National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Hemadhri Govindaraju
Student ID: X23285052

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hemadhri Govindaraju
Student ID:	X23285052
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	808
Page Count:	9

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:	Hemadhri Govindaraju
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

Hemadhri Govindaraju
X23285052

1 Introduction

This manual gives a step-by-step guideline on how to install and implement a machine learning powered predictive auto-scaling system based on AWS EC2 instances. It helps users configure and deploy and experiment with LSTM, Prophet and Hybrid models to predict CPU utilization and dynamically scale.

2 Scope

The manual of configuration is applicable to:

- DevOps engineers or researchers deploying AWS ML-based auto-scaling.
- Time-series forecasting projects of cloud infrastructures.
- Setups with AWS EC2, CloudWatch, S3, Auto Scaling Groups (ASG) and Python-based ML models.

3 Objective

- Use predictive models (LSTM, Prophet, Hybrid) to predict CPU usage.
- Auto scaling EC2 on estimative workloads.
- Evaluate accuracy, SLA violations, cost, and latency in comparison to traditional ASG.
- Visualization of data in Grafana through CloudWatch metrics.

4 System Requirements

4.1 Hardware Requirements:

- AWS EC2 instance (t3.large recommended: 2 vCPUs, 8 GB RAM)

4.2 Software Requirements:

- OS: Ubuntu 20.04 or Amazon Linux 2
- Python 3.9+
- Pip, Virtualenv
- AWS CLI & IAM roles with S3, CloudWatch, and Auto Scaling permissions
- Required Python packages: boto3, pandas, numpy, tensorflow, prophet, scikit-learn, matplotlib

4.3 Other Services:

- AWS CloudWatch (with CloudWatch Agent installed)
- Amazon S3 (for model storage)
- AWS ASG setup
- Grafana (self-hosted or managed)

5 Installation Instruction

5.1 Python Environment

```
sudo apt update
sudo apt install python3-pip
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
```

5.2 Install CloudWatch Agent

```
sudo yum install amazon-cloudwatch-agent
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-
   wizard
sudo systemctl start amazon-cloudwatch-agent
```

5.3 CloudWatch Agent Configure:

Save file:

```
    sudo nano /opt/aws/amazon-cloudwatch-agent/bin/config.json
```

```
{
  "agent": {
    "metrics_collection_interval": 60,
    "run_as_user": "root"
```

```
  },
  "metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "cpu": {
        "measurement": [
          "cpu_usage_user",
          "cpu_usage_system",
          "cpu_usage_idle"
        ],
        "metrics_collection_interval": 60
      },
      "mem": {
        "measurement": ["mem_used_percent"],
        "metrics_collection_interval": 60
      },
      "disk": {
        "measurement": ["disk_used_percent"],
        "metrics_collection_interval": 60,
        "resources": ["/"]
      },
      "net": {
        "measurement": ["bytes_sent", "bytes_recv"],
        "metrics_collection_interval": 60,
        "resources": ["*"]
      }
    }
  }
}
```

5.4 Clone ML Auto-Scaling Repo

```
git clone https://github.com/x23285052-Hemadhri/ml\_auto\_scaling
cd ml\_auto\_scaling
```

5.5 Project Structure

```
/home/ec2-user/
|-- ml_scaling_lstmcontroller.py
|-- ml_scaling_prophetcontroller.py
|-- ml_scaling_hybridcontroller.py
|-- train_lstm_model.py
|-- train_prophet_model.py
|-- export_ml_asg_metrics.py
```

```
|-- logs/
|   `-- *.log
|-- env/
```

6 Execution

6.1 Workload simulation for testing:

Save the file:

```
    nano simulate.sh
```

```
# Loop CPU load: 60s stress, 60s cooldown
while true; do
  echo "[INFO] CPU Load for 60s"
  stress-ng --cpu 2 --timeout 60

  echo "[INFO] Memory Load for 60s"
  stress-ng --vm 1 --vm-bytes 500M --timeout 60

  echo "[INFO] I/O Load for 60s"
  stress-ng --io 2 --timeout 60

  echo "[INFO] Cooling down for 60s"
  sleep 60
done

echo "Workload simulation complete!"
```

Change sh mode and run

```
chmod +x simulate.sh
./simulate.sh
```

6.2 Schedule ML Controllers via Crontab

6.2.1 Install:

```
sudo yum install cronie -y
sudo systemctl start crond
sudo systemctl enable crond

crontab -e
```

6.2.2 ML Controller (every 1 minute):

```
* * * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   ml_scaling_lstmcontroller.py >> /home/ec2-user/logs/ml_lstm.log 2>&1
* * * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   ml_scaling_prophetcontroller.py >> /home/ec2-user/logs/ml_prophet.log
   2>&1
# OR use hybrid if that's your preferred method:
*/5 * * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   ml_scaling_hybridcontroller.py >> /home/ec2-user/logs/ml_hybrid.log
   2>&1
```

6.3 Train Models (every 3 hours)

```
0 */3 * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   train_prophet_model.py >> /home/ec2-user/logs/train_prophet.log 2>&1
0 */3 * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   train_lstm_model.py >> /home/ec2-user/logs/train_lstm.log 2>&1

# === Export ASG + EC2 metrics to S3 (every 6 hours) ===
0 */6 * * * /home/ec2-user/env/bin/python3 /home/ec2-user/
   export_ml_asg_metrics.py >> /home/ec2-user/logs/export_metrics.log
   2>&1
```

6.4 Push to CloudWatch

- Controllers push:
  - PredictedCPU (per model)
  - Action (1 = scale_up, 0 = no_action)

7 Experiments

7.1 Experiment 1: Traditional ASG vs ML-based Scaling

- Metrics: Avg CPU, SLA violations, cost

7.2 Experiment 2: Forecasting Accuracy

- Compare RMSE & MAE of LSTM vs Prophet

7.3 Experiment 3: Hybrid Evaluation

- Evaluate reduction in false positives

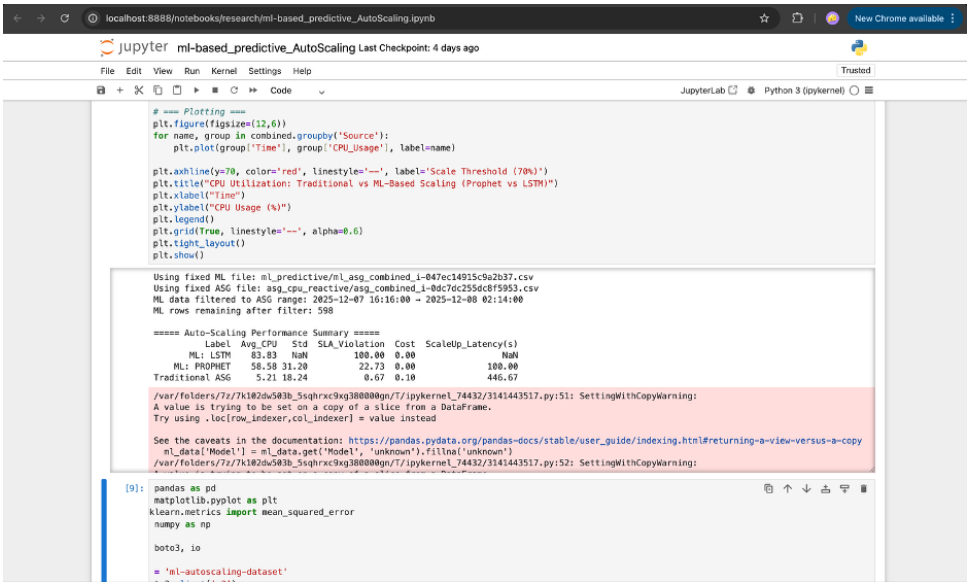

Figure 1: Metrics: Avg CPU, SLA violations, cost

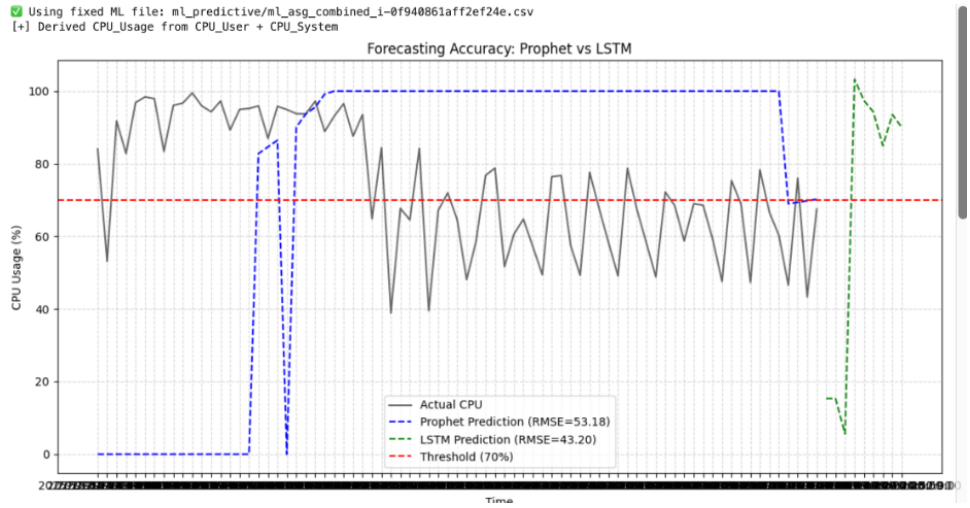

Figure 2: Forecasting Accuracy

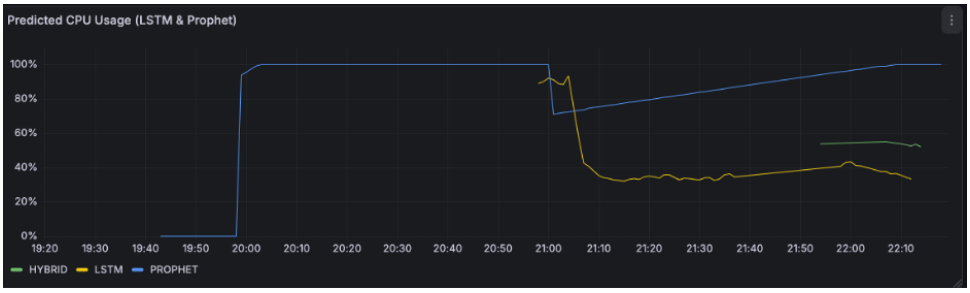

Figure 3: Predicted CPU

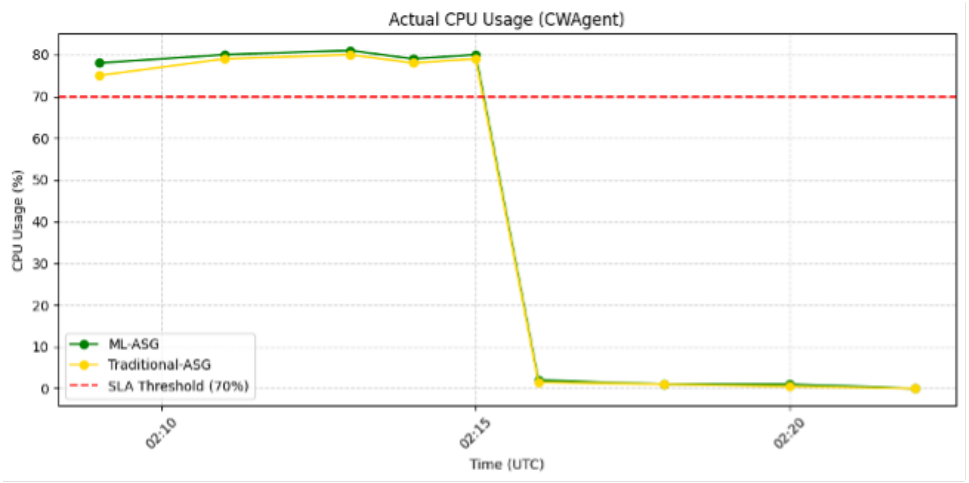

Figure 4: Actual CPU usage

7.4 Experiment 4: Real-Time Visualization

- Dashboard in Grafana showing:
  - Predicted CPU (LSTM, Prophet, Hybrid)
  - Actual CPU usage
  - Scaling decisions
  - Instance count over time

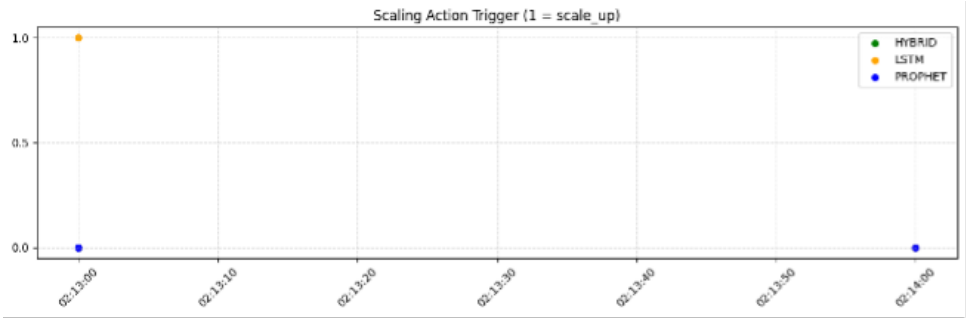

Figure 5: Scaling Decision

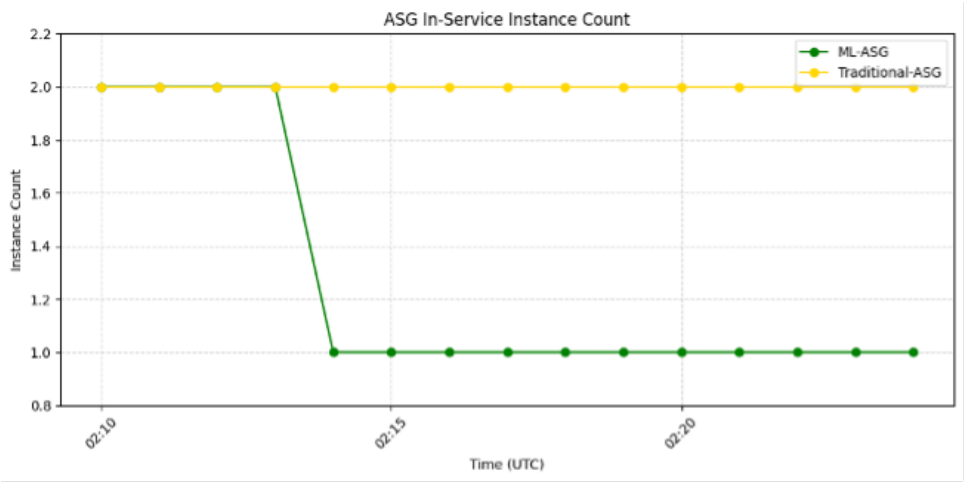

Figure 6: Instance Count

8 Outputs and Analysis

8.1 Logs

- Stored in ml_scaling_logs.csv
- Uploaded to s3://ml-autoscaling-dataset/ml_predictive/ml_scaling_logs.csv

8.2 Visualizations (Grafana Panels)

- Forecast vs Actual CPU
- Scaling action timeline
- ASG instance count

8.3 Evaluation Summary (Sample):

Scaling Strategy	Avg. CPU (%)	Std. Dev	SLA Violations (%)	Cost ($)	Scale-Up Latency (s)
ML: LSTM	83.83	N/A	100.00	0.00	N/A
ML: Prophet	58.58	31.20	22.73	0.00	100.00
Traditional ASG	5.21	18.24	0.67	0.10	446.67

Table 1: Performance Comparison of Traditional and ML-Based Auto-Scaling Strategies

9 Summary

This manual allows end to end application of predictive ML-bound auto-scaling on AWS EC2. Enhancing the cost, CPU efficiency, and SLA compliance during dynamic workloads, LSTM and Prophet model-based forecast-driven decisions are less expensive than alternative solutions to the problem. The integration of Grafana provides observability. The extensions that might be considered in the future feature Kubernetes support, RL-based adaptation, and anomaly-aware hybrid scaling.

References