National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Oluseyi Iyetomide Simon Coker
Student ID: x23370751

School of Computing
National College of Ireland

Supervisor: Dr. Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Oluseyi Iyetomide Simon Coker
Student ID:	x23370751
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr. Abubakr Siddig
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Page Count:	12

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:	Oluseyi Iyetomide Simon Coker
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):	

Configuration Manual

Oluseyi Iyetomide Simon Coker
x23370751

1 Introduction

This configuration manual provides a complete, step-by-step guide for reproducing all experiments carried out in the research project on Kubernetes failure recovery and resilience. The purpose of this document is straightforward: if another person follows the instructions presented here, they should be able to recreate the same environments, run the same tests, and obtain results consistent with those reported in the main dissertation.

The manual describes the two environments used throughout the study: a local Minikube setup running on Windows 11 via WSL2, and a lightweight K3s cluster hosted on an AWS EC2 instance. Both environments were intentionally kept simple, consistent, and aligned with the requirements of the workload, ensuring that the behaviour of Kubernetes could be observed clearly under controlled failure scenarios.

All configuration steps, commands, manifests, and directory structures shown in this manual reflect the actual workflow used during the project. This includes installation of required tools, deployment of the application under test, setup of monitoring systems, execution of failure injections, and generation of analysis plots. The goal is not only reproducibility, but clarity—so each experiment can be repeated reliably without additional assumptions.

By following this manual, readers will be able to reproduce the baseline test, pod crash scenario, CPU stress scenario, and network impairment scenario across both local and cloud environments, replicating the full experimental pipeline from deployment to evaluation.

2 Prerequisites and Installation

This section describes the software prerequisites and installation steps required to reproduce the local and cloud environments used in this project. All commands are shown for the configuration actually used (Windows 11 host, WSL2 Ubuntu 22.04, Minikube with Docker, K3s on Amazon Linux 2023, and `wrk`-based load generation). If you are already running a native Linux desktop or server, you can skip the WSL2 setup and start from the Docker installation onwards.

2.1 Host System and WSL2

These steps are only required when running on Windows. Native Linux users can proceed directly to the Docker Engine subsection.

1. Ensure the host operating system is **Windows 10/11** with support for WSL2.

2. Enable WSL and the Virtual Machine Platform from an elevated PowerShell:

```
dism.exe /online /enable-feature ^
  /featurename:Microsoft-Windows-Subsystem-Linux ^
  /all /norestart

dism.exe /online /enable-feature ^
  /featurename:VirtualMachinePlatform ^
  /all /norestart
```

3. Install the WSL2 kernel update (if prompted) and set WSL2 as default:

```
wsl --set-default-version 2
```

4. Install **Ubuntu 22.04** from the Microsoft Store and complete the initial user setup.

2.2 Docker Engine (with WSL2 Integration)

1. Install Docker Desktop for Windows.
2. Enable WSL2 integration for Ubuntu in Docker Desktop settings.
3. Validate Docker availability inside Ubuntu WSL2 or native Linux:

```
docker version
docker info
```

2.3 kubectl, Minikube, and Helm

kubectl

```
sudo apt update
sudo apt install -y curl ca-certificates gnupg lsb-release

curl -LO "https://dl.k8s.io/release/$(curl -L -s \
  https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client
```

Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
minikube version
```

Helm

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 \
  | bash

helm version
```

2.4 wrk HTTP Load Generator

```
sudo apt install -y build-essential git libssl-dev
git clone https://github.com/wg/wrk.git
cd wrk
make
sudo cp wrk /usr/local/bin/
wrk --version
```

2.5 CPU Stress Tooling (stress / stress-ng)

```
sudo apt install -y stress-ng
stress-ng --version
```

(In Kubernetes, the experiments use `polinux/stress` with `stress --cpu 2 --timeout 60s`.)

2.6 Python Virtual Environment (for Plotting)

```
sudo apt install -y python3 python3-pip python3-venv

cd ~/Research_Project
python3 -m venv experiments/venv
source experiments/venv/bin/activate

pip install matplotlib pandas
```

2.7 AWS CLI and SSH / EC2 Instance Connect

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" \
  -o "awscliv2.zip"

unzip awscliv2.zip
sudo ./aws/install
aws --version

aws configure
```

To connect to the EC2 instance you can either:

- Use SSH from your terminal:

  ```
  ssh -i /path/to/key.pem ec2-user@EC2_PUBLIC_IP
  ```

- Or use **EC2 Instance Connect** from the AWS Management Console to open a browser-based shell.

All cloud-side commands in this manual assume you have a shell open on the EC2 instance via one of these methods.

3 Project Directory Structure

All project files were organised under `~/Research_Project` on the WSL2 Ubuntu environment. The layout below shows only the folders and files that are relevant for reproducing the experiments.

```
~/Research_Project
 experiments/
    baseline/
    netem/
    pod_crash/
    resource_stress/
    local/
       baseline/
       netem/
       pod_crash/
       resource_stress/
       summary_local.csv
    cloud/
       baseline/
       netem/
       pod_crash/
       resource_stress/
       summary_cloud.csv
    resilience_radar.png
    combined_requests.png
    combined_latency.png
    rto_timeline.png
    plot_local_results.py
    plot_cloud_results.py
    plot_combined_results.py
    plot_rto_timeline.py
    plot_radar.py
 k8s/
    stress-job.yaml           # local CPU stress job
    myapp-toxiproxy.yaml      # local / reference network-fault manifest
 selfhealing-lab/
     k8s/
        app-deployment.yaml
        app-service.yaml
        myapp-netem.yaml      # tc netem variant (used in early prototypes)
        stress-job.yaml
```

```
scripts/
    summarize_cloud.sh
    summarize_local.sh
```

On the EC2 instance used for cloud experiments the working directory was the home folder of the `ec2-user` account. To simplify copying manifests via SCP, the main files were placed directly under `/home/ec2-user`:

```
/home/ec2-user
 Research_Project/              # report, plots, and experiment scripts
 experiments/                   # cloud wrk outputs
 myapp.yaml                     # base K3s deployment + service
 myapp-toxiproxy.yaml           # cloud network-fault manifest (Toxiproxy)
 stress-job.yaml                # cloud CPU stress job
```

The `experiments/` directory is where all local and cloud runs were stored and where the plotting scripts expect to find their input CSV files. The `selfhealing-lab/` folder is a GitHub repository used during development; it contains earlier versions of the Kubernetes manifests and helper scripts but is not required to reproduce the final experiments as long as the manifests listed above are available in either `k8s/` (local) or the EC2 home directory (cloud).

4 Local Kubernetes Environment (Minikube)

4.1 Accessing Grafana (Web Dashboard)

Once the kube-prometheus-stack is installed, Grafana becomes available inside the `monitoring` namespace. Forward the service to your local machine:

```
kubectl -n monitoring port-forward \
$(kubectl -n monitoring get pod -l app.kubernetes.io/name=grafana -o name) \
3000:3000
```

Grafana URL: `http://localhost:3000`

Login Credentials

The default Grafana credentials (as installed by kube-prometheus-stack) are:

- Username: `admin`
- Password: `prom-operator`

If these do not work, retrieve the password directly:

```
kubectl -n monitoring get secret kps-grafana -o jsonpath="{.data.admin-password}" \
  | base64 --decode
```

Useful Grafana Dashboards

Grafana contains many pre-built dashboards. The most useful ones for this project were:

- **Kubernetes / Compute Resources / Pod** – shows CPU and memory per pod.
- **Kubernetes / API Server** – useful for confirming that the control plane stays healthy.
- **Kubernetes / Networking / Namespace (Pods)** – useful for spotting spikes during netem experiments.

These dashboards help visualise pod restarts, CPU throttling and sudden latency shifts during fault-injection runs, although the experience depends on how quickly Grafana loads and whether the port-forward remains stable.

Practical Note

During this project, Grafana occasionally became slow to load or disconnected during port-forwarding. For that reason, Prometheus was used as the primary monitoring tool during live experiments because its UI reacts instantly and shows raw counter values without delay.

4.2 Accessing Prometheus (Recommended for Experiments)

While Grafana provides dashboards, Prometheus is the most reliable interface for observing real-time behaviour during the experiments. Forward the Prometheus service:

```
kubectl -n monitoring port-forward \
svc/kps-kube-prometheus-stack-prometheus 9090:9090
```

Prometheus URL: `http://localhost:9090`

Queries Used During Experiments

Prometheus was used to confirm pod restarts, CPU pressure and event timing. The main queries were:

```
container_cpu_usage_seconds_total{namespace="app"}
container_memory_working_set_bytes{namespace="app"}
kube_pod_container_status_restarts_total{namespace="app"}
```

Prometheus updates every few seconds and does not rely on dashboards, making it easier to detect:

- Pod restarts after deletion
- CPU starvation during stress tests
- Live behaviour during network impairment
- Whether the cluster stabilises after a fault

For most experiments, Prometheus provided clearer and more immediate visibility than Grafana.

4.3 Starting Minikube

The experiments assume a single-node Minikube cluster with 2 vCPUs and 4 GB memory. For a new cluster, run:

```
minikube start --driver=docker --cpus=2 --memory=4g
kubectl get nodes
```

If a Minikube profile already exists, Minikube will reuse it and print warnings that CPU and memory cannot be changed. In that case you can either keep the existing cluster, or delete it and start again:

```
minikube delete
minikube start --driver=docker --cpus=2 --memory=4g
```

4.4 Namespaces

```
kubectl create namespace app
kubectl create namespace monitoring
kubectl get namespaces
```

If you have run the project before, `kubectl` may report `Error from server (AlreadyExists)` for these commands. This simply means the namespaces are already in place and can be safely ignored.

4.5 Installing Prometheus and Grafana

```
helm repo add prometheus-community \
  https://prometheus-community.github.io/helm-charts
helm repo update

helm install kps prometheus-community/kube-prometheus-stack \
  -n monitoring

kubectl -n monitoring get pods
```

4.6 Deploying the Application Under Test

Deployment (2 replicas)

```
kubectl -n app create deployment myapp \
  --image=hashicorp/http-echo:0.2.3 -- \
  -text=ok -listen=:5678

kubectl -n app scale deployment myapp --replicas=2
```

If the deployment already exists, `kubectl` will report `"deployments.apps m̈yappälready exists"`. In that case you can either keep the existing deployment or recreate it:

```
kubectl -n app delete deployment myapp
kubectl -n app create deployment myapp \
  --image=hashicorp/http-echo:0.2.3 -- \
  -text=ok -listen=:5678
```

Service

```
kubectl -n app expose deployment myapp \
  --port=80 --target-port=5678 --type=ClusterIP
```

If the service already exists, `"services m̈yappälready exists"` indicates that no further action is needed.

4.7 Verifying the Setup

```
kubectl -n app get deployments
kubectl -n app get pods
kubectl -n app get svc

kubectl -n app port-forward svc/myapp 8080:80
curl http://localhost:8080/
```

5 Terminal Layout for Experiments and Demo

For both the experiments and the live demo, multiple terminals were kept open in parallel. This made it easier to run `wrk`, watch pods, and keep Grafana and Prometheus port-forwards active at the same time.

5.1 Recommended Terminal Roles (Local Environment)

- **Terminal A – Main control**: general `kubectl` commands, applying manifests, running scripts.
- **Terminal B – App port-forward** (leave running):

  ```
  kubectl -n app port-forward svc/myapp 8080:80
  ```

- **Terminal C – Grafana port-forward** (leave running):

  ```
  kubectl -n monitoring port-forward \
  $(kubectl -n monitoring get pod -l app.kubernetes.io/name=grafana -o name) \
  3000:3000
  ```

- **Terminal D – Prometheus port-forward** (optional, leave running):

  ```
  kubectl -n monitoring port-forward \
  svc/kps-kube-prometheus-stack-prometheus 9090:9090
  ```

- **Terminal E – Watch pods / events** (optional):

  ```
  kubectl -n app get pods -w
  ```

5.2 Terminal Layout in the Cloud Environment

When connected to the EC2 instance (via SSH or EC2 Instance Connect), the same layout can be reproduced using multiple terminal windows or a terminal multiplexer such as `tmux`. One shell is used as the main control terminal, and additional shells are kept open for port-forwarding Grafana/Prometheus and watching pods during cloud experiments.

6 Cloud Environment and Experiment Procedures

6.1 K3s on AWS EC2 (Amazon Linux 2023)

The cloud experiments ran on a `t3.small` EC2 instance (2 vCPUs, 2 GB RAM) using Amazon Linux 2023.

K3s Installation

```
sudo dnf update -y

curl -sfL https://get.k3s.io | sh -

alias kubectl="sudo k3s kubectl"
kubectl get nodes
```

Namespaces, Monitoring, Application Deployment

```
kubectl create namespace app
kubectl create namespace monitoring

helm repo add prometheus-community \
  https://prometheus-community.github.io/helm-charts
helm repo update

helm install kps prometheus-community/kube-prometheus-stack \
  -n monitoring

kubectl -n app create deployment myapp \
  --image=hashicorp/http-echo:0.2.3 -- \
  -text=ok -listen=:5678

kubectl -n app scale deployment myapp --replicas=2

kubectl -n app expose deployment myapp \
  --port=80 --target-port=5678 --type=ClusterIP
```

CloudWatch was used for EC2 CPU and network metrics, and artefacts were copied to S3 for external analysis.

6.2 CloudWatch Monitoring and IAM Limitations

An attempt was made to install and run the Amazon CloudWatch Agent on the EC2 instance so that OS-level metrics such as memory and detailed disk usage could be visualised alongside the experiments. The agent was installed and configured, but metrics were not delivered to CloudWatch because the AWS Academy Learner Lab environment does not permit creating or attaching IAM roles.

Agent logs repeatedly reported missing credentials and the absence of an instance profile. As a result, the `CWAgent` namespace and custom metrics do not appear in CloudWatch. For this project, standard EC2 metrics (for example `CPUUtilization`, `NetworkIn`, `NetworkOut` and status checks) were used instead. These native metrics require no extra IAM configuration and were sufficient to confirm CPU pressure and basic instance health during the experiments.

6.3 Baseline Experiment

Here and in the following scenarios, `tee` is used so that `wrk` output is printed to the terminal and saved to a file.

```
kubectl -n app port-forward svc/myapp 8080:80

wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/baseline/wrk_t2c50_60s.txt
```

6.4 Pod Crash Scenario

```
wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/pod_crash/wrk_podcrash_60s.txt

kubectl -n app delete pod \
$(kubectl -n app get pods -l app=myapp \
-o jsonpath='{.items[0].metadata.name}')


kubectl -n app get events --sort-by=.metadata.creationTimestamp \
  > experiments/pod_crash/events_tail.txt
```

6.5 Resource Stress Scenario

For the CPU saturation experiment a separate pod runs a CPU–burning workload on the same node as the application. The pod never crashes, so Kubernetes does not treat this as a failure, but latency and throughput degrade.

Local Minikube

```
kubectl apply -f k8s/stress-job.yaml

wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/resource_stress/wrk_stress_60s.txt
```

```
kubectl -n app describe job stress-job \
  > experiments/resource_stress/job_desc.txt

kubectl -n app get events --sort-by=.metadata.creationTimestamp \
  > experiments/resource_stress/events_tail.txt
```

Cloud K3s on EC2

On the EC2 instance the same job manifest was copied to the home directory as `stress-job.yaml`. The commands are identical apart from the path:

```
kubectl apply -f stress-job.yaml

wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/resource_stress/wrk_stress_60s.txt

kubectl -n app describe job stress-job \
  > experiments/resource_stress/job_desc.txt

kubectl -n app get events --sort-by=.metadata.creationTimestamp \
  > experiments/resource_stress/events_tail.txt
```

6.6 Network Fault Scenario (Netem / Toxiproxy)

The final scenario introduced artificial network delay and loss. Two variants were used during the project:

- a **tc netem** sidecar manifest for the local Minikube cluster; and
- a **Toxiproxy**-based manifest for the cloud K3s node, which was easier to manage on the constrained EC2 environment.

Both variants implement the same logical fault: approximately 200 ms of latency and 1 % packet loss on requests to the `myapp` service.

Local Minikube (tc netem)

The original Minikube experiments used the `myapp-netem.yaml` manifest from the `selfhealing-lab/k8` directory. This deploys a sidecar that configures `tc qdisc netem` inside the pod:

```
kubectl apply -f selfhealing-lab/k8s/myapp-netem.yaml

wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/netem/wrk_delay200ms_loss1pct.txt

kubectl delete -f selfhealing-lab/k8s/myapp-netem.yaml
```

Cloud K3s on EC2 (Toxiproxy)

On the EC2 node, a simpler Toxiproxy deployment was used. The manifest `myapp-toxiproxy.yaml` (placed in `/home/ec2-user`) creates a proxy in front of `myapp` and applies equivalent delay and loss settings.

```
kubectl apply -f myapp-toxiproxy.yaml

wrk -t2 -c50 -d60s http://localhost:8080/ \
  | tee experiments/netem/wrk_delay200ms_loss1pct.txt

kubectl delete -f myapp-toxiproxy.yaml
```

In both environments the resulting `wrk` outputs are stored under `experiments/netem/` and used to compute the network-resilience scores in the main report.

6.7 Result Processing and Plotting

```
cd ~/Research_Project
source experiments/venv/bin/activate

python3 experiments/plot_local_results.py
python3 experiments/plot_cloud_results.py
python3 experiments/plot_combined_results.py
python3 experiments/plot_rto_timeline.py
python3 experiments/plot_radar.py
```

The scripts read the summary CSV files under `experiments/local/` and `experiments/cloud/` and regenerate the plots used in the main report, including combined throughput, combined latency, RTO timeline and the resilience radar.

6.8 Cleanup Procedures

Local Cluster

```
minikube delete
```

Cloud Cluster

```
sudo systemctl stop k3s
sudo rm -rf /var/lib/rancher/k3s/
```