National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Mohd Sajid Ali
Student ID: x23293519

School of Computing
National College of Ireland

Supervisor: Ahmad Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mohd Sajid Ali		
Student ID:	X23293519		
Programme:	MSc Cloud Computing	**Year:**	2025/26
Module:	MSc Research Project		
Lecturer:	Ahmad Makki		
Submission Due Date:	11/12/2025		
Project Title:	A Cross-Environment Evaluation Framework for Container Orchestration Tools: Optimizing Efficiency and Implementation Trade-offs for Real-World Workloads		
Word Count:	**1368**	**Page Count**	**13**

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: Mohd Sajid Ali

Date: 11th December 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):	

A Cross-Environment Evaluation Framework for Container Orchestration Tools: Optimizing Efficiency and Implementation Trade-offs for Real-World Workloads

Configuration Manual

Mohd Sajid Ali
x23293519

1 Introduction

This configuration manual describes how to reproduce the on-premises, cloud, and hybrid testbed used to evaluate Kubernetes (k3s), Docker Swarm, HashiCorp Nomad, and Red Hat MicroShift with a Django Expense Tracker workload. The steps are organized to match the experimental methodology:

1. Infrastructure setup for on-premises VMs, AWS EC2, and the hybrid layout.
2. Deployment commands for each orchestrator (Kubernetes, Docker Swarm, Nomad, MicroShift).
3. Running Apache JMeter load tests for each scenario and environment.
4. Collecting metrics using the Python export script and other scripts.
5. Executing the ANOVA analysis script.

2 Infrastructure Setup

The testbed consists of:
- Two on-premises VMs (CentOS Stream 10 and RHEL 9) on VMware Workstation.
- One AWS EC2 instance per orchestrator in the cloud (Amazon Linux 2023 or RHEL 9).
- A hybrid setup where orchestrators run on EC2 and monitoring + JMeter run on-prem.

2.1 On-Premises VMs (VMware workstation)

Component	Specification
Host OS	CentOS Stream 10 (Selinux=Permissive)/Rhel 9
vCPUs	2/4
RAM	3/6 GB
Storage	30 GB SSD/30 & 20 GB
Networking	NAT (192.168.74.x), firewall open (ports 8000, NodePort range)
Tools	Docker Engine, k3s, Nomad, MicroShift, Prometheus, Grafana, JMeter

On both VMs, configure basic OS settings (run as root or with sudo):

Set SELinux permissive (CentOS/RHEL):
sudo setenforce 0

sudo sed -i 's/^SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/config

Update packages and install base tools:
sudo dnf update -y

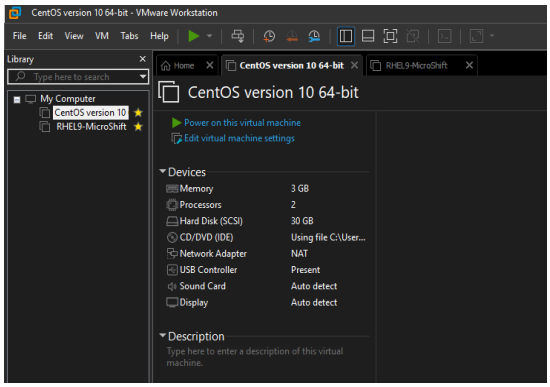
sudo dnf install -y git curl wget vim tar unzip net-tools python3-pip

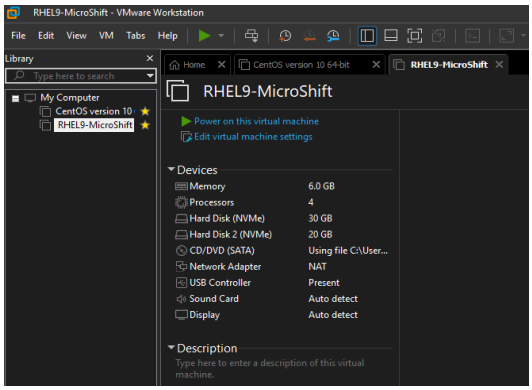
Install Docker from the official repository:
sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo dnf install -y docker-ce docker-ce-cli containerd.io

Enable and start Docker:
sudo systemctl enable --now docker

Add your user to the docker group (log out and back in after this):
sudo usermod -aG docker $USER





2.2 Cloud (AWS EC-2 Instance)

For each Operating System in the cloud environment, provision a separate EC2 instance in us-east-1:

Parameter	Value
Instance Type	t2.medium/t3.large (2 vCPU, 4/8 GB RAM)
OS	Amazon Linux 2023/Rhel 9
Storage	40 GB EBS
Network	Default VPC (public + private IP)
Orchestrator	Nomad/K8s/Swarm/Microshift (single-node)
Monitoring	Prometheus 2.55, Grafana 11
Load Test	JMeter on EC2

After launching, connect via SSH:
ssh ec2-user@<EC2_PUBLIC_IP> # Amazon Linux

ssh clouduser@<EC2_PUBLIC_IP> # RHEL 9

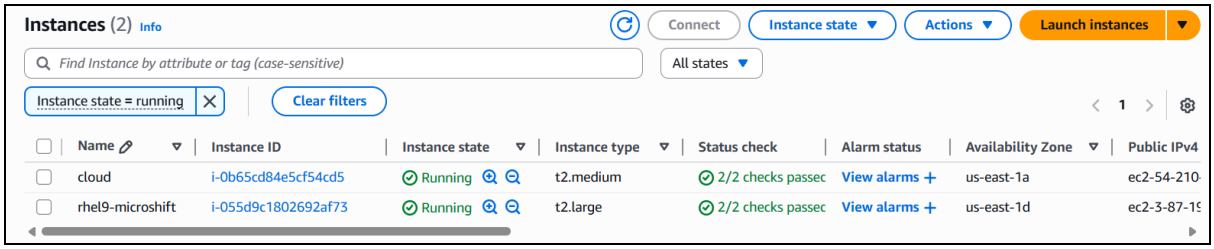
Update packages and install tools (Amazon Linux 2023):
sudo dnf update -y

sudo dnf install -y git curl wget vim tar unzip python3-pip

Install Docker on Amazon Linux 2023:
sudo dnf install -y docker

sudo systemctl enable --now docker

sudo usermod -aG docker ec2-user



2.3 Hybrid (AWS EC-2 Instance + On-Prem VMs)

In the hybrid setup:
- On-prem VMs host Prometheus, Grafana, Blackbox Exporter, Node Exporter, and JMeter.
- Each orchestrator and the Expense Tracker workload run on a single EC2 instance.

Component	On-Prem VM (Monitoring)	Cloud EC2 (Orchestration)
VM OS	CentOS Stream 10/Rhel 9	Amazon Linux 2023/Rhel 9
vCPUs/RAM	2/4 vCPU, 3/6 GB RAM	2 vCPU, 4/8 GB RAM
Orchestrators	Docker-Compose (monitoring only)	Swarm, K8s, Nomad, Microshift (single-node)
Monitoring	Prometheus, Grafana, Blackbox	Node Exporter (exporters only)
Load Testing	JMeter (CRUD tests)	*N/A*
Communication	HTTP + metrics over public IP / SG rules	Exposed services on public IP:8000

Security group rules on each EC2 node should allow inbound traffic for:
- SSH: 22/tcp (from on-prem public IP only)
- Application HTTP: 8000/tcp (Django Expense Tracker)
- Node Exporter: 9100/tcp (from on-prem Prometheus)
- cAdvisor: 8080/tcp (from on-prem Prometheus)

Example security group (ingress):

Type	Port	Source
SSH	22	<ON_PREM_PUBLIC_IP>
HTTP	8000	<ON_PREM_PUBLIC_IP>
Custom TCP	9100	<ON_PREM_PUBLIC_IP>
Custom TCP	8080	<ON_PREM_PUBLIC_IP>

```
[onprem@localhost expense_tracker]$
sudo firewall-cmd --zone=trusted --remove-source=10.42.0.0/16 --permanent
sudo firewall-cmd --zone=public --remove-port=80/tcp --permanent
sudo firewall-cmd --zone=public --remove-port=443/tcp --permanent
sudo firewall-cmd --zone=public --remove-port=5353/udp --permanent
sudo firewall-cmd --reload
```

3 Deployment Commands for each Orchestrator and Monitoring stack

3.1 Kubernetes (k3s)

Install k3s on the orchestrator node:

curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION="v1.22.17+k3s1" sh -s - --write-kubeconfig-mode 644

Verify cluster is running:

sudo kubectl get nodes

NodePort service, expose HTTP on port 8000 and note the node IP:

kubectl get svc -n expense-tracker

Apply manifests to run the application stack:

```
[onprem@localhost k8s]$ ls -lrth
total 24K
-rw-r--r--. 1 onprem docker  186 Oct  6 19:56 secret-db.yaml
-rw-r--r--. 1 onprem docker   57 Oct  6 19:56 namespace.yaml
-rw-r--r--. 1 onprem onprem 2.5K Oct  9 13:40 app.yaml
-rw-r--r--. 1 onprem onprem  175 Oct  9 13:40 local-path-sc.yaml
-rw-r--r--. 1 onprem onprem 1.2K Oct  9 13:40 postgres.yaml
drwxr-xr-x. 2 onprem onprem 4.0K Oct 14 16:08 monitoring
drwxr-xr-x. 2 onprem onprem  172 Nov 19 00:12 scripts
[onprem@localhost k8s]$
kubectl apply -f k8s/secret-db.yaml -f k8s/postgres.yaml -f k8s/app.yaml
kubectl -n expense get pods,svc
```

3.2 Docker Swarm

Initialise a Swarm on the Docker node (on-prem or EC2):

docker swarm init --advertise-addr <NODE_IP>

The repo contains a Swarm stack file such as swarm/stack.yml, deploy:
docker stack deploy -c swarm/stack.yml expense-tracker

Verify services and tasks:
docker service ls
docker service ps expense-tracker_web

```
export $(grep -v '^#' .env | xargs)
docker stack deploy -c stack.yml ${STACK_NAME}
docker stack services ${STACK_NAME}
docker stack ps ${STACK_NAME}
```

The application should be reachable on:
http://<NODE_IP>:8000/

3.3 HashiCorp Nomad

Download Nomad 1.4.x and install (on CentOS/RHEL/Amazon Linux):
curl -fsSL https://releases.hashicorp.com/nomad/1.4.3/nomad_1.4.3_linux_amd64.zip -o nomad.zip

unzip nomad.zip

sudo mv nomad /usr/local/bin/

Create a minimal server/client config in /etc/nomad.d/nomad.hcl and enable the systemd service.

Run Nomad and verify:
sudo systemctl enable --now nomad

nomad node status

The repo contains a job file such as nomad/expense.nomad.hcl, run:

```
[onprem@localhost expense_tracker]$ cd nomad/
[onprem@localhost nomad]$ ls
expense.nomad.hcl  scripts
[onprem@localhost nomad]$
```

nomad job run nomad/expense.nomad.hcl

3.4 Microshift

On the RHEL 9 VM dedicated to MicroShift, enable required repositories and install MicroShift 4.13:

sudo subscription-manager register --username <RH_USERNAME> --password <RH_PASSWORD>
sudo subscription-manager attach --auto
sudo subscription-manager repos --enable rhocp-4.13-for-rhel-9-$(arch)-rpms
*sudo dnf install -y microshift-4.13**

Enable and start MicroShift:
sudo systemctl enable --now microshift

Kubeconfig is usually written to /var/lib/microshift/resources/kubeadmin/kubeconfig.

Export:

export KUBECONFIG=/var/lib/microshift/resources/kubeadmin/kubeconfig

Create Project and Namespace:

```
oc new-project expense-app
oc create namespace expense-app
oc status
```

Set path and necessary permissions and run tools required to send scraped data to Prometheus:

```
KUBECONFIG_PATH="/var/lib/microshift/resources/kubeadmin/kubeconfig"
sudo chown onprem:onprem $KUBECONFIG_PATH
source ~/.bashrc
```

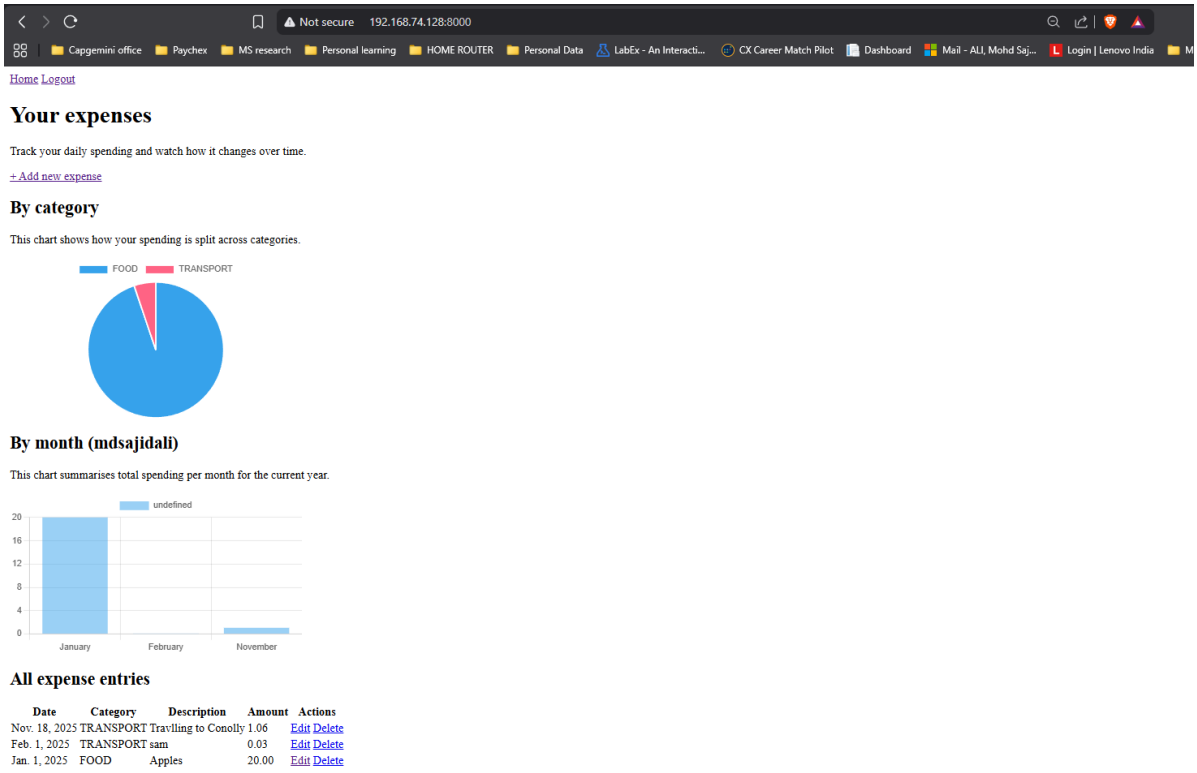
```
oc adm policy add-scc-to-user privileged -z cadvisor -n expense-tracker
oc apply -f cadvisor-daemonset.yaml
oc get route cadvisor -o jsonpath='{.spec.host}{"\n"}'
```

Run application and database along with route service:

```
[onprem@localhost microshift]$ ls -lrth
total 32K
-rw-r--r--. 1 onprem onprem  175 Oct 28 18:37 service.yaml
-rw-r--r--. 1 onprem onprem  194 Oct 28 18:37 route.yaml
-rw-r--r--. 1 onprem onprem  360 Oct 28 18:37 README.md
-rw-r--r--. 1 onprem onprem  705 Oct 28 18:37 postgres.yaml
-rw-r--r--. 1 onprem onprem  910 Oct 28 18:37 deployment.yaml
drwxr-xr-x. 2 onprem onprem   79 Oct 29 15:30 base
-rw-r--r--. 1 root   root    295 Oct 29 16:06 pv-postgres.yaml
drwxr-xr-x. 2 onprem onprem   92 Oct 29 21:01 db
drwxr-xr-x. 2 onprem onprem   95 Oct 29 21:30 app
drwxr-xr-x. 2 onprem onprem 4.0K Nov  2 01:02 monitoring
drwxr-xr-x. 2 onprem onprem 4.0K Nov 19 00:32 scripts
[onprem@localhost microshift]$
```

```
oc apply -f postgres.yaml
oc apply -f expense-deployment.yaml
oc apply -f service.yaml
oc apply -f route.yaml
```

Running Application on the VM ip and port:



3.5 Common monitoring stack for each Orchestrator

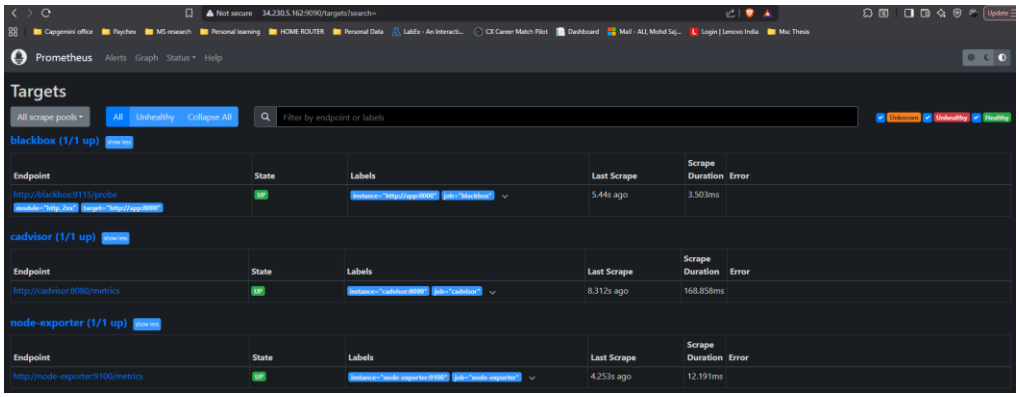
Monitoring stack runs on docker compose for the tools on separate containers.

docker compose -f monitoring-compose.yml up -d

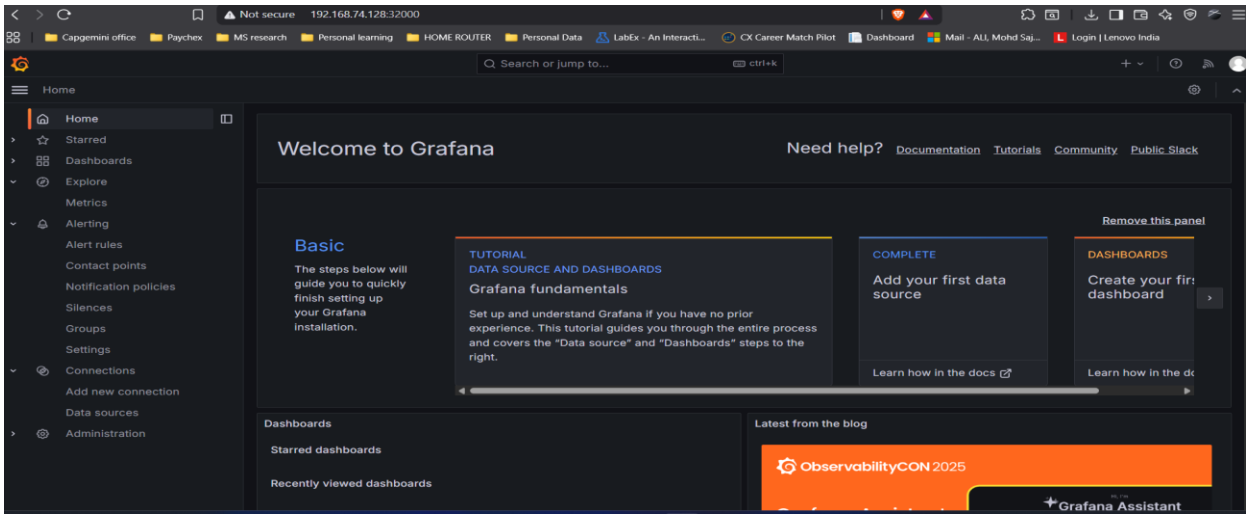
```
[onprem@localhost monitoring]$ ls -lrth
total 12K
-rw-r--r--. 1 onprem onprem 235 Oct 28 18:37 prometheus.yml
-rw-r--r--. 1 onprem onprem   0 Oct 28 18:37 grafana-dashboard.json
drwxr-xr-x. 2 onprem onprem  28 Oct 28 18:37 grafana
drwxr-xr-x. 3 onprem onprem  22 Oct 28 18:37 exporters
-rw-r--r--. 1 onprem onprem 353 Oct 28 18:37 docker-compose.yml
drwxr-xr-x. 2 onprem onprem  98 Oct 31 17:03 prometheus
-rw-r--r--. 1 onprem onprem 950 Nov  2 01:44 microshift_grafana-dashboard.json
drwxr-xr-x. 3 onprem onprem 146 Nov 19 00:37 scripts
[onprem@localhost monitoring]$ cat docker-compose.yml
version: "3.9"
services:
  prometheus:
    image: prom/prometheus:v2.54.1
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana:11.2.0
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=adminpass
[onprem@localhost monitoring]$
```

Prometheus dashboard:



Grafana Dashboard:



4 Apache Jmeter load tests for each scenario

Run below commands to install Jmeter and use the available **expense_crud_test.jmx** file to execute the Jmeter bash script.

Alos, create folders for results and report to store the metrics.

```
sudo dnf update -y
sudo dnf install -y tar wget java-17-openjdk java-17-openjdk-devel

cd /opt
sudo wget https://dlcdn.apache.org/jmeter/binaries/apache-jmeter-$JMETER_VERSION.tgz

sudo tar -xzf apache-jmeter-$JMETER_VERSION.tgz
sudo chown -R $USER:$USER apache-jmeter-$JMETER_VERSION

export JMETER_HOME=/opt/apache-jmeter-$JMETER_VERSION
export PATH=$JMETER_HOME/bin:$PATH

jmeter -v
```

```
echo 'export JMETER_HOME=/opt/apache-jmeter-5.6.3' >> ~/.bashrc
echo 'export PATH=$JMETER_HOME/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

4.1 Baseline Scenario

```
[onprem@localhost jmeter]$ ls -lrth
total 3.6M
-rwxr-xr-x.  1 onprem onprem 1.3K Oct 28 18:37 backup_run_jmeter.sh
-rwxr-xr-x.  1 onprem onprem 1.9K Oct 28 18:37 run_jmeter.sh
-rwxr-xr-x.  1 onprem onprem  408 Oct 28 18:37 run_jmeter_nomad.sh
drwxr-xr-x. 33 onprem onprem 4.0K Nov  4 20:49 report
drwxr-xr-x.  2 onprem onprem 4.0K Nov  4 21:25 results
-rw-r--r--.  1 onprem onprem 5.9K Nov 19 00:32 backup_expense_crud_test.jmx
-rw-r--r--.  1 onprem onprem  15K Nov 19 00:32 expense_crud_test_microshift_hybrid.jmx
-rw-r--r--.  1 onprem onprem 8.1K Nov 19 00:37 expense_crud_test.jmx
```

```
cd ~/researchproject/jmeter

export ORCHESTRATOR="nomad/k8s/swarm/microshift"
export SCENARIO="baseline"
./run_jmeter.sh 50 120 baseline
```

4.2 Spike Scenario

```
cd ~/researchproject/jmeter

export ORCHESTRATOR="nomad/k8s/swarm/microshift"
export SCENARIO="spike"
./run_jmeter.sh 200 120 spike
```

4.3 Fault Scenario

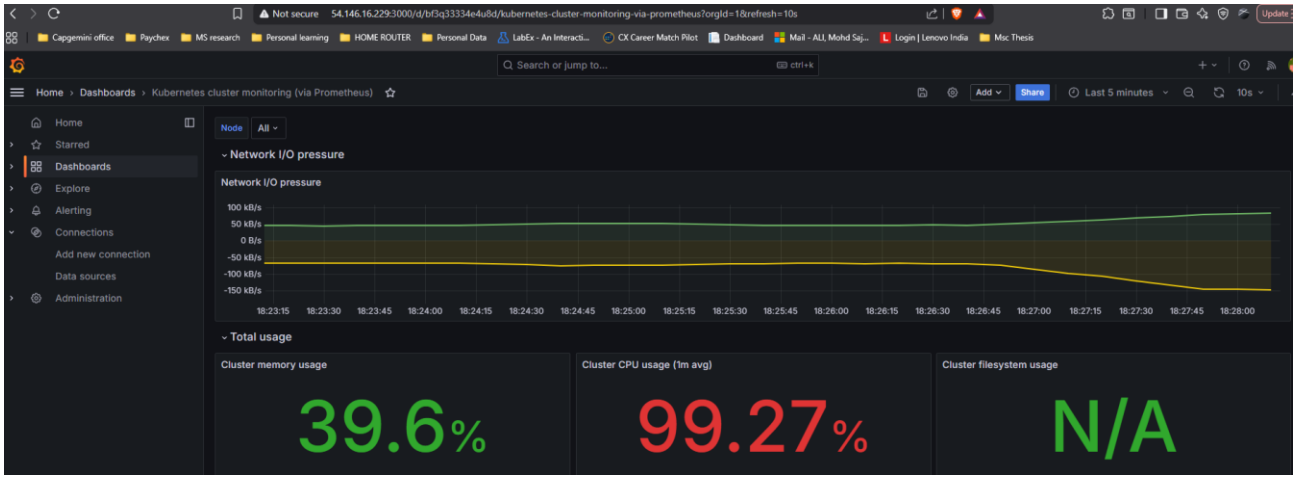
```
cd ~/researchproject/jmeter

export ORCHESTRATOR="nomad/k8s/swarm/microshift"
export SCENARIO="fault"
./run_jmeter.sh 100 120 fault
```

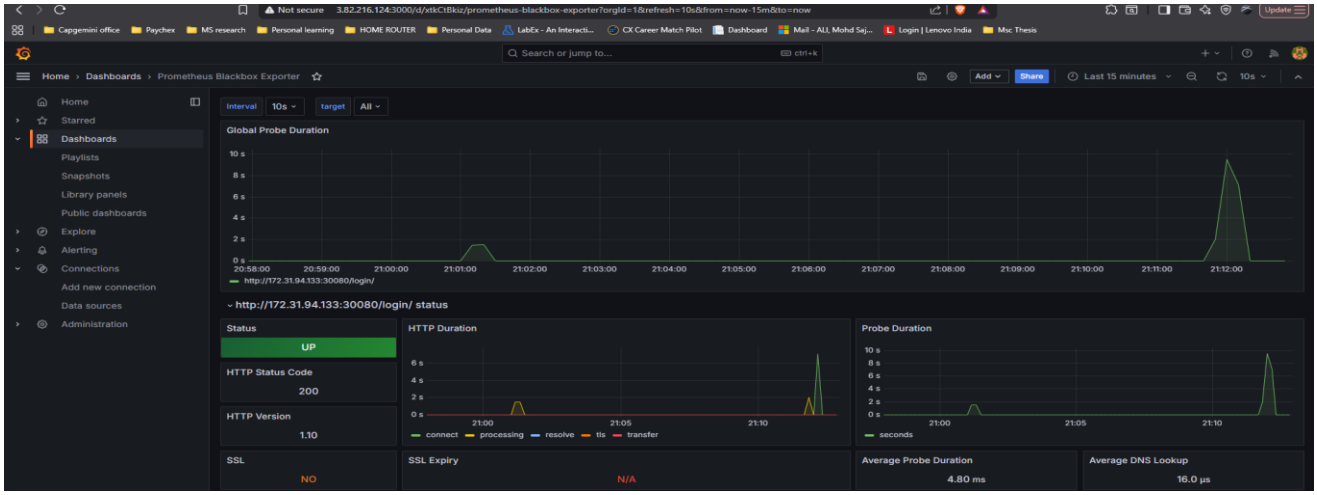
Running Jmeter script:

```
[onprem@localhost jmeter]$ ./run_jmeter.sh 50 120
[INFO] Running JMeter test plan with 50 threads for 120s...
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using expense_crud_test.jmx
Starting standalone test @ 2025 Oct 14 16:44:41 IST (1760456681319)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary +    551 in 00:00:41 =   13.5/s Avg:     7 Min:     0 Max:  1014 Err:   551 (100.00%) Active: 50 Started: 50 Finished: 0
summary +    190 in 00:00:31 =    6.2/s Avg:     5 Min:     0 Max:   653 Err:   190 (100.00%) Active: 50 Started: 50 Finished: 0
summary =    741 in 00:01:11 =   10.4/s Avg:     7 Min:     0 Max:  1014 Err:   741 (100.00%)
summary +    207 in 00:00:29 =    7.1/s Avg:     1 Min:     0 Max:   270 Err:   207 (100.00%) Active: 50 Started: 50 Finished: 0
summary =    948 in 00:01:41 =    9.4/s Avg:     6 Min:     0 Max:  1014 Err:   948 (100.00%)
summary +    155 in 00:00:31 =    5.0/s Avg:     8 Min:     0 Max:   644 Err:   155 (100.00%) Active: 50 Started: 50 Finished: 0
summary =   1103 in 00:02:12 =    8.4/s Avg:     6 Min:     0 Max:  1014 Err:  1103 (100.00%)
```

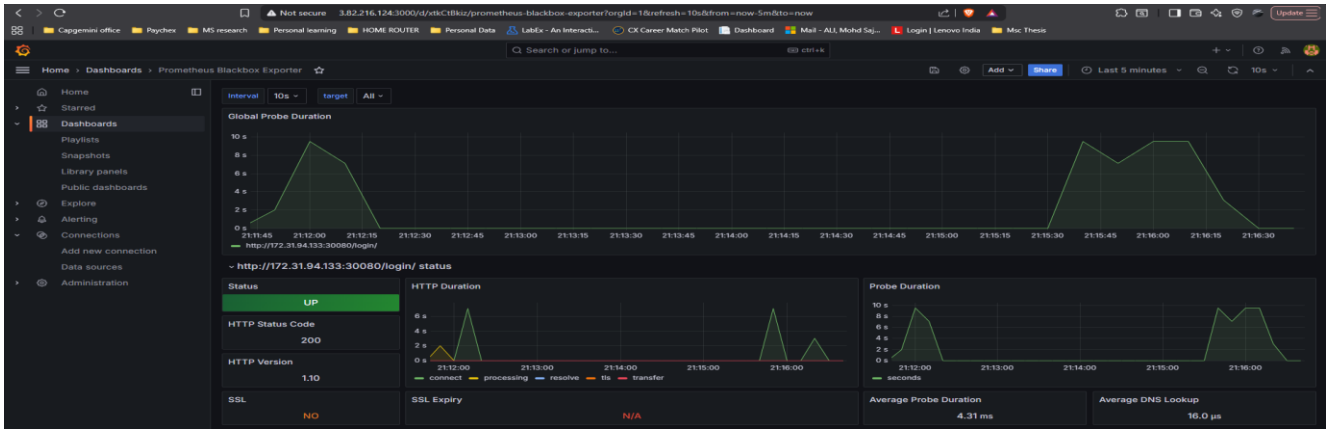
Baseline:



Spike:



Fault:



5 Scripts to capture metrics

After each JMeter run, use the Python export script (e.g. export_metrics.py) to pull Prometheus time-series data and aggregate them into CSV files for analysis.

On the monitoring node (where Prometheus is running):

```
cd ~/monitoring/scripts
```

```
export PROM_URL=http://127.0.0.1:9090/api/v1/query (or IP of the target Prometheus)
export ORCHESTRATOR="k8s"
export SCENARIO="baseline"
python3 export_metrics.py
export_metrics.py uses environment variables:
PROM_URL   = os.getenv("PROM_URL", "http://127.0.0.1:9091/api/v1/query")
ORCHESTRATOR = os.getenv("ORCHESTRATOR", "microshift").lower()
SCENARIO     = os.getenv("SCENARIO", "baseline").lower()
```

The script builds queries for CPU, Memory and HTTP uptime and writes CSVs such as metrics_k8s_baseline_YYYYMMDD_HHMMSS.csv.
Repeat for:
- ORCHESTRATOR = "swarm", "nomad", "microshift"
- SCENARIO = "fault", "spike"

```
[onprem@localhost expense_tracker]$ python3 monitoring/scripts/export_metrics.py
[INFO] Starting metric collection from http://192.168.74.128:32290/api/v1/query
[INFO] Writing data to ./data/metrics_k8s.csv every 60s

[INFO] 2025-10-14T16:41:31.874305 | cpu_usage:0.0125 | memory_usage:1811693568.0000 | http_uptime:0.0000
[INFO] 2025-10-14T16:42:32.187962 | cpu_usage:0.0125 | memory_usage:1885765632.0000 | http_uptime:0.0000
[INFO] 2025-10-14T16:43:32.403558 | cpu_usage:0.0121 | memory_usage:1891213312.0000 | http_uptime:0.0000
[INFO] 2025-10-14T16:44:32.526321 | cpu_usage:0.0260 | memory_usage:2009214976.0000 | http_uptime:0.0000
[WARN] Query failed for http_uptime: HTTPConnectionPool(host='192.168.74.128', port=32290): Read timed out. (read timeout=10)
[INFO] 2025-10-14T16:45:32.923532 | cpu_usage:0.0211 | memory_usage:2071265280.0000 | http_uptime:0.0000
[INFO] 2025-10-14T16:46:44.022171 | cpu_usage:0.0141 | memory_usage:0.0000 | http_uptime:0.0000
[INFO] 2025-10-14T16:47:46.860815 | cpu_usage:0.0149 | memory_usage:2162212864.0000 | http_uptime:0.0000
```

The export_metrics.py runs as a simple loop that periodically queries Prometheus for CPU, memory and HTTP uptime, and appends timestamped rows to a scenario-specific CSV file, providing a reproducible and orchestrator-agnostic metric export pipeline.

```
# ---------------------------------------------------------------------
# DYNAMIC METRIC DEFINITIONS
# ---------------------------------------------------------------------
def get_queries(orc: str):
    orc = orc.lower()

    # ---------- K8s HYBRID (host-level metrics via node-exporter) ----------
    if orc in ["k8s", "kubernetes"]:
        return {
            # Overall node CPU usage (excluding idle) for the k8s EC2 host
            "cpu_usage": 'avg(rate(node_cpu_seconds_total{mode!="idle",job="k8s_cloud_host"}[1m]))',
            # Node memory used = total - available
            "memory_usage": (
                'avg(node_memory_MemTotal_bytes{job="k8s_cloud_host"} '
                '- node_memory_MemAvailable_bytes{job="k8s_cloud_host"})'
            ),
            # HTTP uptime from blackbox job for k8s
            "http_uptime": 'avg_over_time(probe_success{job="blackbox_http_k8s"}[1m])'
        }

    # ---------- Swarm ----------
    elif orc in ["swarm", "docker", "docker-swarm"]:
        return {
            "cpu_usage": 'sum(rate(container_cpu_usage_seconds_total{container_label_com_docker_swarm_service_name="expense_app"}[1m]))',
            "memory_usage": 'avg(container_memory_usage_bytes{container_label_com_docker_swarm_service_name="expense_app"})',
            "http_uptime": 'avg_over_time(probe_success{job="blackbox_http_swarm"}[1m])'
        }

    # ---------- Nomad ----------
    elif orc == "nomad":
        return {
            "cpu_usage": 'avg(rate(node_cpu_seconds_total{mode!="idle",job="nomad_cloud_host"}[1m]))',
            "memory_usage": 'avg(node_memory_MemTotal_bytes{job="nomad_cloud_host"} - node_memory_MemAvailable_bytes{job="nomad_cloud_host"})',
            "http_uptime": 'avg_over_time(probe_success{job="blackbox_http_nomad"}[1m])'
        }

    # ---------- MicroShift ----------
    elif orc in ["microshift", "openshift"]:
        return {
            "cpu_usage": 'avg(rate(node_cpu_seconds_total{mode!="idle",job="microshift_cloud_host"}[1m]))',
            "memory_usage": 'avg(node_memory_MemTotal_bytes{job="microshift_cloud_host"} - node_memory_MemAvailable_bytes{job="microshift_cloud_host"})',
            "http_uptime": 'avg_over_time(probe_success{job="blackbox_http_microshift"}[1m])'
        }

    # ---------- Fallback ----------
    else:
        print(f"[WARN] Unknown orchestrator '{orc}', using generic node metrics.")
        return {
            "cpu_usage": 'avg(rate(node_cpu_seconds_total{mode!="idle"}[1m]))',
            "memory_usage": 'avg(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes)',
            "http_uptime": 'avg_over_time(probe_success[1m])'
        }

METRICS = get_queries(ORCHESTRATOR)
```

Use the CSVs and JTLs for plots and ANOVA in analysis/ as required.

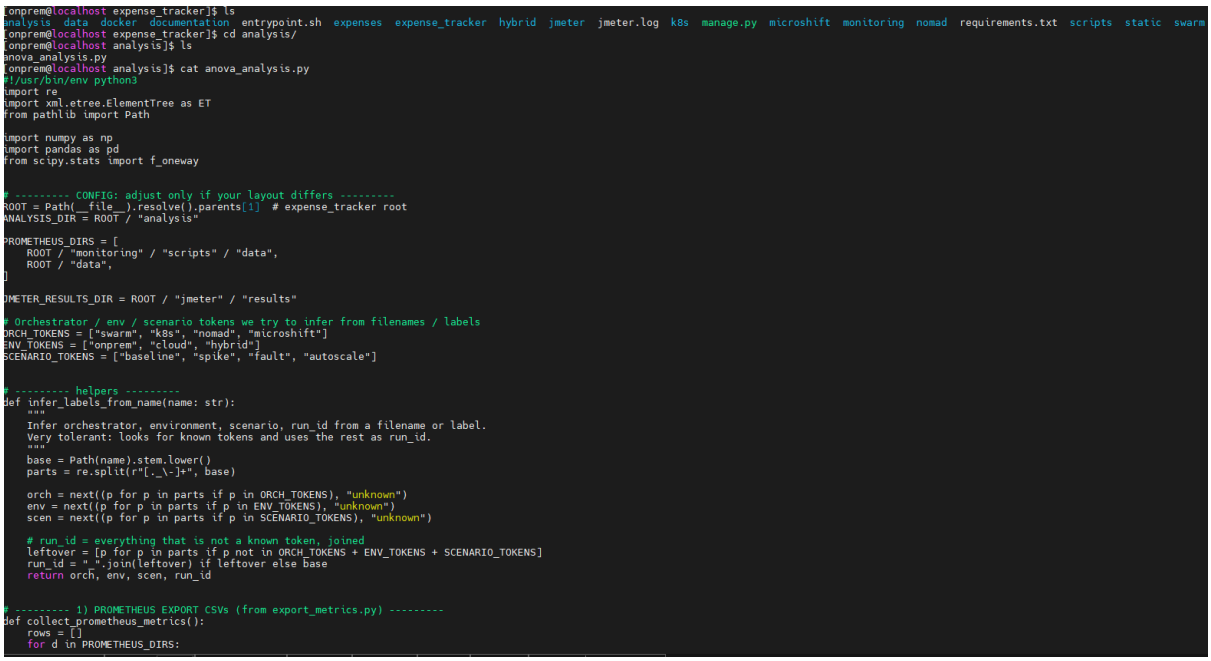
6 Anova Analysis

Once all metrics CSV files have been generated (latency, throughput, CPU, memory, uptime, etc.), run the ANOVA analysis script to statistically compare orchestration platforms.

The script should:
- Load all metrics/*.csv files.
- Compute one-way ANOVA for key metrics (average latency, throughput, CPU, memory, uptime).
- Write per-metric p-values and group means to results/anova_summary.csv for inclusion in the report.

cd ~/research_project/expense_tracker/analysis

*python3 anova_analysis.py *
*--input-dir ../metrics *
--output results/anova_summary.csv



```
[onprem@localhost expense_tracker]$ ls
analysis  data  docker  documentation  entrypoint.sh  expenses  expense_tracker  hybrid  jmeter  jmeter.log  k8s  manage.py  microshift  monitoring  nomad  requirements.txt  scripts  static  swarm
[onprem@localhost expense_tracker]$ cd analysis/
[onprem@localhost analysis]$ ls
anova_analysis.py
[onprem@localhost analysis]$ cat anova_analysis.py
#!/usr/bin/env python3
import re
import xml.etree.ElementTree as ET
from pathlib import Path

import numpy as np
import pandas as pd
from scipy.stats import f_oneway


# --------- CONFIG: adjust only if your layout differs ---------
ROOT = Path(__file__).resolve().parents[1]  # expense_tracker root
ANALYSIS_DIR = ROOT / "analysis"

PROMETHEUS_DIRS = [
    ROOT / "monitoring" / "scripts" / "data",
    ROOT / "data",
]

JMETER_RESULTS_DIR = ROOT / "jmeter" / "results"

# Orchestrator / env / scenario tokens we try to infer from filenames / labels
ORCH_TOKENS = ["swarm", "k8s", "nomad", "microshift"]
ENV_TOKENS = ["onprem", "cloud", "hybrid"]
SCENARIO_TOKENS = ["baseline", "spike", "fault", "autoscale"]


# --------- helpers ---------
def infer_labels_from_name(name: str):
    """
    Infer orchestrator, environment, scenario, run_id from a filename or label.
    Very tolerant: looks for known tokens and uses the rest as run_id.
    """
    base = Path(name).stem.lower()
    parts = re.split(r"[._\-]+", base)

    orch = next((p for p in parts if p in ORCH_TOKENS), "unknown")
    env = next((p for p in parts if p in ENV_TOKENS), "unknown")
    scen = next((p for p in parts if p in SCENARIO_TOKENS), "unknown")

    # run_id = everything that is not a known token, joined
    leftover = [p for p in parts if p not in ORCH_TOKENS + ENV_TOKENS + SCENARIO_TOKENS]
    run_id = "_".join(leftover) if leftover else base
    return orch, env, scen, run_id


# --------- 1) PROMETHEUS EXPORT CSVs (from export_metrics.py) ---------
def collect_prometheus_metrics():
    rows = []
    for d in PROMETHEUS_DIRS:
```

References

Prakash, R. (2023) “Benchmarking Container Orchestration Tools on Application Performance in the Cloud.”
Link: **https://norma.ncirl.ie/7403/1/ramprakash.pdf**

Malhotra, R., Bansal, A. and Kessentini, M. (2024) “A Systematic Literature Review on Maintenance of Software Containers,” *ACM Computing Surveys*, 56(8), pp. 1–38.

Link: https://doi.org/10.1145/3645092.

Malviya, A. and Dwivedi, R.K. (2022) "A Comparative Analysis of Container Orchestration Tools in Cloud Computing," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India: IEEE, pp. 698–703.
Link: https://doi.org/10.23919/INDIACom54597.2022.9763171.

Pan, Y. *et al.* (2019) "A Performance Comparison of Cloud-Based Container Orchestration Tools," in *2019 IEEE International Conference on Big Knowledge (ICBK),* Beijing, China: IEEE, pp. 191–198.
Link: https://doi.org/10.1109/ICBK.2019.00033.

Pankowski, A. and Powroźnik, P. (2023) "Comparison of application container orchestration platforms," *Journal of Computer Sciences Institute*, 29, pp. 383–390.
Link: https://doi.org/10.35784/jcsi.3823.