

Configuration Manual

MSc Research Project
Cloud Computing

Oluwaseyi Ezekiel Ademola
Student ID: 23240334

School of Computing
National College of Ireland

Supervisor: Ahmed Hamza Ibrahim

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Oluwaseyi Ezekiel Ademola
Student ID:	23240334
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Ahmed Hamza Ibrahim
Submission Due Date:	12th December 2024
Project Title:	Configuration Manual
Word Count:	601
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:	Oluwaseyi Ezekiel Ademola
Date:	12th December 2024

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

Oluwaseyi Ezekiel Ademola
23240334

1 Introduction

This configuration manual provides detailed instructions for configuring and running the research simulation experiment. It highlights steps taken to reproduce the simulation from data ingestion, execution of simulation experiment with custom algorithm up to the result evaluation analysis.

2 System Requirements

The experiment was focused on testing the ability of the proposed algorithm to improve cold start latency, execution time, and resource consumption of serverless functions primarily based on simulation with the iFogSim simulator toolkit (Buyya and Srirama; 2019) using the IntelliJ IDEA CE IDE. The table below illustrates the main components such as programming languages, tools, and libraries used.

Table 1: Tools and Technologies

Type	Tool/Technology	Description
Simulation framework	iFogSim	Simulation framework for cloud & edge computing
Programming Language	Java	For implementing simulation environment and location-aware algorithm and resources
Data Preparation	Python, Pandas, Google Colab	Used for preprocessing and visualization datasets
Version Control	Git & Github	Managing source code

3 Installation and Setup

3.1 Data Preparation Setup

The data preprocessing was carried out using Python scripts with Jupyter Notebook in Visual Studio IDE and the cleaned datasets used as input data to the simulation can be

retrieved from this repository <https://github.com/oluademola/dataset-prep>. The raw datasets can be retrieved from (Microsoft-Research; 2019) and (Microsoft-Research; 2011) for the Azure functions and T-Drive datasets respectively. The dataset should be placed following the file structure illustrated in Figure 1 to run the data preprocessing.

```

import pandas as pd
pd.set_option("display.max_rows",None)
pd.set_option("display.max_columns",None)
import os

# setting directories
input_dir = '../Datasets/azurefunctions-dataset2019'
output_file = '../Processed/processed_invocations_per_function.csv'
output_file2 = '../Processed/processed_function_durations.csv'
output_file3 = '../Processed/processed_app_memory.csv'

# List to hold processed DataFrames for each file
all_data_processed = []

# Loop over all 14 CSV files
for i in range(1, 15):
    file_path = os.path.join(input_dir, f'invocations_per_function_md.anon.d(i:82).csv')

    # Load the data
    df = pd.read_csv(file_path)

    # Select invocation columns and convert them to integers
    invocation_columns = [str(i) for i in range(1, 1441)]
    data_hourly = df[invocation_columns].copy()
    data_hourly.columns = range(1, 1441) # Convert column names to integers for grouping

    # Transpose, group by every 60 columns (for hourly aggregation), then transpose back
    data_hourly = data_hourly.T.groupby(lambda x: (x - 1) // 60).sum().T

    # Rename columns to hourly format (0 to 23)
    data_hourly.columns = [f'Hour_{i}' for i in range(24)]

    # Combine with original identifiers and trigger type
    data_processed = pd.concat([df[['HashOwner', 'HashApp', 'HashFunction', 'Trigger']], data_hourly), axis=1)

    # Identify peak hours for each function (e.g., top 3 peak hours)
    peak_hours = data_hourly.apply(lambda x: x.nlargest(3).index.tolist(), axis=1) # Top 3 peak hours

    # Add peak hours as a new column
    data_processed['Peak_Hours'] = peak_hours

    # Append the processed DataFrame to the list
    all_data_processed.append(data_processed)

final_data = pd.concat(all_data_processed, ignore_index=True)
row_count = len(final_data)
print(f"The number of rows in the file is: {row_count}")
# final_data.to_csv(output_file, index=False)
# print("Data processing complete. The output has been saved to", output_file)

... The number of rows in the file is: 618545

```

Figure 1: Data Preprocessing

3.2 iFogSim Simulation Setup

1. Clone the iFogSim simulator toolkit from its Git repository a folder:

```
git clone https://github.com/Cloudslab/iFogSim
```

2. Open the IntelliJ IDE
3. Click on the File dropdown menu and select New > "project from existing resources".
4. Confirm that the Java version and external libraries are added to the project.

5. Navigate to `src > org > fog > test > perfeval` and run any of the example applications to verify the installation with the required libraries was successful. You should get a response in the console as shown in Figure 2 below.

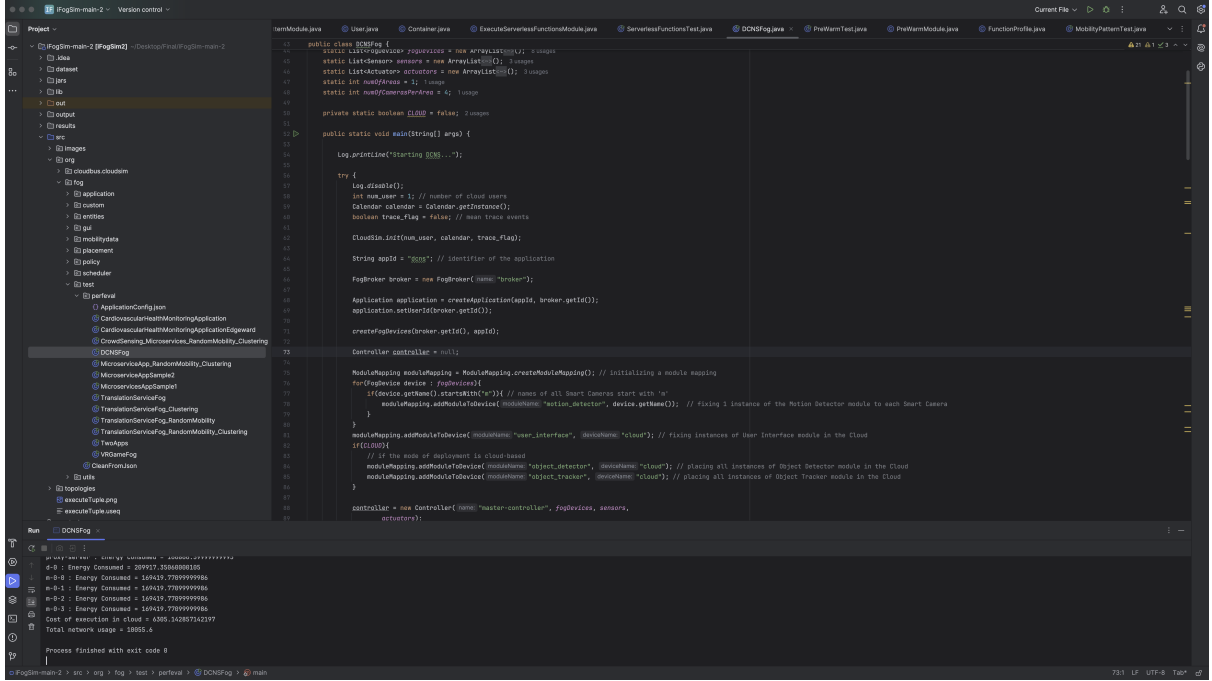


Figure 2: Example Application

4 Running Simulation Experiment

4.1 Project file setup

To configure the simulation environment. Setup the project files with the following steps:

1. Copy the preprocessed Azure Functions datasets to the project's `dataset/Processed1` folder.
2. Copy the preprocessed T-Drive datasets to the project's `dataset/T_drive` folder.
3. Copy the project codebase named `custom` to the `src > org > fog > folder`.
4. Verify the `custom` directory contains theses folders `mobility pattern`, `prewarm`, `serverlessfunctions`, and `test` with their appropriate java files within.
5. Ensure OpenCSV library use is less than version 5.0

4.2 Running the Experiment with the Algorithm

1. Run the first component of the algorithm by navigating to the `src > org > fog > custom > test > MobilityPatternTest` to run the mobility pattern module. It generates events that'd be used in the final execution of the functions for the effective execution of functions closer to the user.

The screenshot displays an IDE with two tabs: `MobilityPatternModule.java` and `MobilityPatternTest.java`. The `MobilityPatternTest.java` tab is active, showing the following code:

```

1 package org.fog.custom.test;
2
3 > import ...
4
5
6
7 public class MobilityPatternTest {
8
9     public static void main(String[] args) {
10         // Initialize the Mobility Pattern Module with the dataset path
11         MobilityPatternModule mobilityModule = new MobilityPatternModule( datasetPath: "dataset/T_drive/1.txt"
12
13         // Add some geofences
14         mobilityModule.addGeofence( name: "Geofence1", latitude: 48.7128, longitude: -74.0860, radius: 500); // E
15         mobilityModule.addGeofence( name: "Geofence2", latitude: 37.7749, longitude: -122.4194, radius: 300); // F
16
17         // Process user movements and detect geofence events
18         mobilityModule.processUserMovements((userId, geofenceName) -> {
19             System.out.println("User " + userId + " entered geofence: " + geofenceName);
20         });
21
22         // Print all users and their trajectories
23         System.out.println("\nAll Users and Trajectories:");
24         for (User user : mobilityModule.getUsers()) { // Use User type explicitly
25             System.out.println("User ID: " + user.getId());
26         }
27     }
28 }

```

The `Run` tab shows the output of the `MobilityPatternTest` class. The output consists of a series of location events, each represented by a line of text: "Location -> Latitude: [value], Longitude: [value]". The events are as follows:

```

Location -> Latitude: 116.691580, Longitude: 39.851650
Location -> Latitude: 116.691580, Longitude: 39.851650
Location -> Latitude: 116.691630, Longitude: 39.851690
Location -> Latitude: 116.691540, Longitude: 39.851730
Location -> Latitude: 116.691540, Longitude: 39.851750
Location -> Latitude: 116.691590, Longitude: 39.851670
Location -> Latitude: 116.691630, Longitude: 39.851410
Location -> Latitude: 116.691620, Longitude: 39.851610
Location -> Latitude: 116.691600, Longitude: 39.851610
Location -> Latitude: 116.691540, Longitude: 39.851880
Location -> Latitude: 116.691580, Longitude: 39.851970
Location -> Latitude: 116.691580, Longitude: 39.851870
Location -> Latitude: 116.691590, Longitude: 39.851830
Location -> Latitude: 116.691620, Longitude: 39.851470
Location -> Latitude: 116.691560, Longitude: 39.851580
Location -> Latitude: 116.691550, Longitude: 39.851640
Location -> Latitude: 116.691530, Longitude: 39.851650
Location -> Latitude: 116.691530, Longitude: 39.851660
Location -> Latitude: 116.661790, Longitude: 39.883720
Location -> Latitude: 116.603940, Longitude: 39.907410
Location -> Latitude: 116.545590, Longitude: 39.914740
Location -> Latitude: 116.521950, Longitude: 39.916000
Location -> Latitude: 116.485030, Longitude: 39.914220
Location -> Latitude: 116.444600, Longitude: 39.921560
Location -> Latitude: 116.400470, Longitude: 39.925940
Location -> Latitude: 116.441520, Longitude: 39.932360
Location -> Latitude: 116.483470, Longitude: 39.919540
Location -> Latitude: 116.507890, Longitude: 39.931280
Location -> Latitude: 116.531740, Longitude: 39.915360
Location -> Latitude: 116.571560, Longitude: 39.902630
Location -> Latitude: 116.547230, Longitude: 39.908410

```

At the bottom of the console, it says "Process finished with exit code 0".

Figure 3: Mobility Pattern Events

2. Run the Prwarm module that handles the container pre-warming based on geofence entry events by navigating to the `src > org > fog > custom > test > PreWarmTest`.

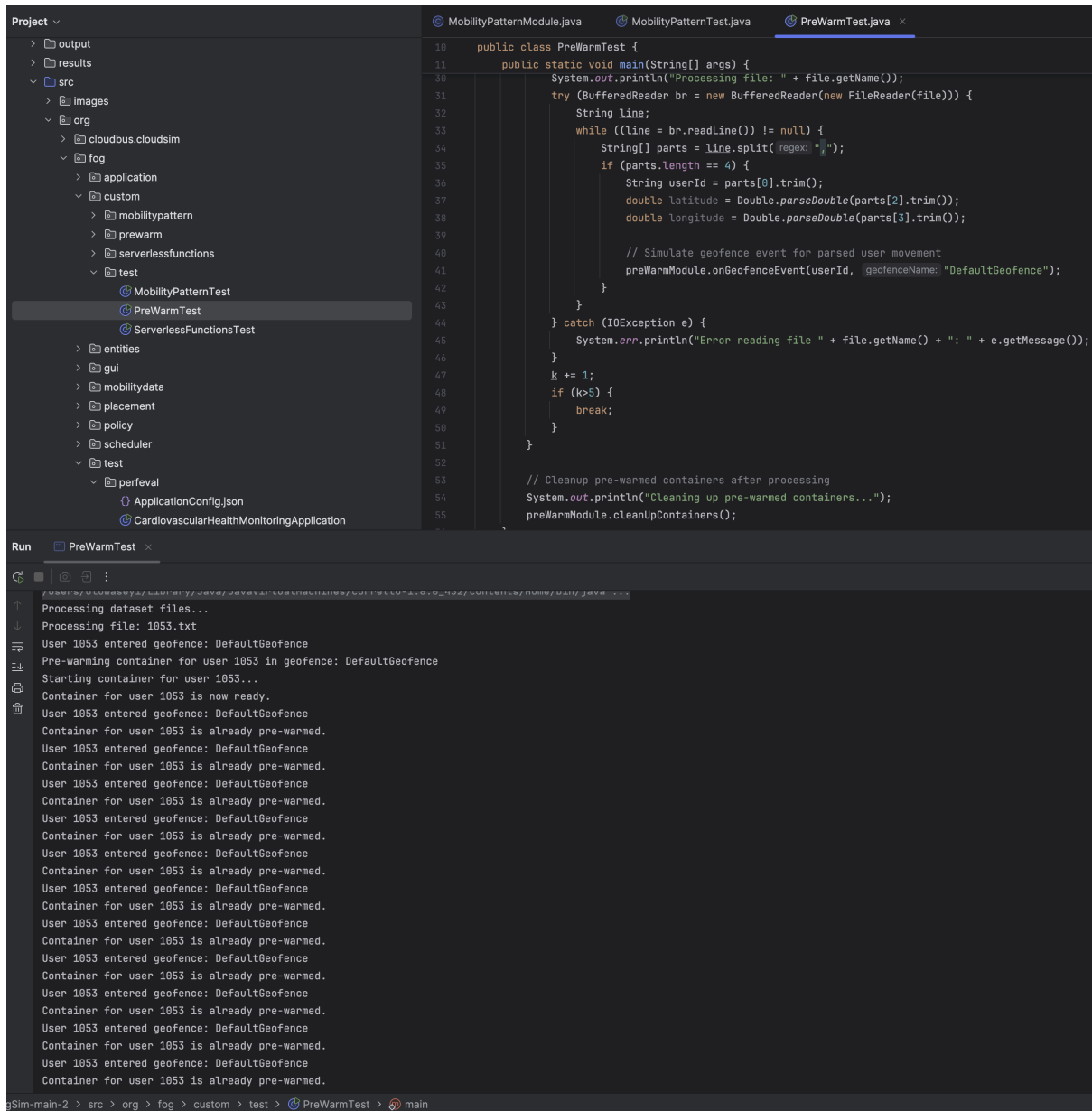


Figure 4: Prewarm module of the Algorithm

- Run the complete simulation of serverless functions under both vanilla and custom location-aware configurations navigating to the `src > org > fog > custom > test > ServerlessFunctionsTest`. It generates the performance metrics across peak and non-peak hours for cold start latency, execution time, and resource consumption which are exported and analyzed in the next section

```

24 public class ServerlessFunctionsTest {
25     public static void main(String[] args) {
26
27         // Load Data from Dataset
28         String invocationsFile = "dataset/Processed1/processed_invocations_per_function.csv";
29         String durationsFile = "dataset/Processed1/processed_function_durations.csv";
30         String memoryFile = "dataset/Processed1/processed_app_memory.csv";
31         Map<String, FunctionProfile> functionProfiles = loadFunctionProfiles(invocationsFile, durationsFile, memoryFile);
32
33         // Create Fog Devices
34         List<FogDevice> fogDevices = createFogDevices();
35
36         // Create Application
37         Application application = createApplication(appId: "ServerlessApp", userId: 1);
38
39         // Initialize ExecuteServerlessFunctionsModule
40         ExecuteServerlessFunctionsModule module = new ExecuteServerlessFunctionsModule(
41             functionProfiles,
42             loadInvocationPatterns(invocationsFile),
43             fogDevices
44         );
45
46         // Simulate all and print final metrics
47         System.out.println("Testing Vanilla Configuration for All Functions:");
48         module.simulateAndPrintAggregateMetrics(application, optimized: false);
49
50         System.out.println("Testing Custom Configuration for All Functions:");
51         module.simulateAndPrintAggregateMetrics(application, optimized: true);
52
53         } catch (Exception e) {
54             e.printStackTrace();
55             System.out.println("Simulation terminated due to unexpected error.");
56         }
57     }
58
59     // Create Fog Devices
60     private static List<FogDevice> createFogDevices() {
61         List<FogDevice> fogDevices = new ArrayList<>();
62
63         try {
64             fogDevices.add(createFogDevice(name: "Cloud", mips: 10000, ram: 40000, upBw: 1000, downBw: 1000, storage: 1000000, ratePerMbps: 1000000));
65             fogDevices.add(createFogDevice(name: "Gateway", mips: 4000, ram: 10000, upBw: 500, downBw: 500, storage: 500000, ratePerMbps: 500000));
66             fogDevices.add(createFogDevice(name: "Edge", mips: 2000, ram: 4000, upBw: 200, downBw: 200, storage: 200000, ratePerMbps: 200000));
67         } catch (Exception e) {
68             e.printStackTrace();
69         }
70
71         System.out.println("Fog Devices Created.");
72         return fogDevices;
73     }
74 }

```

```

Run ServerlessFunctionsTest
FUNCTION PROFILES LOADED.
Fog Devices Created.
Application Created.
Invocation Patterns Loaded.
Testing Vanilla Configuration for All Functions:
Simulating all functions across Peak and Non-Peak hours...
===== Aggregated Metrics for Peak Hours =====
Total Execution Time: 73851951.99 ms
Test Data Start Location: 18500000.00 ms
Sim-main-2 > src > org > fog > custom > test > ServerlessFunctionsTest > main

```

Figure 5: Simulation execution

5 Evaluating the Results

1. Results are collected through the console of the execution of the `ServerlessTestFunction` module.
2. The results are analyzed in the `analysis` notebook file using `numpy` and `matplotlib`
3. Run the `analysis` notebook file to see the analyzed results in charts



Figure 6: Result Analysis

References

Buyya, R. and Srirama, S. N. (2019). *Modeling and Simulation of Fog and Edge Computing Environments Using iFogSim Toolkit*, Wiley, pp. 433–465.

Microsoft-Research (2011). T-drive trajectory dataset. Licensed under the Microsoft Research License Agreement.

URL: <https://www.kaggle.com/datasets/arashnic/tdriver/data>

Microsoft-Research (2019). Azure functions dataset. Licensed under CC BY 4.0.

URL: <https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md>