National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Supriya Sunil Naik
Student ID: 23302356

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Supriya Sunil Naik
Student ID:	23302356
Programme:	Cloud Computing **Year:** 2024/25
Module:	MSc Research Project
Lecturer:	Ahmed Makki
Submission Due Date:	11/08/2025
Project Title:	An approach to optimize Kubernetes resource management through its adaptive scheduling system combined with threshold-based predictive scaling
Word Count:	…615 **Page Count:** ……8

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: 11/08/2025

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

Supriya Sunil Naik
Student ID: 23302356

1 Introduction

This configuration document is step-by-step tutorial needed to set up and deploy the research project of resource management optimization in Kubernetes clusters. The project involves deploying an adaptive scheduler and integrating threshold-based predictive autoscaling models with Alibaba Cluster Trace data.

2 Setup Environment

Open PowerShell as a Admin:
Run below command:
minikube start --driver=docker --cpus=4 --memory=8192

Install Kubernetes and check if Kubernetes is running or not:
kubectl get nodes

Run below command:
kubectl version –client

Install Helm using Chocolatey:

```
PS C:\Users\hrish\AlibabaClusterTrace> Set-ExecutionPolicy Bypass -Scope Process -Force; `
>> [System.Net.ServicePointManager]::SecurityProtocol = `
>> [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
>> iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
>>
WARNING: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.
WARNING: An existing Chocolatey installation was detected. Installation will not continue. This script will not
overwrite existing installations.
If there is no Chocolatey installation at 'C:\ProgramData\chocolatey', delete the folder and attempt the installation
again.

Please use choco upgrade chocolatey to handle upgrades of Chocolatey itself.
If the existing installation is not functional or a prior installation did not complete, follow these steps:
 - Backup the files at the path listed above so you can restore your previous installation if needed.
 - Remove the existing installation manually.
 - Rerun this installation script.
 - Reinstall any packages previously installed, if needed (refer to the lib folder in the backup).

Once installation is completed, the backup folder is no longer needed and can be deleted.
```

choco install kubernetes-helm

Check Installation version:
helm version

For Monitoring install Prometheus + Grafana and add Helm chart:

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

Installing into Kubernetes:

```
PS C:\Users\hrish\AlibabaClusterTrace> helm install prometheus prometheus-community/kube-prometheus-stack
>>
NAME: prometheus
LAST DEPLOYED: Sat Aug  9 21:46:41 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

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

Installing Metrics Server:

```
PS C:\Users\hrish\AlibabaClusterTrace> kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
>>
serviceaccount/metrics-server unchanged
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader unchanged
clusterrole.rbac.authorization.k8s.io/system:metrics-server unchanged
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader unchanged
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator unchanged
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server unchanged
service/metrics-server unchanged
deployment.apps/metrics-server configured
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io unchanged
PS C:\Users\hrish\AlibabaClusterTrace> kubectl top nodes
>>
NAME       CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
minikube   1721m        43%    3254Mi          40%
```

After 1 minutes, run to check:
kubectl top nodes

3 Load the Dataset

Install Python and pip

Install Jupyter Notebook and libraries
pip install jupyter pandas numpy matplotlib seaborn statsmodels pmdarima scikit-learn

Place Alibaba dataset in folder

Run the below code to create preprocessed csv file:

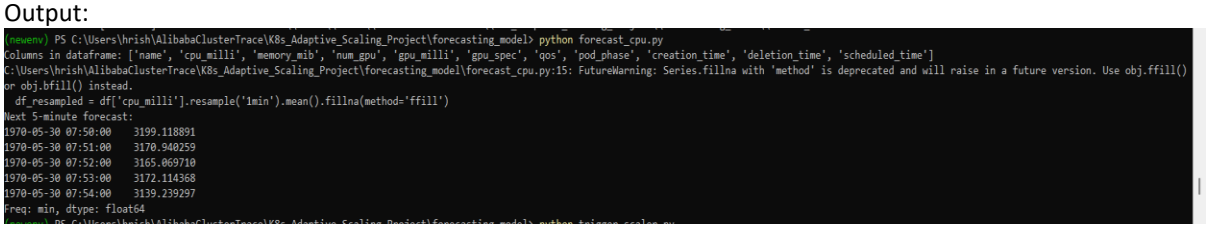
```
import pandas as pd

df = pd.read_csv("../dataset/openb_pod_list_cpu037.csv")
print("Data loaded")

df['timestamp'] = pd.to_datetime(df['timestamp'])
df.set_index('timestamp', inplace=True)

df = df.resample("1min").mean().interpolate()

print(df.head())

df.to_csv("../dataset/preprocessed_data.csv")
print(" Preprocessing complete. Data saved as preprocessed_data.csv")
```

4 Forecasting Model

Step 1:
Create a python code for cpu usage in next 5 minutes and run forecast_cpu.py :

```
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing

df = pd.read_csv(
    r"C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\dataset\preprocessed_data.csv",
    index_col='timestamp', parse_dates=True
)

print("Columns in dataframe:", df.columns.tolist())

#  timestamp index sorting
df = df.sort_index()


df_resampled = df['cpu_milli'].resample('1min').mean().fillna(method='ffill')


model = ExponentialSmoothing(df_resampled, trend="add", seasonal="add", seasonal_periods=60)
fit = model.fit()

forecast = fit.forecast(5)
print("Next 5-minute forecast:")
print(forecast)


forecast.to_csv(r"C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\forecasting_model\latest_forecast.csv")
```

Output:



Step 2:
Create trigger_scaler.py for threshold CPU usage 75% and run it:

```
import pandas as pd
import subprocess


df = pd.read_csv("latest_forecast.csv", index_col=0)

# Threshold for CPU usage  75%
THRESHOLD = 0.75


avg_cpu = df.iloc[:, 0].mean()

print(f"Forecasted average CPU: {avg_cpu:.2f}")


if avg_cpu > THRESHOLD:
    print(" High forecast detected. Scaling up deployment...")
    subprocess.run(["kubectl", "scale", "deployment", "hello-minikube", "--replicas=5"])
else:
    print(" CPU within limits. No scaling action taken.")
```

```
error: no objects passed to scale
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\forecasting_model> kubectl get deployments
>>
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
hello-minikube                    1/1     1            1           10d
prometheus-grafana                1/1     1            1           3d18h
prometheus-kube-prometheus-operator   1/1     1            1           3d18h
prometheus-kube-state-metrics     1/1     1            1           3d18h
prometheus2-grafana               1/1     1            1           20h
prometheus2-kube-prometheu-operator   1/1     1            1           20h
prometheus2-kube-state-metrics    1/1     1            1           20h
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\forecasting_model> python trigger_scaler.py
Forecasted average CPU: 3169.30
⚠ High forecast detected. Scaling up deployment...
deployment.apps/hello-minikube scaled
```

Step3:
Install Pip Kubernetes:
Pip install Kubernetes

Create scheduler.py to assign incoming node with predicted load to implement adaptive scheduler and run it:

```python
def score_nodes(pod_name, forecast):

    config.load_kube_config()

    v1 = client.CoreV1Api()

    nodes = v1.list_node().items

    scores = {}

    for node in nodes:
        node_name = node.metadata.name

        alloc_cpu = node.status.allocatable.get('cpu')
        if alloc_cpu is None:
            alloc_cpu = 0
        else:
            alloc_cpu = int(alloc_cpu)


        used_cpu = 1

        #  forecast CPU load for this node (simulate as 0.5)
        future_util = forecast.get(node_name, 0.5)

        # Score = free CPU minus forecasted usage
        score = alloc_cpu - used_cpu - future_util
        scores[node_name] = score

    # Pick node with max score (most available capacity)
    best_node = max(scores, key=scores.get)
    print(f"Best node to schedule pod {pod_name}: {best_node}")
    return best_node

if __name__ == "__main__":

    test_forecast = {
        "minikube": 0.3,
    }
    score_nodes("test-pod", test_forecast)
```

```
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\custom_scheduler> python scheduler.py
Best node to schedule pod example-pod: minikube
```

Step4:
Run below two command in powershell:
kubectl get svc prometheus-operated -n default

```
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\custom_scheduler> kubectl get svc prometheus-operated -n default
>>
NAME                  TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
prometheus-operated   ClusterIP   None         <none>        9090/TCP   3d18h
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\custom_scheduler> kubectl port-forward svc/prometheus-operated 9090:9090 -n default
>>
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

Run above snapshot command in another powershell and keep it open to get live cluster metrics from Prometheus.:

Create and run prometheus_query.py:

```python
from prometheus_api_client import PrometheusConnect
import datetime

def get_node_cpu_usage(prom_url='http://localhost:9090'):
    prom = PrometheusConnect(url=prom_url, disable_ssl=True)

    query = 'avg(rate(node_cpu_seconds_total{mode!="idle"}[5m])) by (instance)'

    end_time = datetime.datetime.now()
    start_time = end_time - datetime.timedelta(minutes=5)

    result = prom.custom_query_range(
        query=query,
        start_time=start_time,
        end_time=end_time,
        step='60s',
    )

    cpu_usage = {}
    for metric_data in result:
        instance = metric_data['metric']['instance']
        values = metric_data['values']
        avg_val = sum(float(v[1]) for v in values) / len(values)
        cpu_usage[instance] = avg_val

    return cpu_usage

if __name__ == "__main__":
    usage = get_node_cpu_usage()
    for node, val in usage.items():
        print(f"Node: {node} CPU usage: {val:.2f} cores")
```

```
newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\custom_scheduler> python prometheus_query.py
ode: 192.168.49.2:9100 CPU usage: 0.01 cores
```

Step5:
Create and run node_info.py that retrieves CPU, memory from Kubernetes API:

```python
from kubernetes import client, config

def get_nodes_cpu():
    config.load_kube_config()  # loads ~/.kube/config for cluster connection
    v1 = client.CoreV1Api()
    nodes = v1.list_node().items

    node_cpu_info = {}
    for node in nodes:
        name = node.metadata.name
        cpu_alloc = node.status.allocatable['cpu']
        # CPU allocatable is string like '4' or '4000m' (m=millicores)
        # convert to float cores:
        if cpu_alloc.endswith('m'):
            cpu_alloc = float(cpu_alloc[:-1]) / 1000
        else:
            cpu_alloc = float(cpu_alloc)
        node_cpu_info[name] = cpu_alloc
    return node_cpu_info

if __name__ == "__main__":
    cpu_info = get_nodes_cpu()
    for node, cpu in cpu_info.items():
        print(f"{node}: {cpu} CPU cores allocatable")
```

```
(newenv) PS C:\Users\hrish\AlibabaClusterTrace\K8s_Adaptive_Scaling_Project\custom_scheduler> python node_info.py
minikube: 4.0 CPU cores allocatable
```

Step6:
Create and run app.yaml:

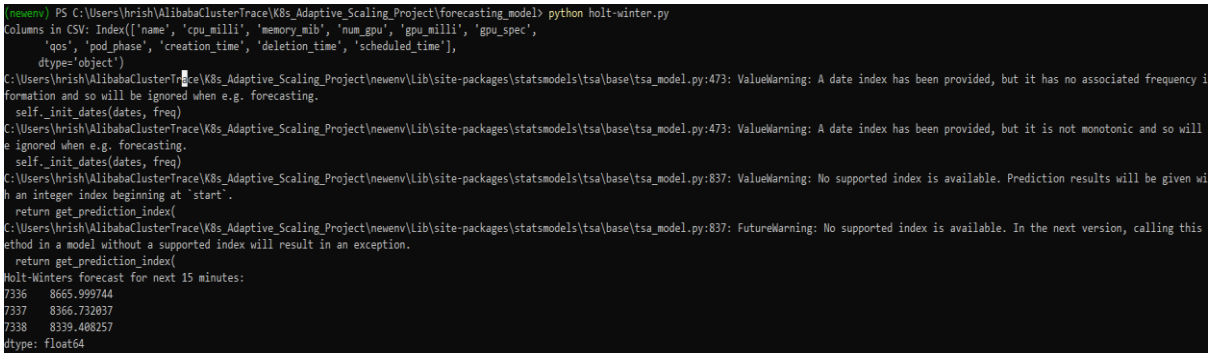
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-api
  template:
    metadata:
      labels:
        app: test-api
    spec:
      containers:
      - name: test-api
        image: nginxdemos/hello
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: "100m"
          limits:
            cpu: "500m"
```

Step7 :
Install and Load test using K6:
choco install k6

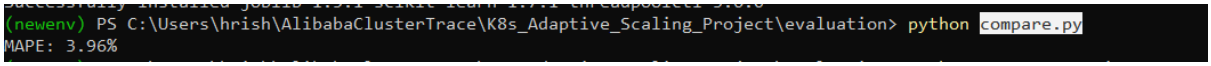
Create and run load_test:
k6 run load_test.js

Step8:
Create and run Holt_Winter.py to plot:

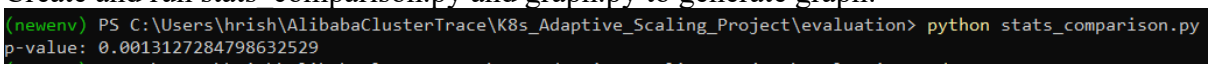
```
import matplotlib.pyplot as plt


df = pd.read_csv("../dataset/preprocessed_data.csv", parse_dates=['timestamp'])
df = df.sort_values('timestamp')
df.set_index('timestamp', inplace=True)

print("Columns in CSV:", df.columns)


cpu_daily = df['cpu_milli'].resample('D').mean()  # resample daily, take mean CPU

cpu_daily = cpu_daily.interpolate(method='time')


print(f"Data length after resample: {len(cpu_daily)}")


seasonal_periods = 7

model = ExponentialSmoothing(cpu_daily, trend='add', seasonal='add', seasonal_periods=seasonal_periods)
fit = model.fit()


forecast = fit.forecast(3)

print("Holt-Winters forecast for next 3 days:")
print(forecast)

# Plotting last 30 days + forecast
plt.figure(figsize=(10,6))
plt.plot(cpu_daily[-30:], label='Historical Daily CPU Usage')
plt.plot(forecast.index, forecast.values, label='Forecast', marker='o')
plt.xlabel('Date')
plt.ylabel('CPU Milli')
plt.title('CPU Usage Holt-Winters Forecast')
plt.legend()
plt.grid(True)
plt.show()
```



Step9:
Create and run compare.py to calculate MAPE:



Step10:
Create and run stats_comparison.py and graph.py to generate graph:



References

Kubernetes (2025). Kubernetes. [Online]. Available: *https://kubernetes.io.* [Accessed: 10- Aug- 2025].
URL: https://kubernetes.io/

Minikube. (2025). Minikube. [Online]. Available*: https://minikube.sigs.k8s.io.* [Accessed: 10- Aug- 2025].
URL: https://minikube.sigs.k8s.io