Configuration Manual

MSc Research Project
MSc in Cloud Computing

Aryan Singh
Student ID: 23270152

School of Computing
National College of Ireland

Supervisor: Prof. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Singh
Student ID:	23270152
Programme:	MSc in Cloud Computing
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Prof. Punit Gupta
Submission Due Date:	15/09/2025
Project Title:	Configuration Manual
Word Count:	992
Page Count:	6

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:	Aryan Singh
Date:	14th September 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

Aryan Singh
23270152

1 Introduction

This how-to guide consists of description of installation and operation of the AI-Based Cloud Resource Forecasting and Auto-Scaling Simulation. It uses historical cloud workload data (2008-2017) to train predictive models (SARIMA and LSTM) on predicting (forecasting) CPU Utilization, Execution Time and Wait Time. The simulation extends forecasts up to 7 days consecutively beyond December 2017 and links a Scale Up, Scale Down or Hold decision with a threshold-based scaling mechanism. This write-up will guarantee that one is able to replicate simulation environment, training the models, and test the results without using live cloud infrastructure.

2 Required Tools and Software

Component	**Description**
Operating System	Windows 10/11, macOS, or Linux (Ubuntu 20.04 or later recommended)
Python Version	Python 3.10+
Integrated Development Environment (IDE)	PyCharm (Community or Professional) OR Visual Studio Code
Python Libraries	• `numpy` – numerical operations • `pandas` – data handling • `matplotlib` – plotting results • `scikit-learn` – data preprocessing & evaluation • `statsmodels` – SARIMA implementation • `tensorflow` / `keras` – LSTM implementation • `flask` – API for serving forecasts and scaling logic
Optional Tools	Jupyter Notebook – for interactive exploration

Table 1: Required Tools and Software for the Project

3 Configuration Steps

Step 1 – Download the Dataset

The following part describes how the same simulation may be reproduced using the CEMAT Euler workload dataset.

- Go on to the Parallel Workloads Archive Logs Page.
- Scroll down till you find a dataset named **CIEMAT Euler**.
- Click the download icon in the rightmost column (blue floppy disk symbol) to download the `.swf` file.
- Save the file at a local place (e.g., `datasets/ciemat_euler.swf`).

Step 2 – Install Python and Required Libraries

Ensure Python 3.10+ is installed:

```
python --version
```

Install required dependencies:

```
pip install -r requirements.txt
```

Step 3 – Load and Preprocess the Dataset

Use the `swfparser` library or a custom Python parser to read the `.swf` file .Convert UNIX timestamps to `datetime` objects and aggregate the data to the desired time granularity (e.g., hourly or daily).

```python
import pandas as pd

# Input SWF file
swf_file = "CIEMAT-Euler-2008-1.swf"
csv_output = "CIEMAT-Euler-2008-1_clean.csv"

# Proper column names as per SWF spec
columns = [
    "JobNumber", "SubmitTime", "WaitTime", "RunTime",
    "AllocatedProcessors", "UsedCPUTime", "UsedMemory",
    "RequestedProcessors", "RequestedTime", "RequestedMemory",
    "Status", "UserID", "GroupID", "Executable",
    "QueueNumber", "PartitionNumber", "PrecedingJob",
    "ThinkTimeFromPrecedingJob"
]

data_lines = []
with open(swf_file, 'r') as f:
    for line in f:
        line = line.strip()
        if not line or line.startswith(";"):
            continue  # skip comments
        parts = line.split()
        data_lines.append([int(x) for x in parts])

df = pd.DataFrame(data_lines, columns=columns)
df.to_csv(csv_output, index=False)

print(f"✅ Clean CSV saved as: {csv_output}")
```

Figure 1: CEMAT Clean

After creating the clean CSV file, convert UNIX timestamps to `datetime` objects and engineer new features for analysis: Extract the following fields:

- Arrival Time
- Entry Time
- Execution Time
- Completion Time
- CPU Utilization
- Wait Time

```python
# Base UnixStartTime in seconds (from SWF header)
unix_start_time = 1226926907
base_datetime = datetime.utcfromtimestamp(unix_start_time)

# If not already done, load the CSV and convert SubmitTime to datetime:
df1 = pd.read_csv("CIEMAT-Euler-2008-1_clean.csv")

# Add real ArrivalTime (datetime)
df1["ArrivalTime"] = df1["SubmitTime"].apply(lambda x: base_datetime + timedelta(seconds=int(x)))

# Compute EntryTime (SubmitTime + WaitTime)
df1["EntryTime_seconds"] = df1["SubmitTime"] + df1["WaitTime"]
df1["EntryTime"] = df1["EntryTime_seconds"].apply(lambda x: base_datetime + timedelta(seconds=int(x)))

# ExecutionTime is already RunTime, but for clarity:
df1["ExecutionTime"] = df1["RunTime"]

# CompletionTime = SubmitTime + WaitTime + RunTime
df1["CompletionTime_seconds"] = df1["SubmitTime"] + df1["WaitTime"] + df1["RunTime"]
df1["CompletionTime"] = df1["CompletionTime_seconds"].apply(lambda x: base_datetime + timedelta(seconds=int(x)))

# CPU Utilization = UsedCPUTime / RunTime
df1["CPU_Utilization"] = df1.apply(
    lambda row: row["UsedCPUTime"] / row["RunTime"] if row["RunTime"] > 0 else 0,
    axis=1
)
# WaitTime is already present
df1["WaitTime_seconds"] = df1["WaitTime"]

# Optional: drop intermediate *_seconds columns if not needed
df1.drop(columns=["EntryTime_seconds", "CompletionTime_seconds"], inplace=True)
```

Figure 2: UNIX timestamps conversion

Step 4 – Train SARIMA & LSTM Models

- Split the dataset into:
  - Training (e.g., 2008–2016)
  - Testing (e.g., 2017)
- Fit the SARIMA model for CPU usage forecasting.
- Fit the LSTM model for capturing non-linear workload patterns.
- Save trained models in a `models/` directory.

4 Run the Simulation

In order to carry out the simulation, launch the main Python script located at the root of the project:

```
python main.py
```

The simulation script performs the following operations:

- **Load Trained Models:** The already trained models SARIMA and LSTM in `models/` project directory are loaded into memory.
- **Forecast Workload:** The models make a CPU utilization forecast for the then subsequent n days relying on past workload patterns.
- **Apply Threshold-Based Scaling Rules:**
  - **Scale Up:** Activated when the predicted CPU of use goes over the upper limit (e.g., $> 80\%$).
  - **Scale Down:** It is activated in case of predicted CPU use that is less than the lower threshold (e.g., $< 30\%$).

```python
        # Decision logic
        if (cpu > cpu_threshold_up or wait > wait_mean + wait_std or exe > exe_mean + exe_std):
            action = "Scale Up"
            note = "High resource usage - Scaling & Security Check Triggered"
        elif (cpu < cpu_threshold_down and wait < wait_mean * 0.3 and exe < exe_mean * 0.3):
            action = "Scale Down"
            note = "Low usage - Downscale safe"
        else:
            action = "Hold"
            note = "Normal usage"

        predictions.append({
            "date": forecast_date.strftime("%Y-%m-%d"),
            "CPU_Utilization": round(cpu, 2),
            "WaitTime_seconds": round(wait, 2),
            "ExecutionTime": round(exe, 2),
            "action": action,
            "note": note
        })

    return jsonify(predictions)

except Exception as e:
    logging.exception("Error during forecasting:")
    return jsonify({"error": str(e)}), 500
```

Figure 3: Decision Logic

127.0.0.1:5001/forecast

retty print


```
{
  "CPU_Utilization": 0.0,
  "ExecutionTime": 269639.76,
  "WaitTime_seconds": 0.0,
  "action": "Scale Down",
  "date": "2018-01-01",
  "note": "Low usage - downscale safe"
},
{
  "CPU_Utilization": 0.26,
  "ExecutionTime": 216493.3,
  "WaitTime_seconds": 0.0,
  "action": "Scale Down",
  "date": "2018-01-02",
  "note": "Low usage - downscale safe"
},
{
  "CPU_Utilization": 0.56,
  "ExecutionTime": 214147.75,
  "WaitTime_seconds": 0.0,
  "action": "Hold",
  "date": "2018-01-03",
  "note": "Normal usage"
},
{
  "CPU_Utilization": 0.83,
  "ExecutionTime": 214986.02,
  "WaitTime_seconds": 5353.7,
  "action": "Hold",
  "date": "2018-01-04",
  "note": "Normal usage"
},
{
  "CPU_Utilization": 1.1,
  "ExecutionTime": 253174.95,
  "WaitTime_seconds": 12709.48,
  "action": "Hold",
  "date": "2018-01-05",
  "note": "Normal usage"
},
{
  "CPU_Utilization": 1.38,
```


Figure 4: Forecast

  - **Hold:** In case forecasted CPU usage is within the acceptable range no scaling action is taken.

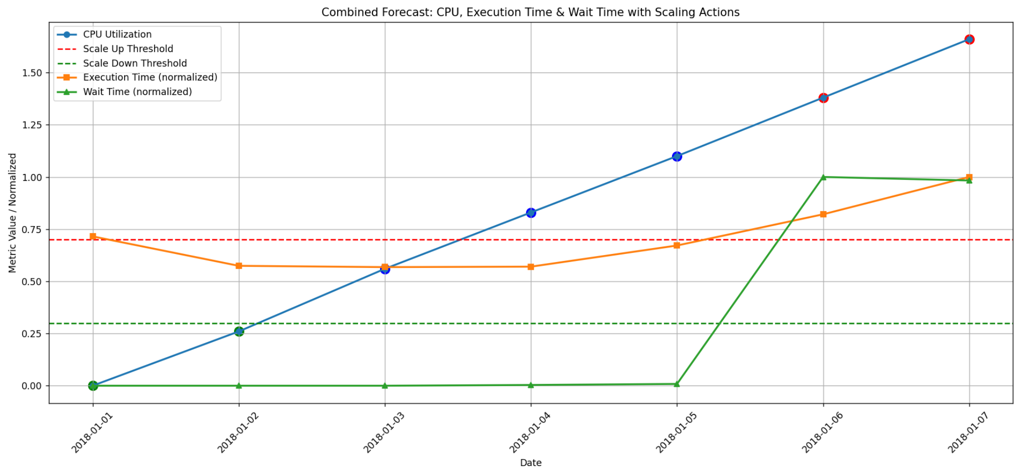

Figure 5: Prediction Graph

- **Log Actions:** All scaling decisions, and predictions are entered into simulation logs to be viewed later.

5 Validate the Results

Test the predicted and the actual workload values with the test dataset.

Feature	MAE	RMSE
CPU_Utilization	0.01	0.01
WaitTime_seconds	0.01	0.01
ExecutionTime	0.20	0.24

Feature	R^2 Score
CPU_Utilization	0.9996
WaitTime_seconds	0.9997
ExecutionTime	0.3183

Figure 6: Evaluation

- **Mean Absolute Error (MAE)**: It is a measurement of the average of the size of the errors including the direction of the errors not included in the predictions.

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N} |y_t - \hat{y}_t|$$

- **Root Mean Squared Error (RMSE)**: Depicts big errors more severely and forms some measure of spread of the errors.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}$$

Additionally, review the **scaling decision logs** to verify correctness:

- Ensure that *Scale Up* decisions occur, when CPU utilization and at least one of the wait time or execution time rides above their upper mentioned limits.
- Ensure that *Scale Down* decisions happen only after all of monitored metrics are below their low thresholds for 2 consecutive days.
- Confirm that *Hold* Step-by-step measures are taken when the use is not out of limits.

References