --raw /apis/external.metrics.k8s.io/v1beta1
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl rollout restart deployment prometheus-adapter
deployment.apps/prometheus-adapter restarted
```

- Step 2 : Deploy Application and HPA resources
  Apply the code Kubernetes objects for the application and the HPA.
  Terminal 1 –
    - kubectl apply -f django-app-deployment.yaml
    - kubectl apply -f hpa-traffic-phase0.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f django-app-deployment.yaml
deployment.apps/django-app-deployment configured
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f hpa-traffic-phase0.yaml
horizontalpodautoscaler.autoscaling/django-app-hpa-rps created
```

- Step 3 – Deploy the load generator pod
  Deploy the utility pod and copying the necessary traffic data and scripts in the load generator pod
  Terminal 1 –
    - Kubectly apply -f load-generator.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f load-generator-phase1.yaml
role.rbac.authorization.k8s.io/deployment-scaler-role created
rolebinding.rbac.authorization.k8s.io/deployment-scaler-binding created
pod/load-generator created
```

  Terminal 2 –
    - Kubectl get pods -w
      (Wait for the load generator pods to enter the running state before copying the scrips and proceeding further)

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   0          22m
django-app-deployment-7d56c69b48-bjqg8              1/1     Running   0          21h
django-app-deployment-7d56c69b48-dw5f7              1/1     Running   0          21h
django-app-deployment-7d56c69b48-fb89r              1/1     Running   0          18h
load-generator                                      1/1     Running   0          39s
prometheus-adapter-67fc66b55-crzqm                  1/1     Running   0          9m35s
prometheus-grafana-6db7b86cf7-z2xw7                 3/3     Running   0          22m
prometheus-kube-prometheus-operator-6fc45cb74f-s2hvq     1/1     Running   0          22m
prometheus-kube-state-metrics-5d7bdb7c86-zdmdr      1/1     Running   0          22m
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0          22m
prometheus-prometheus-node-exporter-22vb8           1/1     Running   0          22m
pushgateway-78589bf75c-jvrwm                        1/1     Running   0          21h
```

  Terminal 1 –
  Once the load generator pod is up and running
    - kubectl cp data/phase0_synthetic_traffic.csv load-generator:/data/phase0_synthetic_traffic.csv
    - kubectl cp run_phase0_load_generator.sh load-generator:/run_phase0_load_generator.sh

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cp data/phase0_synthetic_traffic.csv loa
d-generator:/data/phase0_synthetic_traffic.csv
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cp run_phase0_load_generator.sh load-gen
erator:/run_phase0_load_generator.sh
```

- Step 4 – Make sure server is running for application and Grafana
  Run following command in different terminal
  Terminal 3 –
    - kubectl port-forward service/django-app-service 8080:80

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl port-forward service/django-app-service
8080:80
Forwarding from 127.0.0.1:8080 -> 8000
Forwarding from [::1]:8080 -> 8000
Forwarding from [::1]:8080 -> 8000
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> Kubectl port-forward svc/pushgateway 9091:9091
Forwarding from 127.0.0.1:9091 -> 9091
Forwarding from [::1]:9091 -> 9091
Handling connection for 9091
Handling connection for 9091
```

Terminal 5 -

- kubectl port-forward svc/prometheus-stack-grafana 3000:80

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl port-forward svc/prometheus-stack-grafan
a 3000:80
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

- since we cleaned everything new password is needed
  Run command in terminal 1-
  $secret = kubectl get secret prometheus-stack-grafana -o jsonpath="{.data.admin-password}"
  [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($secret))

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> $secret = kubectl get secret prometheus-stack-gr
afana -o jsonpath="{.data.admin-password}"
>> [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($secret))
N2ZW2A8qVbqcQBc4HQY3qaQrHVSrEHfwdwsfweXL
```

- Step 5 – Execute the load
  Initiating the load test from the load generator pod. This process will run for approximately 60 minutes

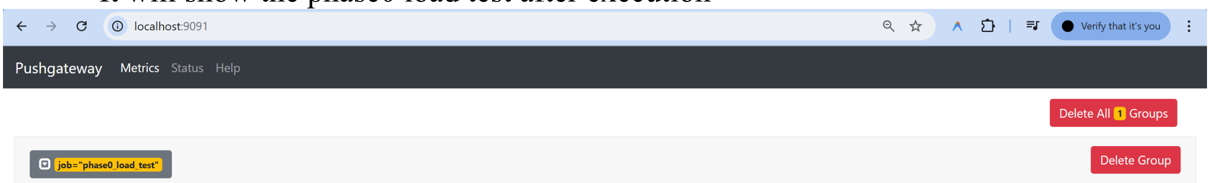
  Terminal 5 –

  - kubectl exec -it load-generator -- /bin/sh /run_phase0_load_generator.sh

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- /bin/sh /run_
phase0_load_generator.sh
---
Initializing test: Cleaning up any stale metrics from previous runs...
Cleanup complete.
---
Starting Phase 0 Load Test (60-minute duration, 1 row per minute)
---
Processing new minute: Sending 922 concurrent users for the next 60s...
```

```
Status code distribution:
  [200] 156543 responses

Error distribution:
  [44]  Get "http://django-app-service.default.svc.cluster.local:80/": dial tcp 10.96.238.2:80:
 connect: connection refused

---
Processing new minute: Sending 949 concurrent users for the next 60s...

Summary:
  Total:        60.1922 secs
  Slowest:      2.6647 secs
  Fastest:      0.0051 secs
  Average:      0.3570 secs
  Requests/sec: 2791.5748

  Total data:   4704868 bytes
  Size/request: 28 bytes

Response time histogram:
  0.005 [1]     |
  0.271 [73454] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.537 [64908] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.803 [22855] |■■■■■■■■■■■■
  1.069 [4538]  |■■
  1.335 [1339]  |■
  1.601 [458]   |
  1.867 [284]   |
  2.133 [157]   |
  2.399 [30]    |
  2.665 [7]     |


Latency distribution:
  10% in 0.1357 secs
  25% in 0.1929 secs
  50% in 0.3018 secs
  75% in 0.4659 secs
  90% in 0.6369 secs
  95% in 0.7666 secs
  99% in 1.1312 secs
```

- Step 6 – Check pushgateway
  It will show the phase0 load test after execution



- Step 7 – Check command the data is pushing into the external for active users
  Run following command in terminal 1 –
  kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/namespaces/default/total_active_users"

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl get --raw "/apis/external.metrics.k8s.io
/v1beta1/namespaces/default/total_active_users"
{"kind":"ExternalMetricValueList","apiVersion":"external.metrics.k8s.io/v1beta1","metadata":{},"it
ems":[{"metricName":"total_active_users","metricLabels":{"job":"phase0_load_test"},"timestamp":"20
25-12-07T18:24:23Z","value":"526"}]}
```

4 Phase 1 – Benchmarking Perfect Foresight

4.1 Introduction

The step by step process of running, implementing the phase 1 with result are explained in this section. Different files and there code understanding will be explained along with the live commands run in different terminal simultaneously for the execution of the phase 1 and record the results from metrics. And at last will monitor the Grafana dashboard to see the results on the perfect forecast knowledge of the traffic.

4.2 System Prerequisites

Before proceeding, ensure the following components are installed and configured:

- A running Kubernetes cluster (example: kind, Docker Desktop, or a cloud-based cluster), currently it was run locally using IDE Visual Studio Code.
- kubectl command-line tool, configured to communicate with your cluster.
- helm command line tool, for deploying the monitoring stack.
- The complete project source code, including all YAML files and scripts.

4.3 Core Configuration files

The Phase 1 experiment is defined by a set of specific, isolated configuration files:

- prepare_data.py –
  A preliminary script that processes the 8-day Alibaba dataset into a per-minute traffic plan (final_load_test_traffic.csv), which serves as the "future" for our simulation.

- live_predictor_hybrid.py –
  This is the core scaler for Phase 1. It acts as the "perfect" predictor by reading final_load_test_traffic.csv to determine future needs. It also includes a reactive safety net by monitoring live traffic from Prometheus, ensuring it always provisions enough replicas to meet the maximum of either the future or present demand.

- run_phase1_load_generator.sh –
  The load generation script. It reads the exact same final_load_test_traffic.csv file and executes the user load against the application, minute by minute.

- django-app-deployment.yaml –
  The standard Kubernetes deployment for the stateless Django web application.
- load-generator.yaml: A universal utility pod definition that includes a full build environment with Python, hey, curl, and all necessary libraries to run the experiments for all phases.

- monitoring-values-standard.yaml –
  A clean Helm values file for the kube-prometheus-stack that enables the Pushgateway but keeps the Prometheus Adapter disabled, as it is not needed for this phase.

- grafana-dashboard-phase1.json –
  A pre-configured Grafana dashboard for visualizing the results of the Phase 1 experiment.

4.4 Step by step execution for phase 1

Execute the following commands in sequence from the root directory on specified terminal

- Step 1 – Clean up the phase 0 configuration

  Run following command in terminal 1 –

  - helm uninstall Prometheus-stack : Remove the monitoring and will apply another for phase 1
  - kubectl delete hpa django-app-hpa-rps --ignore-not-found=true : delete the phase 0 HPA
  - helm uninstall prometheus-adapter : To remove the adaptor from phase 0.
  - kubectl delete configmap prometheus-adapter-rules-phase0 --ignore-not-found=true : Remove adaptor mapping as its not needed for phase 1
  - kubectl delete pod load-generator --ignore-not-found=true : Delete the load generator pod and rerceat and copy the files for phase 1
  - kubectl scale deployment django-app-deployment --replicas=1 : Delete all the remaining pods and scale down to 1 if any extra pods are active

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm uninstall prometheus-stack
release "prometheus-stack" uninstalled
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl delete hpa django-app-hpa-rps --ignore-not-found=true
horizontalpodautoscaler.autoscaling "django-app-hpa-rps" deleted from default namespace
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl delete configmap prometheus-adapter-rules-phase0 --ignore-not-found=true
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl delete pod load-generator --ignore-not-found=true
pod "load-generator" deleted from default namespace
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl scale deployment django-app-deployment --replicas=1
deployment.apps/django-app-deployment scaled
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm uninstall prometheus-adapter
release "prometheus-adapter" uninstalled
```

- Step 1 – Check the remaining active pods for phase 1, there should be 2 active pods

  Run following command in terminal 2 –

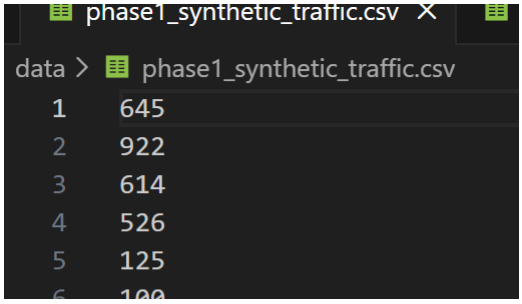
  Terminal 2 –

  Kubectl get pods

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl get pods -w
NAME                                  READY   STATUS    RESTARTS   AGE
django-app-deployment-7d56c69b48-s6c4t   1/1     Running   0          111m
pushgateway-78589bf75c-vx57j          1/1     Running   0          108m
```

- Step 2 – Replicate the phase0_synthetic_data.csv to phase1_synthetic_data.csv for the pods to scale up before the actual hits, this will be the perfect low load for the prior to ML load to test how the scaling react if it has perfect forecast

  phase1_synthetic_data.csv



- Step 3 – Install and configure the monitoring stack
  Deploy the kube-prometheus-stack using the standard configuration that enable the pushgateway.
  Terminal 1 –
  - helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
  - helm repo update
  - helm install prometheus prometheus-community/kube-prometheus-stack --values monitoring-values-standard.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo add prometheus-community https://prome
theus-community.github.io/helm-charts
"prometheus-community" already exists with the same configuration, skipping
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "grafana" chart repository
...Successfully got an update from the "prometheus-adapter" chart repository
Update Complete. ⎈Happy Helming!⎈
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> helm install prometheus prometheus-community/kube-prom
etheus-stack --values monitoring-values-standard.yaml
I1207 19:58:16.927710   48144 warnings.go:110] "Warning: unrecognized format \"int64\""
I1207 19:58:16.927710   48144 warnings.go:110] "Warning: unrecognized format \"int32\""
I1207 19:58:17.056506   48144 warnings.go:110] "Warning: unrecognized format \"int64\""
I1207 19:58:17.056506   48144 warnings.go:110] "Warning: unrecognized format \"int32\""
I1207 19:58:26.364997   48144 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless se
rvices"
NAME: prometheus
LAST DEPLOYED: Sun Dec  7 19:58:17 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace default get secrets prometheus-grafana -o jsonpath="{.data.admin-password}" | base
64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace default get pod -l "app.kubernetes.io/name=grafana,app.kubernete
s.io/instance=prometheus" -oname)
  kubectl --namespace default port-forward $POD_NAME 3000

Get your grafana admin user password by running:

  kubectl get secret --namespace default -l app.kubernetes.io/component=admin-secret -o jsonpath="{.item
s[0].data.admin-password}" | base64 --decode ; echo


Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configu
re Alertmanager and Prometheus instances using the Operator.
```

- Step 4 – Fetch Grafana password as the fresh installation was done –
  Run the following command in terminal –
  Terminal 1 –
  $passwordBytes = kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}';
  [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($passwordBytes))

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> $passwordBytes = kubectl get secret prometheus-grafana
 -o jsonpath='{.data.admin-password}'; [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase
64String($passwordBytes))
s3PmekUPaPlEzWuoQhm0cDObFw1unv7GeSw9Kz5t
```

- Step 3 – Deploy load Generator
  Apply the Kubernetes objects for the Django application and the load generator pod.
  Terminal 1 –
  - kubectl apply -f load-generator-phase1.yaml

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl apply -f .\load-generator-phase1.yaml
role.rbac.authorization.k8s.io/deployment-scaler-role created
rolebinding.rbac.authorization.k8s.io/deployment-scaler-binding created
pod/load-generator created
```

  Terminal 2 –
  - kubectl get pods -w

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl get pods -w
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   0          21h
django-app-deployment-7d56c69b48-s6c4t                   1/1     Running   0          23h
load-generator                                           1/1     Running   0          16s
prometheus-grafana-cc8679dcd-jsffr                       3/3     Running   0          21h
prometheus-kube-prometheus-operator-6fc45cb74f-wnfqj     1/1     Running   0          21h
prometheus-kube-state-metrics-5d7bdb7c86-lw78c           1/1     Running   0          21h
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0          21h
prometheus-prometheus-node-exporter-vwhlz                1/1     Running   0          21h
pushgateway-78589bf75c-vx57j                             1/1     Running   0          23h
```

- Step 4 – Copy phase 1 scripts experiment execution files to load generator pod
  Terminal 1 –
  - kubectl cp phase1_live_predictor_hybrid.py load-generator:\phase1_live_predictor_hybrid.py
  - kubectl cp run_phase1_load_generator.sh load-generator:\run_phase1_load_generator.sh
  - kubectl cp data/phase1_synthetic_traffic.csv load-generator:/data/phase1_synthetic_traffic.csv

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cp phase1_live_predictor_hybrid.py load-genera
tor:\phase1_live_predictor_hybrid.py
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cp run_phase1_load_generator.sh load-generator
:\run_phase1_load_generator.sh
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl cp data/phase1_synthetic_traffic.csv load-gene
rator:/data/phase1_synthetic_traffic.csv
```

  Step 5 – execute the phase 1 experiment from 2 different terminals
  Terminal 3 –
  - kubectl exec -it load-generator -- /bin/sh /run_phase1_load_generator.sh

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- /bin/sh /run_phase1
_load_generator.sh
--- Starting Synthetic Load Test (low Volume) ---

-----------------------------------------------------------
Minute 1: Sending ACTUAL traffic of 645 users...
-----------------------------------------------------------

Summary:
  Total:        60.7724 secs
  Slowest:      2.1001 secs
  Fastest:      0.1101 secs
  Average:      0.8228 secs
  Requests/sec: 779.2850

  Total data:   1326052 bytes
  Size/request: 28 bytes

Response time histogram:
  0.110 [1]     |
  0.309 [210]   |
  0.508 [1660]  |■■
  0.707 [8633]  |■■■■■■■■■■■■
  0.906 [28107] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  1.105 [7032]  |■■■■■■■■■■
  1.304 [1209]  |■■
  1.503 [347]   |
  1.702 [110]   |
  1.901 [25]    |
  2.100 [25]    |
```

Terminal 4 –

- kubectl exec -it load-generator -- python /phase1_live_predictor_hybrid.py

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- python /phase1_live
_predictor_hybrid.py
--- HYBRID SAFETY NET (FINAL): Correct Data Handling ---

[2025-12-07 22:11:26] Cycle 0...
 -> PROACTIVE: Future needs 5 pods.
  -> SUCCESS: Prometheus reports current traffic is 645 users.
 -> REACTIVE:  Present needs 5 pods.
 -> HYBRID DECISION: Taking MAX(5, 5) = 5 pods.
 -> SCALING ACTION: 1 -> 5 pods.

[2025-12-07 22:12:28] Cycle 1...
 -> PROACTIVE: Future needs 4 pods.
  -> SUCCESS: Prometheus reports current traffic is 922 users.
 -> REACTIVE:  Present needs 7 pods.
 -> HYBRID DECISION: Taking MAX(4, 7) = 7 pods.
 -> SCALING ACTION: 5 -> 7 pods.
```

5 Processing Alibaba Data and Evaluation

5.1 Downloading the data using script

For ML prediction we need a dataset for this we have selected the Alibaba cluster 2018 datset which is a 8 day utilization of system, it will be downloaded and processed to prediction and evaluation.

5.2 Core file configuration

- Create Download_dataset.py to download the Alibaba cluster 2018 machine_usage.tar.gz
- Process the Alibaba data create file – process_alibaba_data.py to extract values for machine 1 – m_1
- Convert the cpu usages of Alibaba process data to traffic from minimum range to 200 to maximum of 2500 (assumption based) using python script file process_alibaba_cpu_traffic.py\
- Train all 3 models and save trained model file for further evaluation
  - Prophet Model – Create file train_prophet.py
  - Arima Model – Create file train_arima.py
  - LSTM – Create file train_lstm.py
- Evaluate all 3 models to find out which model performs best with high accuracy and least error.

```
download_dataset.py > ...
1   import os
2   import requests
3   import tarfile
4
5   print("--- Step 1: Data Download ---")
6
7   # --- Configuration BASED ON YOUR FOLDER STRUCTURE ---
8   DATA_DIR = 'data'  # USES YOUR EXISTING 'data' FOLDER
9   DOWNLOAD_URL = 'http://aliopentrace.oss-cn-beijing.aliyuncs.com/v2018Traces/machine_usa
10  TAR_FILE_PATH = os.path.join(DATA_DIR, 'machine_usage.tar.gz')
11  FINAL_CSV_PATH = os.path.join(DATA_DIR, 'machine_usage.csv')
12
13  # Safety check: This will use your existing 'data' folder.
14  if not os.path.exists(DATA_DIR):
15      os.makedirs(DATA_DIR)
16
17  # --- Download the file if it doesn't exist ---
18  if not os.path.exists(FINAL_CSV_PATH):
19      print(f"Downloading dataset from {DOWNLOAD_URL}...")
20      try:
21          with requests.get(DOWNLOAD_URL, stream=True) as r:
22              r.raise_for_status()
23              with open(TAR_FILE_PATH, 'wb') as f:
24                  for chunk in r.iter_content(chunk_size=8192):
25                      f.write(chunk)
26          print("Download complete.")
27
28          # --- Extract the .tar.gz file ---
```

process_alibaba_data.py > ...

```python
# process_alibaba_dataset.py

import pandas as pd
import os

print("--- Step 2: Data Processing (Without Scaling) ---")

# --- Configuration BASED ON YOUR FOLDER STRUCTURE ---
DATA_DIR = 'data'
RAW_CSV_PATH = os.path.join(DATA_DIR, 'machine_usage.csv')
PROCESSED_CSV_PATH = os.path.join(DATA_DIR, 'alibaba_processed_data.csv')

TARGET_MACHINE_ID = 'm_1'
CHUNK_SIZE = 1000000  # Read 1 million rows at a time

# --- Define the column names ONCE, upfront ---
COLUMN_NAMES = ['machine_id', 'timestamp', 'cpu_util_percent', 'mem_util_percent', 'mem_g

# --- Check if raw data exists ---
if not os.path.exists(RAW_CSV_PATH):
    print(f"FATAL: Raw data file not found at '{RAW_CSV_PATH}'.")
    print("Please run 'download_data.py' first.")
    exit()

# --- Process the large file in chunks to save memory ---
print(f"Loading raw data from {RAW_CSV_PATH} in chunks to conserve memory...")
chunk_list = []

# Create an iterator, providing the column names directly to the reader.
csv_iterator = pd.read_csv(
    RAW_CSV_PATH,
    header=None,
    chunksize=CHUNK_SIZE,
    names=COLUMN_NAMES
)

for chunk in csv_iterator:
    # Filter the chunk to find rows for our target machine
    filtered_chunk = chunk[chunk['machine_id'] == TARGET_MACHINE_ID]
```

process_alibaba_cpu_traffic.py > ...

```python
# process_alibaba_cpu_traffic.py

import pandas as pd
import os
import numpy as np

print("--- Step 2: Data Processing (CPU to Traffic Conversion) ---")

# --- Configuration ---
DATA_DIR = 'data'
RAW_CSV_PATH = os.path.join(DATA_DIR, 'machine_usage.csv')
PROCESSED_CSV_PATH = os.path.join(DATA_DIR, 'alibaba_traffic_data.csv')

TARGET_MACHINE_ID = 'm_1'
CHUNK_SIZE = 1000000
COLUMN_NAMES = ['machine_id', 'timestamp', 'cpu_util_percent', 'mem_util_percent', 'mem_g

# --- Simulation Parameters ---
MIN_TRAFFIC = 200
MAX_TRAFFIC = 2500

if not os.path.exists(RAW_CSV_PATH):
    print(f"FATAL: Raw data file not found at '{RAW_CSV_PATH}'.")
    exit()

print(f"Loading raw data from {RAW_CSV_PATH} in chunks...")
chunk_list = []

csv_iterator = pd.read_csv(
    RAW_CSV_PATH,
    header=None,
    chunksize=CHUNK_SIZE,
    names=COLUMN_NAMES
)

for chunk in csv_iterator:
    filtered_chunk = chunk[chunk['machine_id'] == TARGET_MACHINE_ID]
    chunk_list.append(filtered_chunk)
```

train_prophet.py

```python
# train_prophet.py

import pandas as pd
from prophet import Prophet
from joblib import dump
import os
import sys

print("--- TRAINING PROPHET (Raw Data / 80-20 Split) ---")
DATA_PATH = os.path.join('data', 'alibaba_traffic_data.csv')
MODEL_PATH = os.path.join('models', 'phase2_prophet_model.joblib')

if not os.path.exists(DATA_PATH):
    print("FATAL: Data file not found.")
    exit()

# 1. Load Data
print("Loading data...")
df = pd.read_csv(DATA_PATH)
df['ds'] = pd.to_datetime(df['ds'])

# 2. Split Data (80% Train, 20% Test)
# We must split strictly by time (index) to match LSTM/ARIMA
split_point = int(len(df) * 0.8)
train_df = df.iloc[:split_point]

print(f"Total Rows: {len(df)}")
print(f"Training on first: {len(train_df)} rows")

# 3. Train
# specific settings to make it faster on raw data
print("Training Prophet (this might take 1-2 minutes)...")
model = Prophet(
    daily_seasonality=True,
    weekly_seasonality=False, # Data is only ~8 days, weekly is weak
    yearly_seasonality=False,
    uncertainty_samples=0     # Disabling error bars speeds up training 10x
)
```

train_arima.py > ...

```python
# train_arima.py

import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from joblib import dump
import os
import time

print("--- TRAINING ARIMA (Raw Data / 80-20 Split) ---")
DATA_PATH = os.path.join('data', 'alibaba_traffic_data.csv')
MODEL_PATH = os.path.join('models', 'phase2_arima_model.joblib')

# 1. Load Raw Data
print("Loading data...")
df = pd.read_csv(DATA_PATH)
series = df['y'] # Raw 10-second intervals

# 2. Split Data (Crucial for valid evaluation)
# We train on first 80%, so we can test on the remaining 20% later
split_point = int(len(series) * 0.8)
train_series = series.iloc[:split_point]

print(f"Total Rows: {len(series)}")
print(f"Training on first: {len(train_series)} rows (80%)")

# 3. Train
print("Fitting ARIMA (5,1,0)...")
start_time = time.time()
model = ARIMA(train_series, order=(5,1,0))
model_fit = model.fit()
end_time = time.time()

print(f"Training finished in {(end_time - start_time):.2f} seconds.")

# 4. Save
dump(model_fit, MODEL_PATH)
print("SUCCESS: ARIMA Model saved.")
```

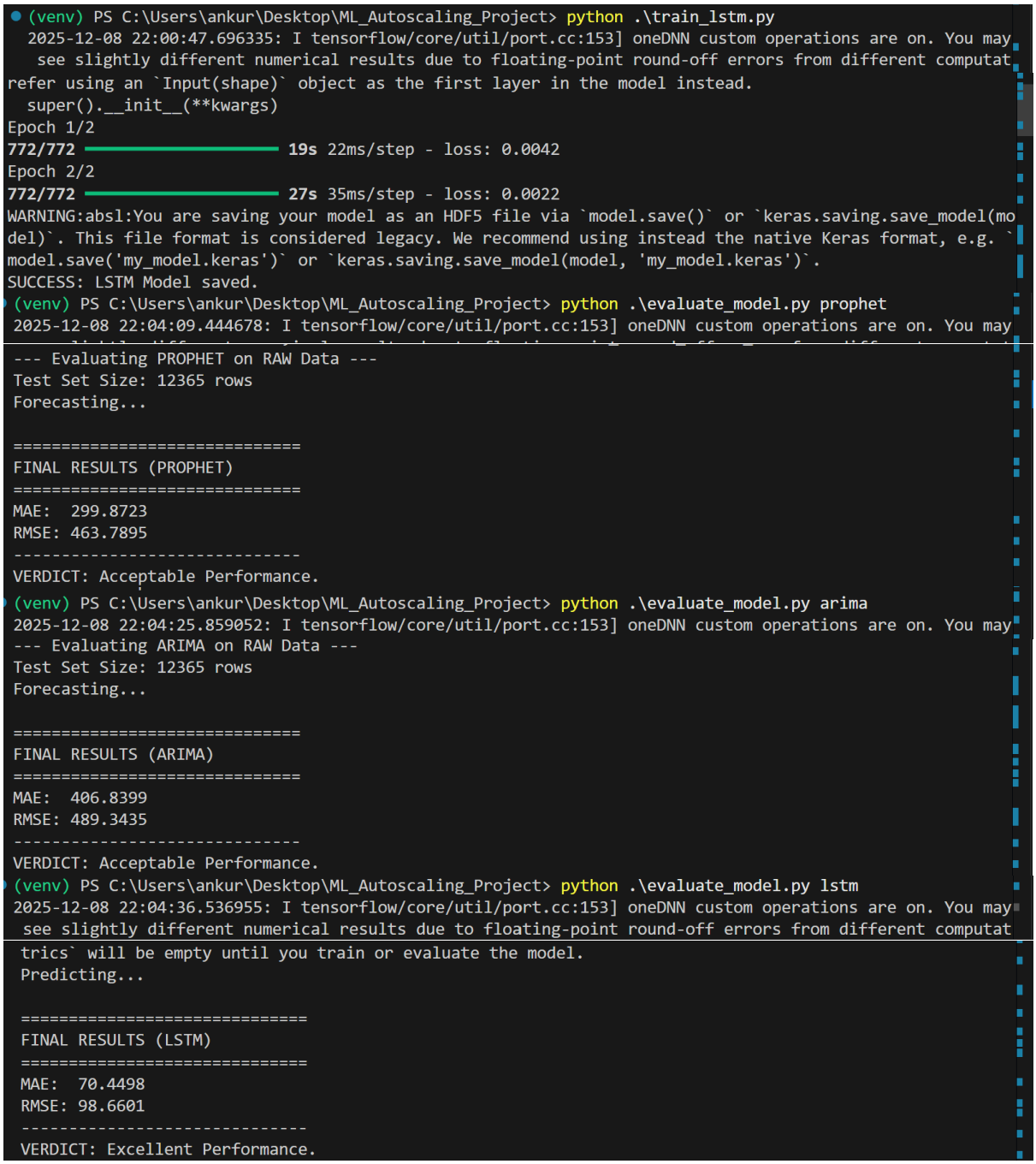
train_lstm.py

```python
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from joblib import dump
import os

print("--- TRAINING LSTM (Raw Data / 80-20 Split) ---")
DATA_PATH = os.path.join('data', 'alibaba_traffic_data.csv')
MODEL_FILE = os.path.join('models', 'phase2_lstm_model.h5')
SCALER_FILE = os.path.join('models', 'phase2_lstm_scaler.joblib')

# 1. Load
df = pd.read_csv(DATA_PATH)
data = df['y'].values.reshape(-1, 1)

# 2. Scale
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# 3. Split 80/20
split_point = int(len(scaled_data) * 0.8)
train_data = scaled_data[:split_point]

# 4. Create Sequences
X_train, y_train = [], []
n_steps = 60 # Look back 60 * 10s = 10 minutes

for i in range(n_steps, len(train_data)):
    X_train.append(train_data[i-n_steps:i, 0])
    y_train.append(train_data[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# 5. Build & Train
print(f"Training LSTM on {len(X_train)} samples...")
model = Sequential([
```

5.3 Execute python script to train and evaluate the model performance

- Make sure to activate the virtual environment to run python scripts in fresh terminal –
- Run following command in terminal –
    - Python train_prophet.py
    - Python train_arima.py
    - Python train_lstm.py
    - Python evaluate_model.py prophet
    - Python evaluate_model.py arima
    - Python evaluate_model.py lstm

```
(venv) PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> python .\train_prophet.py
--- TRAINING PROPHET (Raw Data / 80-20 Split) ---
Loading data...
Total Rows: 61824
Training on first: 49459 rows
Training Prophet (this might take 1-2 minutes)...
21:59:52 - cmdstanpy - INFO - Chain [1] start processing
22:00:12 - cmdstanpy - INFO - Chain [1] done processing
Saving model to models\phase2_prophet_model.joblib...
SUCCESS: Prophet Model saved.
(venv) PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> python .\train_arima.py
--- TRAINING ARIMA (Raw Data / 80-20 Split) ---
Loading data...
Total Rows: 61824
Training on first: 49459 rows (80%)
Fitting ARIMA (5,1,0)...
Training finished in 1.28 seconds.
SUCCESS: ARIMA Model saved.
```



6 Phase 2 – LSTM Model

6.1 Core file configuration

Create following files –

- Alibaba_traffic_data.csv : Traffic load sequence that hey will use to push in Django app.
- Load-generator-phase2.yaml – for creating load-generator pod that will have all the necessary file and will execute
- Phase2_brain_scaler.py – file that will be responsible for scaling the pods
- Phase2_traffic_simulator.py – File responsible to run the traffic load from load-generator pod sending request to http to Django app.

- Phase2_latency_check.py – This file will tell the real time latency of the app response.
- Phase2_lstm_model.h5 and phase2_lstm_scaler.joblib – these 2 files are generated after the LSTM model is trained

load-generator-phase2.yaml > { } metadata > abc name

io.k8s.api.core.v1.Pod (v1@pod.json) | io.k8s.api.rbac.v1.RoleBinding (v1@rolebinding.json) | io.k8s.api.rbac.v1.Role (v1@role.json)

```yaml
# load-generator-phase2.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-scaler-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments", "deployments/scale"]
  verbs: ["get", "list", "watch", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: deployment-scaler-binding
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
roleRef:
  kind: Role
  name: deployment-scaler-role
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Pod
metadata:
  name: load-generator
spec:
  serviceAccountName: default
  volumes:
  - name: shared-data
```

phase2_brain_scaler.py > ...

```python
# phase2_brain_scaler.py

import pandas as pd
import time
import requests
import subprocess
import numpy as np
from joblib import load
from tensorflow.keras.models import load_model

# --- CONFIGURATION ---
MODEL_FILE = "/data/phase2_lstm_model.h5"
SCALER_FILE = "/data/phase2_lstm_scaler.joblib"
# The Brain needs the full dataset for initialization
DATA_FILE = "/data/alibaba_traffic_data.csv"
INPUT_METRICS_URL = "http://pushgateway:9091/metrics"
OUTPUT_METRICS_URL = "http://pushgateway:9091/metrics/job/phase2_brain"
DEPLOYMENT_NAME = "django-app-deployment"

# CAPACITY
REQUESTS_PER_POD = 50

# TIME SETTINGS
HISTORY_STEPS = 60
LOOK_AHEAD_STEPS = 6
COOLDOWN_STEPS = 6
CONFIDENCE_BUFFER = 1.20

print("--- [BRAIN] Starting PROACTIVE AI Scaler with HISTORICAL SEEDING ---")

try:
    model = load_model(MODEL_FILE)
    package = load(SCALER_FILE)
```

phase2_traffic_simulator.py > ...

```python
# phase2_traffic_simulator.py

import pandas as pd
import time
import requests
import sys
import subprocess

# --- CONFIG ---
DATA_FILE = "/data/alibaba_traffic_data.csv"
PUSHGATEWAY_URL = "http://pushgateway:9091/metrics/job/phase2_traffic"
# USE INTERNAL DNS
DJANGO_URL = "http://django-app-service:80"

print("--- [REAL TRAFFIC GENERATOR] Starting (Using HEY) ---")

try:
    df = pd.read_csv(DATA_FILE)
    split_point = int(len(df) * 0.8)
    test_data = df.iloc[split_point:].reset_index(drop=True)
except Exception as e:
    print(f"FATAL: {e}")
    sys.exit(1)

for index, row in test_data.iterrows():
    target_users = int(row['y'])
    timestamp = row['ds']

    # 1. TELL THE BRAIN (Push Metric)
    metric_data = f"""
# TYPE current_simulated_traffic gauge
current_simulated_traffic {target_users}
"""
```

6.2 Execute the load-generator pod and copy files

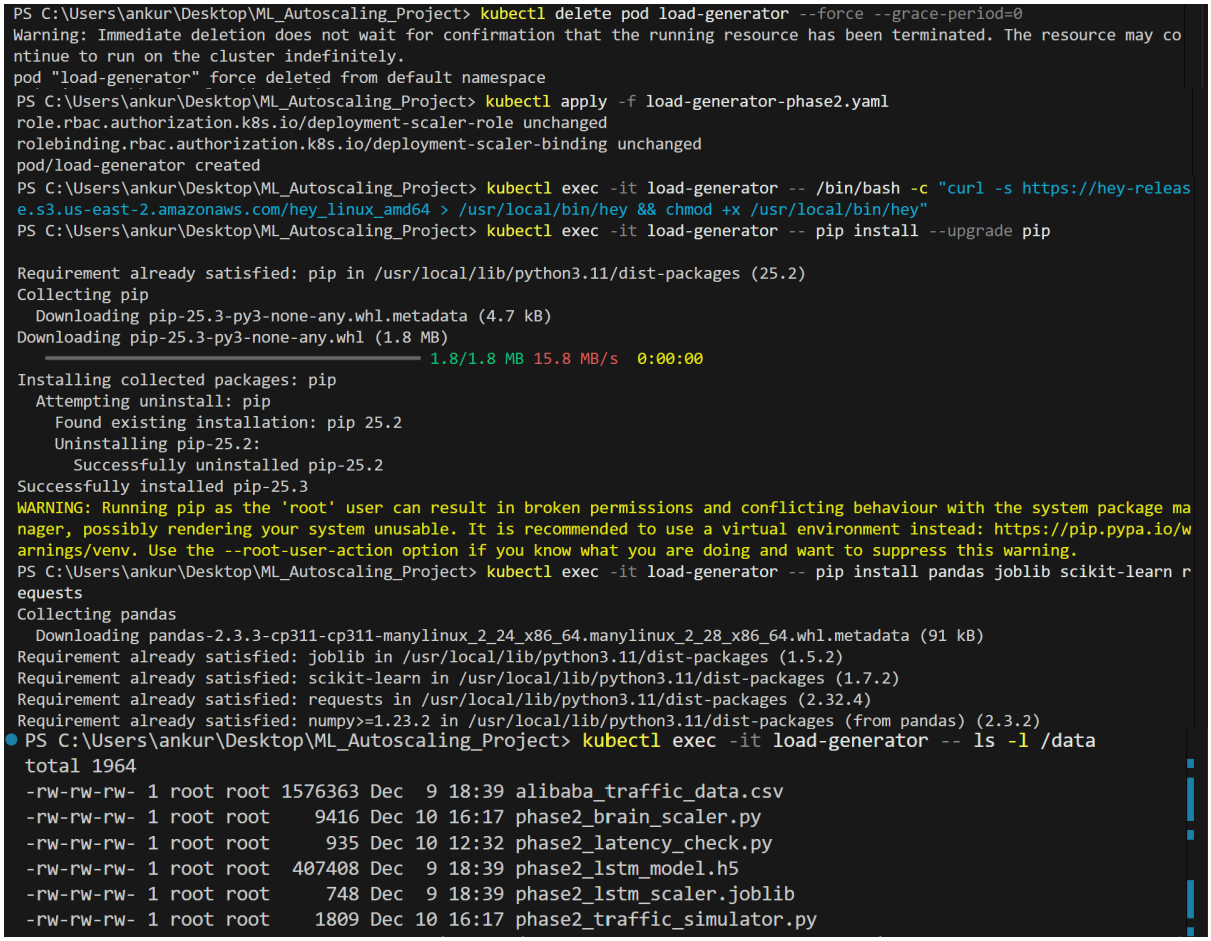
- Apply the load-generator pod in terminal
  Terminal 1 –
  Run the following command –
  - Kubectl delete pod load-generator –force –grace-period=0 : delete the old pod
  - Kubectl apply -f load-generator-phase2.yaml : apply the phase 2 pod

  Wait for 30 second and check if the pod is active and running
  - Kubectl get pods

- Once the load generator is active, upgrade pip , install hey, install libraries
  Run the following command in the terminal –
  - kubectl exec -it load-generator -- /bin/bash -c "curl -s https://hey-release.s3.us-east-2.amazonaws.com/hey_linux_amd64 > /usr/local/bin/hey && chmod +x /usr/local/bin/hey"
  - kubectl exec -it load-generator -- pip install --upgrade pip
  - kubectl exec -it load-generator -- pip install pandas joblib scikit-learn requests

- Once complete copy the file to execute in load-generator pod-
  Run following command in terminal –
  - kubectl cp data/alibaba_traffic_data.csv load-generator:/data/alibaba_traffic_data.csv
  - kubectl cp models/phase2_lstm_model.h5 load-generator:/data/phase2_lstm_model.h5

- kubectl cp models/phase2_lstm_scaler.joblib load-generator:/data/phase2_lstm_scaler.joblib
- kubectl cp phase2_traffic_simulator.py load-generator:/data/phase2_traffic_simulator.py
- kubectl cp phase2_brain_scaler.py load-generator:/data/phase2_brain_scaler.py
- kubectl exec -it load-generator – ls -l /data : to check all the files exists in the load-generator pod before execution



6.3 Run the application and execute

- Run Django service, pushgateway and Grafana
  Run the following command –
  Terminal 2 –
  kubectl port-forward service/django-app-service 8080:80
  Terminal 3 –
  Kubectl port-forward svc/pushgateway 9091:9091
  Terminal 4 –
  kubectl port-forward svc/prometheus-grafana 3000:80

- Once running then start with execution of phase2
  Run the following command –
  - Terminal 5 –

  kubectl exec -it load-generator -- python /data/phase2_traffic_simulator.py

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- python /data/phase2_traffic_simulator.py

--- [REAL TRAFFIC GENERATOR] Starting (Using HEY) ---
[2018-04-09 13:17:30] Traffic: 1258 | Running 'hey' for 10s...
[2018-04-09 13:17:40] Traffic: 1258 | Running 'hey' for 10s...
[2018-04-09 13:17:50] Traffic: 1281 | Running 'hey' for 10s...
[2018-04-09 13:18:00] Traffic: 1235 | Running 'hey' for 10s...
[2018-04-09 13:18:10] Traffic: 1166 | Running 'hey' for 10s...
[2018-04-09 13:18:20] Traffic: 1166 | Running 'hey' for 10s...
[2018-04-09 13:18:30] Traffic: 1166 | Running 'hey' for 10s...
```

- Terminal 6 –

kubectl exec -it load-generator -- python /data/phase2_brain_scaler.py

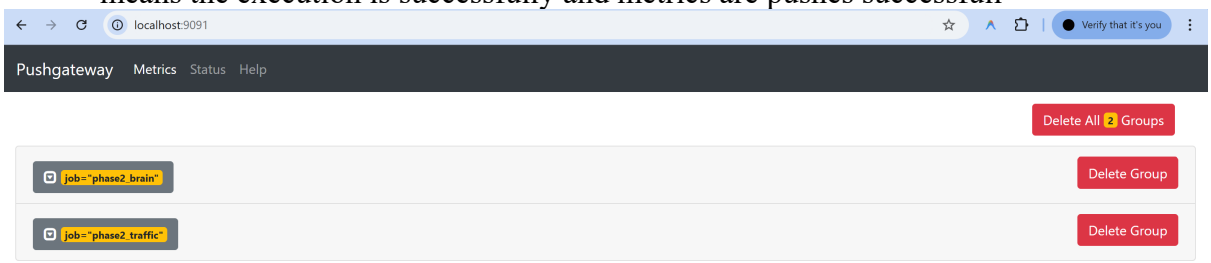
```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- python /data/phase2_brain_scaler.py
2025-12-11 10:40:03.021579: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly differe
nt numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the envi
ronment variable `TF_ENABLE_ONEDNN_OPTS=0`.
```

- Terminal 7 –

kubectl exec -it load-generator -- python /data/phase2_latency_check.py

```
PS C:\Users\ankur\Desktop\ML_Autoscaling_Project> kubectl exec -it load-generator -- python /data/phase2_latency_check.py
--- LIVE LATENCY CHECKER ---
Pinging http://django-app-service:80 every 2 seconds...
SUCCESS - Latency: 299ms
SUCCESS - Latency: 5ms
SUCCESS - Latency: 301ms
SUCCESS - Latency: 100ms
SUCCESS - Latency: 101ms
SUCCESS - Latency: 211ms
```

- once executed check pushgateway server – you will see following 2 groups running means the execution is successfully and metrics are pushes successfull



- Check Grafana dashboard

References

1. Prometheus Authors, 2025. *Prometheus: From metrics to insight, Available at: https://prometheus.io/docs/*
2. Kubernetes, 2025. *Horizontal Pod Autoscaler, https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/*
3. Google Cloud, 2025. *Best practices for running cost-optimized Kubernetes applications, Available at: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture*
4. Grafana Labs, 2025. *Grafana Documentation, Available at: https://grafana.com/docs/*
5. Django Software Foundation, 2025. *Django Documentation, Available at: https://docs.djangoproject.com/*
6. Alibaba Group, 2018. *Alibaba Cluster Trace Program, Available at: https://github.com/alibaba/clusterdata*