

Configuration Manual

MSc Research Project

Cloud Computing

Sachin Reddy Chidurala

Student ID: X22140476

School of Computing

National College of Ireland

Supervisor: Dr. Punit Gupta

National College of Ireland
MSc Project Submission Sheet

School of Computing

Student Name: Sachin Reddy Chidurala
Student ID: X22140476
Programme: Cloud Computing **Year:** 2023
Module: Msc Research Project
Lecturer: Dr. Punit Gupta
Submission Due Date: 14-12-2023
Project Title: Strategic Management of Multi Cloud Adoption: A Framework for Decision-making and Governance
Word Count: 836 **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: Sachin Reddy Chidurala

Date: 14-12-2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Sachin Reddy Chidurala
Student ID: X22140476

1 Introduction

This article is a detailed guide on setting up the Cloud sim simulator for container-based simulations. It explains the tools and technology needed for the setup. The article also covers important pre-requirements and steps for using the simulator effectively. It focuses on both the technical aspects and how to use the simulator.

2 Tools/Technologies Pre-requirements

Here are some of the essential conditions/pre-requirements for the setup.

- JavaSE - 1.8
- Eclipse IDE
- Cloudsim - cloudsim-4.0.

3 Deployment Configuration

Before running simulations with the Cloudsim tool on a Mac, there are several preparatory steps to follow, particularly concerning Java installation, as Cloudsim is a Java-based application:

3.1 Installation of Java

Java is essential for running Cloudsim since it's a Java application. To create and manage your Cloudsim projects effectively, you can use Eclipse, a popular Java IDE. Eclipse is not just an IDE but also a platform for developing IDE plugins and rich client applications.

3.1.1 Using Eclipse as an IDE for Java

Eclipse is well-suited for Java development and is recommended for running simulations in Cloudsim. It provides various tools and features that make coding, debugging, and testing Java applications more manageable.

3.1.2 Downloading Java

To get started, you need to download and install the Java Development Kit (JDK). You can download JDK from the following URL:

- Java Download Link: <https://www.oracle.com/java/technologies/downloads/>

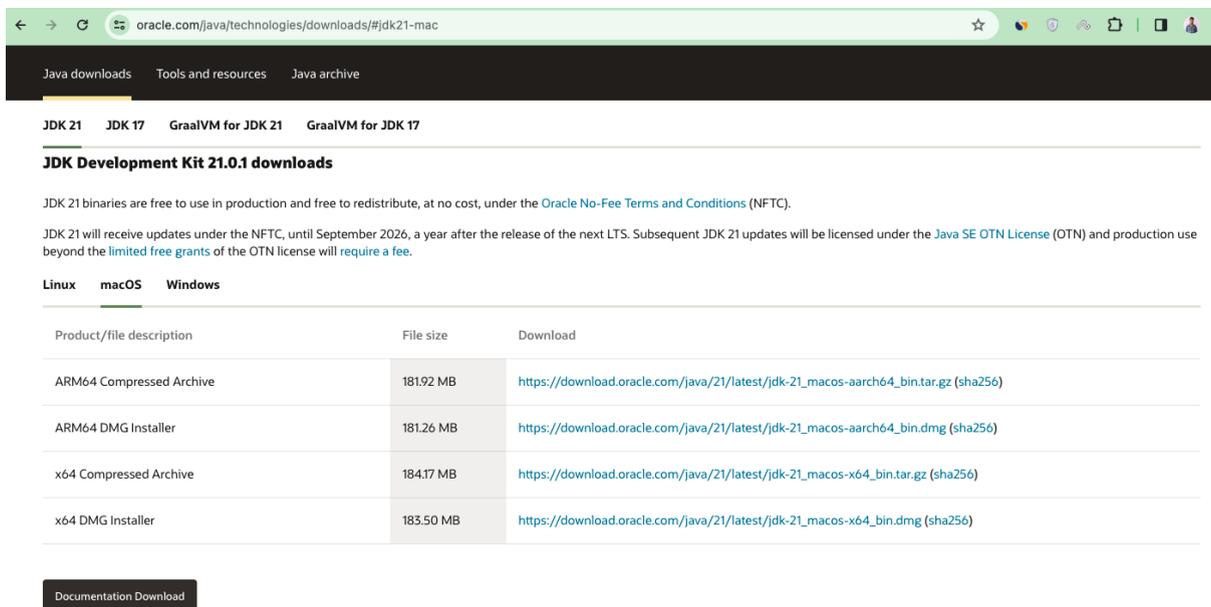


Figure 1: Java JDK download

After downloading the JDK file, Install the JDK file.



Figure 2: Java JDK Installation

After Installing Java, Open the terminal and check for the Java version by typing `java -version` in the terminal.

A screenshot of a macOS terminal window. The title bar shows the user 'sachinreddychidurala' and the shell '-bash' with a window size of '80x24'. The terminal output shows the last login time, a message about switching to zsh, and the command 'java -version' which returns the version '21.0.1' 2023-10-17 LTS, along with details about the Java(TM) SE Runtime Environment and HotSpot(TM) 64-Bit Server VM.

```
sachinreddychidurala — -bash — 80x24
Last login: Mon Dec 11 12:54:53 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[Sachins-MacBook-Air:~ sachinreddychidurala$ java -version ]
java version "21.0.1" 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 21.0.1+12-LTS-29)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.1+12-LTS-29, mixed mode, sharing)
Sachins-MacBook-Air:~ sachinreddychidurala$ █
```

Figure 3: Java version check in the terminal

Once Java is installed on the Mac, proceed to set up Eclipse and configure it for the Cloudsim projects. Ensure that the Java version downloaded is compatible with Cloudsim and the Eclipse version.

3.2 Installation of Eclipse IDE

To use Eclipse on the Mac OS, the system must meet certain criteria.

Memory	Processor	File Size:	OS
4GB	Intel Core, Apple Silicon	30 GB	MacOS 10.9 or later

Table 1: System Requirements for Eclipse IDE

Downloading Eclipse:

- Visit the Eclipse download page (<https://www.eclipse.org/downloads/>).
- Choose the Eclipse IDE suitable for your development needs (e.g., Eclipse IDE for Java Developers).
- Download the macOS version.



Figure 4: Eclipse IDE for Java Developers download

After downloading the eclipse, unzip the file and install the file. When trying to Launch it for the first time the Mac will ask “are you sure you want to open eclipse” since it is downloaded from the internet. Press Open.

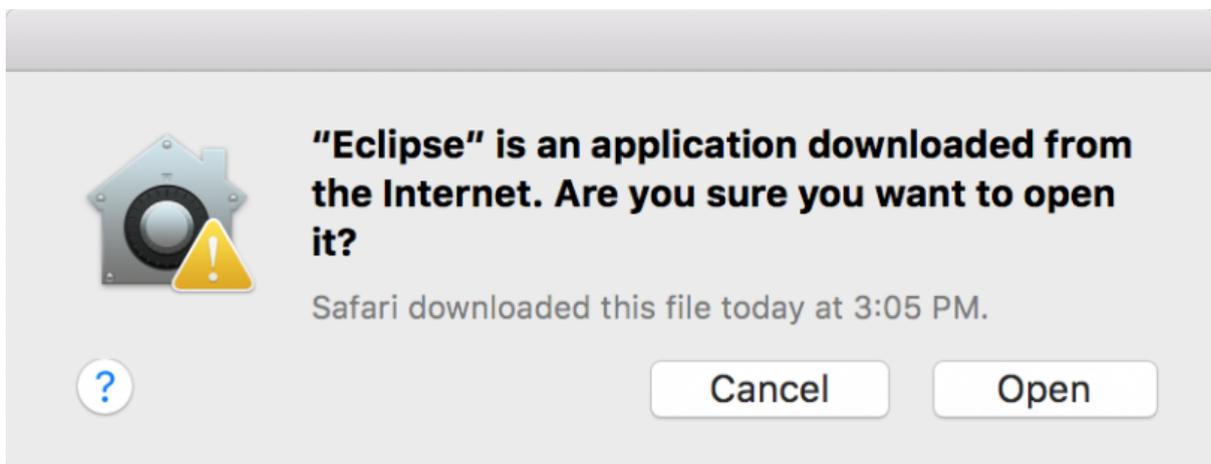


Figure 5: Eclipse IDE popup Window

After Opening the eclipse installer, select “Eclipse IDE for Java Developers”.

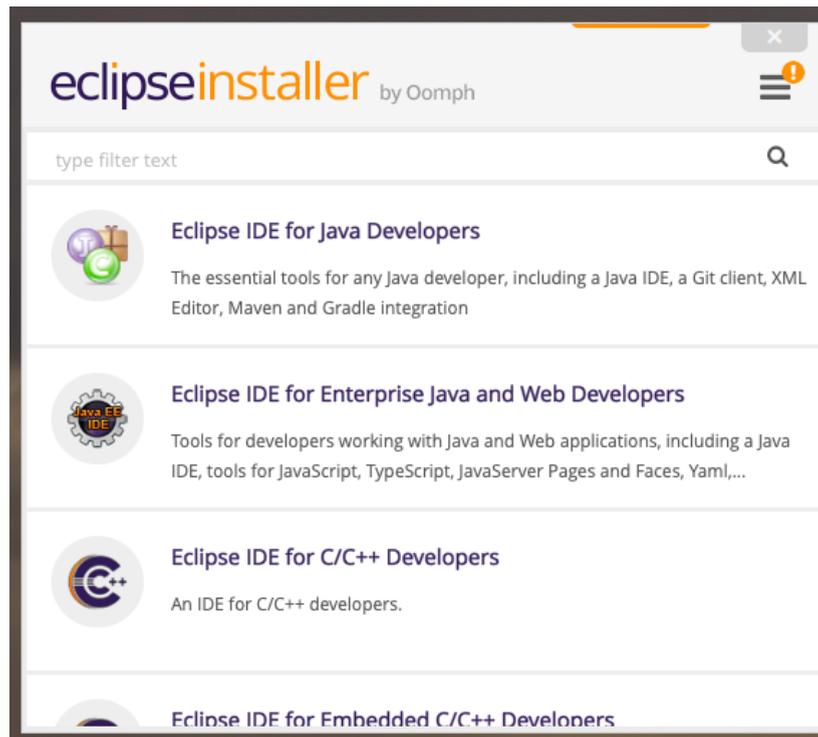


Figure 6: Eclipse installer

In the next window, from the drop down select the Java JDK and give the installation folder and enter "Install".

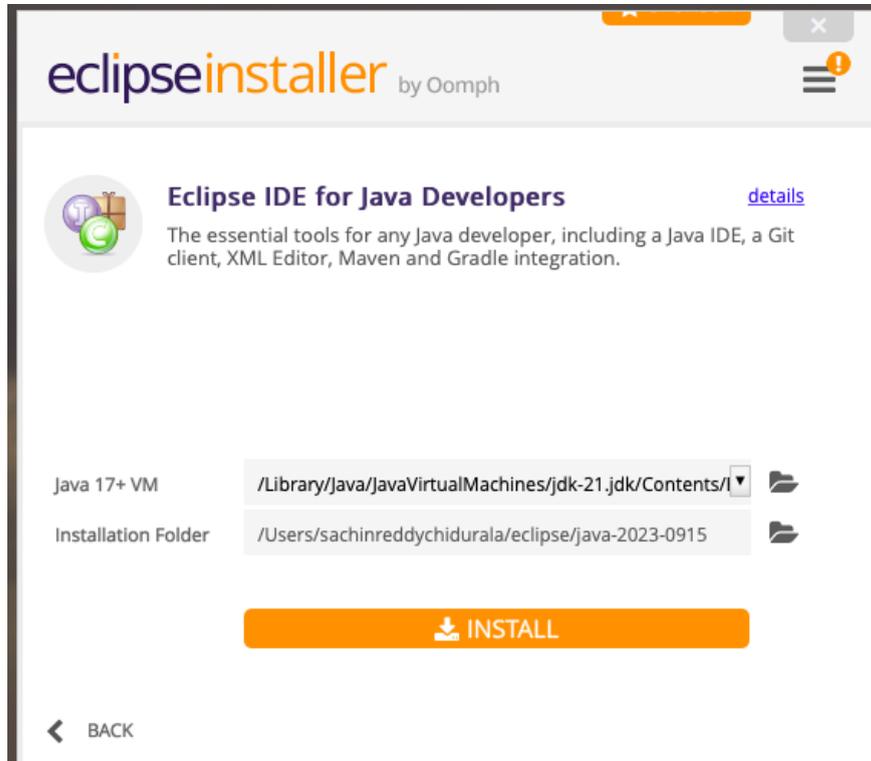


Figure 7: Eclipse Installer Installation

Now Eclipse Installation is done and a welcome pop window will display.



Figure 8: Eclipse Welcome Popup window

3.3 CloudSim Installation

3.3.1 Downloading Artifact (Cloudsim-4.0):

Download the cloudsim-4.0 zip file from Github using the link and Unzip the file. (<https://github.com/Cloudslab/cloudsim/releases>).

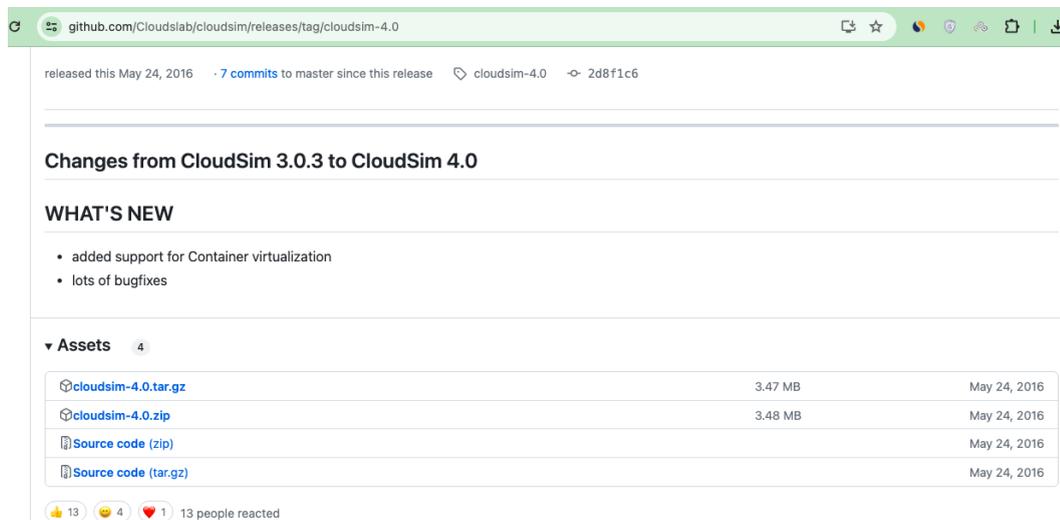


Figure 9: Downloading Cloudsim from Github

Now Open the Eclipse and just go to file and select import, then select Maven and from the dropdown choose the existing Maven project.

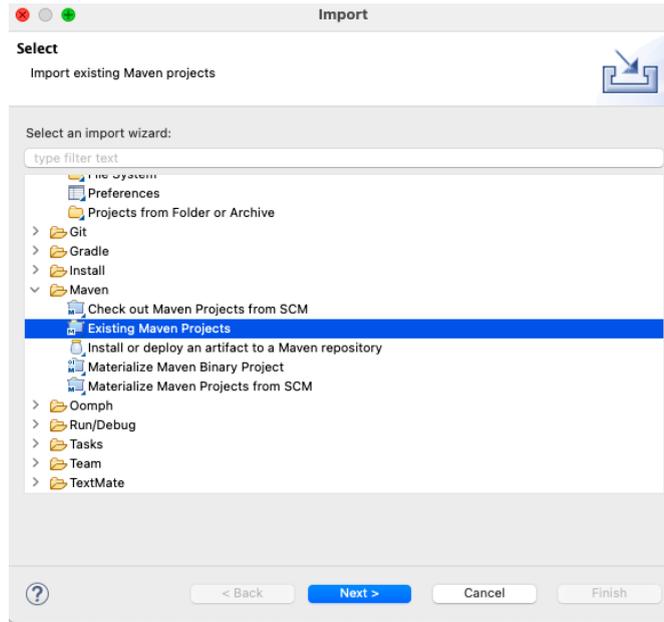


Figure 10: Importing Cloudsim from Existing Maven Projects

Next in the Root directory browse the Cloudsim Zip file and click finish. Make sure to mark all the boxes.

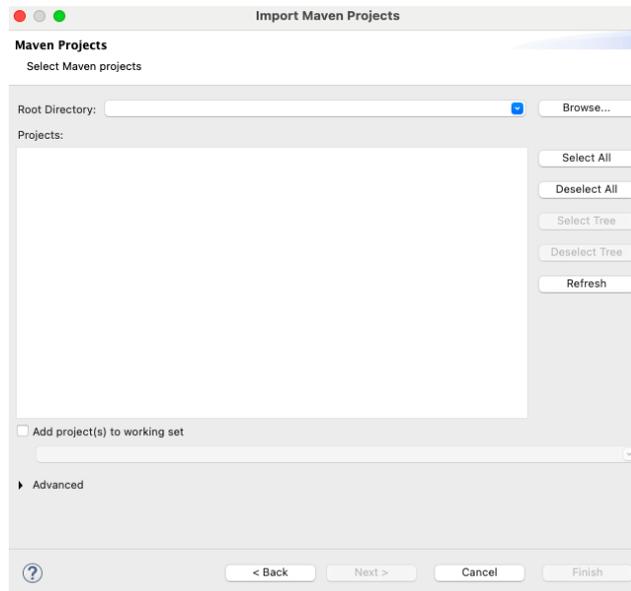


Figure 11: Browsing Cloudsim from local

Just wait for the build process to finish and now Cloudsim is imported into Eclipse. After the build process is completed the Eclipse IDE Looks like the below image.

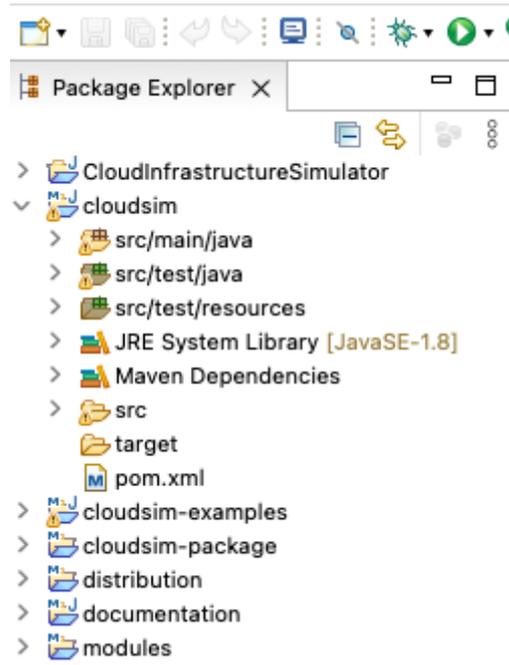


Figure 12: Cloudsim Package Explorer

Now right-click on the cloudsim- example folder and select Build Path and choose Java Build Path and add External Jars using Add external archive.

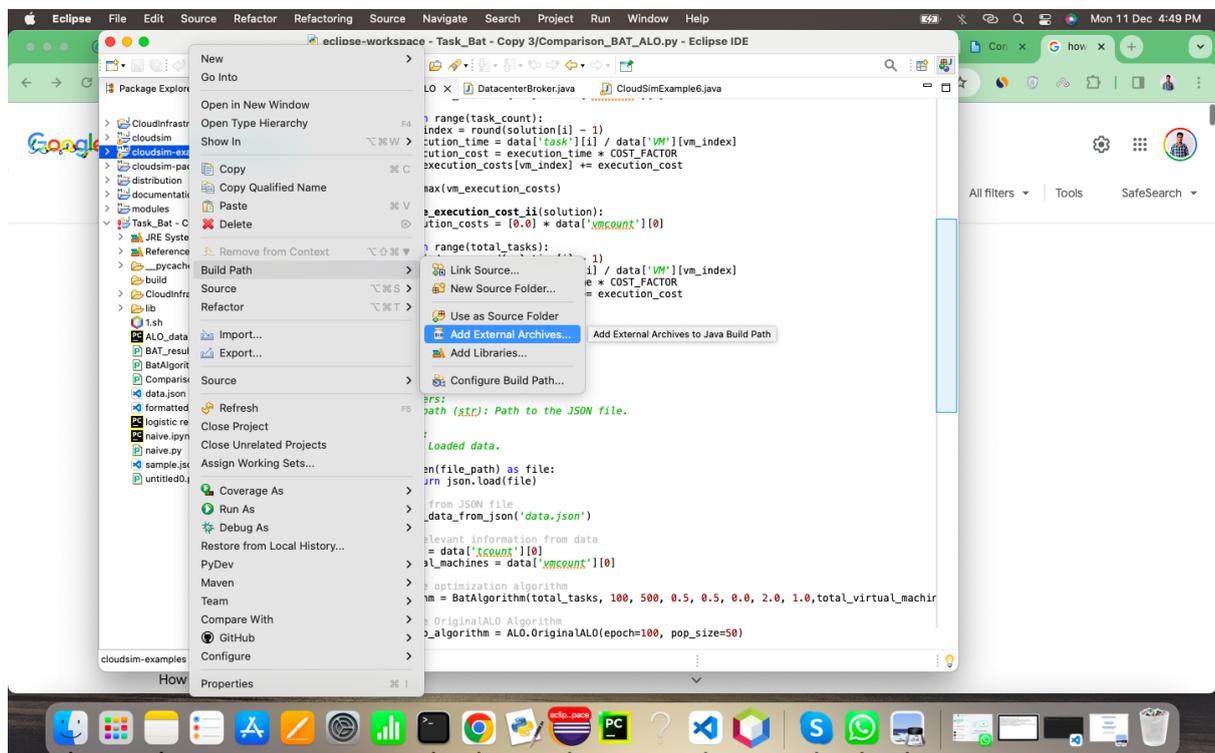


Figure 13: Adding Jar files in the java Build path

Now choose the Jars from the cloudsim zip file.

Name	Date Modified	Size	Kind
cloudsim-4.0.jar	24-May-2016 at 1:34 PM	441 KB	Java JAR file
cloudsim-examples-4.0.jar	24-May-2016 at 1:34 PM	5.1 MB	Java JAR file

Figure 14: Cloudsim Jar Files in the system

4. Validations

Now will run the cloudsim example 6.

```

283 /**
284  * Prints the Cloudlet objects
285  * @param list list of Cloudlets
286  */
287 private static void printCloudletList(List<Cloudlet> list) {
288     int size = list.size();
289     Cloudlet cloudlet;
290
291     String indent = "  ";
292     Log.println();
293     Log.println("===== OUTPUT =====");
294     Log.println("Cloudlet ID" + indent + "STATUS" + indent +
295               "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time"
296               );
297     DecimalFormat dft = new DecimalFormat("###.##");
298     for (int i = 0; i < size; i++) {

```

```

<terminated> CloudSimExample6 (1) [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (11-Dec-2023
11 SUCCESS 3 11 3 0.2 3.2
23 SUCCESS 3 11 3 0.2 3.2
35 SUCCESS 3 11 3 0.2 3.2
0 SUCCESS 2 0 4 0.2 4.2
12 SUCCESS 2 0 4 0.2 4.2
24 SUCCESS 2 0 4 0.2 4.2
36 SUCCESS 2 0 4 0.2 4.2
1 SUCCESS 2 1 4 0.2 4.2
13 SUCCESS 2 1 4 0.2 4.2
25 SUCCESS 2 1 4 0.2 4.2
37 SUCCESS 2 1 4 0.2 4.2
2 SUCCESS 2 2 4 0.2 4.2
14 SUCCESS 2 2 4 0.2 4.2
26 SUCCESS 2 2 4 0.2 4.2
38 SUCCESS 2 2 4 0.2 4.2
3 SUCCESS 2 3 4 0.2 4.2
15 SUCCESS 2 3 4 0.2 4.2
27 SUCCESS 2 3 4 0.2 4.2
39 SUCCESS 2 3 4 0.2 4.2
CloudSimExample6 finished!

```

Figure 15: Cloudsim Example 6

The validations have been performed using Cloudsim and algorithms like BAT and ALO.

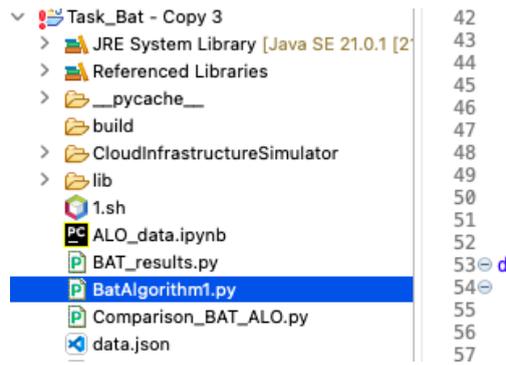


Figure 16: Imported BAT algorithm

From the data.json file the code will take the input with number of VMS, VMS capacity and Task count.

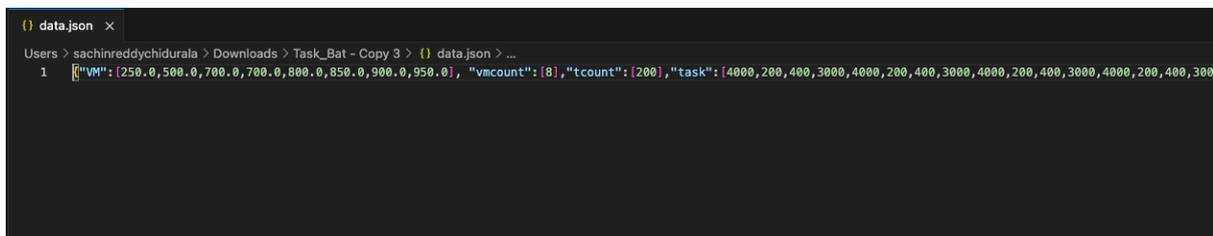


Figure 17: Data.json code where it shows VM count, VM capacity, Task Count

Also have implemented the BAT algorithm to find the execution cost and Run time for different tasks 100, 200, and 500. Below is the screenshot of the BAT optimization code.

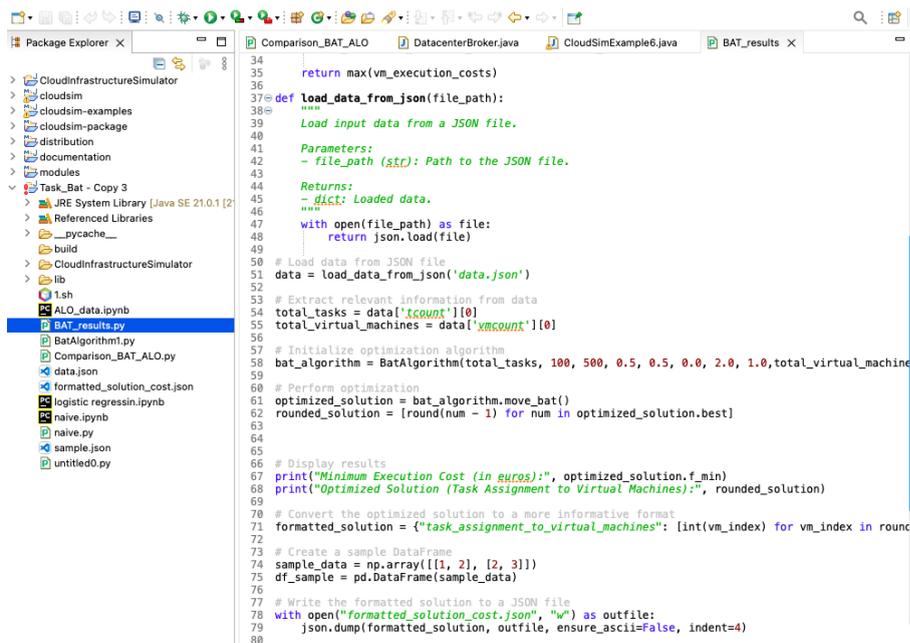


Figure 18: BAT algorithm Optimization code snippet.

Below are the execution cost and runtime results of the BAT optimization technique.

```

1 import random
2 from BatAlgorithm1 import BatAlgorithm
3 import json
4 import numpy as np
5 import pandas as pd
6 from sklearn.model_selection import train_test_split
7 from sklearn.naive_bayes import GaussianNB
8 from sklearn.metrics import accuracy_score
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 # Conversion factor for execution time to cost (0.00015 euros per second)
13 COST_FACTOR = 0.00015
14
15 def optimize_execution_cost(task_count, solution):
16     """
17     Objective function to optimize the execution cost of tasks on virtual machines.
18
19     Parameters:
20     - task_count (int): Total number of tasks.
21     - solution (list): Solution vector representing the assignment of tasks to virtual machines.
22     - data (dict): Input data.
23
24     Returns:
25     - float: Maximum execution cost among all virtual machines.
26     """
27     vm_execution_costs = [0.0] * data['vmcount'][0]
28
29     for i in range(task_count):
30         vm_index = round(solution[i] - 1)
31         execution_time = data['task'][i] / data['VM'][vm_index]

```

Console Output:

```

<terminated> BAT_results.py [usr/local/bin/python]
Minimum Execution Cost (in euros): 0.01084285714285714
Optimized Solution (Task Assignment to Virtual Machines): [0, 1, 1, 6, 3, 6, 6, 4, 6, 6, 6, 5, 6, 7, 7, 0

```

Figure 19: Simulation result for BAT algorithm with Execution cost and Task Assignment

Also I have done comparison of BAT results with the ALO algorithms results by importing the ALO algorithm using Mealy library.

```

1 from mealy import FloatVar, ALO
2 import random
3 import json
4 import time
5 from BatAlgorithm1 import BatAlgorithm
6
7 import numpy as np
8 import pandas as pd
9 from sklearn.model_selection import train_test_split
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.metrics import accuracy_score
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14 from mealy import FloatVar
15
16 # Conversion factor for execution time to cost (0.00015 euros per second)
17 COST_FACTOR = 0.00015
18
19 def optimize_execution_cost(task_count, solution):
20     """
21     Objective function to optimize the execution cost of tasks on virtual machines.
22
23     Parameters:
24     - task_count (int): Total number of tasks.
25     - solution (list): Solution vector representing the assignment of tasks to virtual machines.
26     - data (dict): Input data.
27
28     Returns:
29     - float: Maximum execution cost among all virtual machines.
30     """
31     vm_execution_costs = [0.0] * data['vmcount'][0]
32
33     for i in range(task_count):
34         vm_index = round(solution[i] - 1)
35         execution_time = data['task'][i] / data['VM'][vm_index]
36         execution_cost = execution_time * COST_FACTOR
37         vm_execution_costs[vm_index] += execution_cost
38
39     return max(vm_execution_costs)
40

```

Console Output:

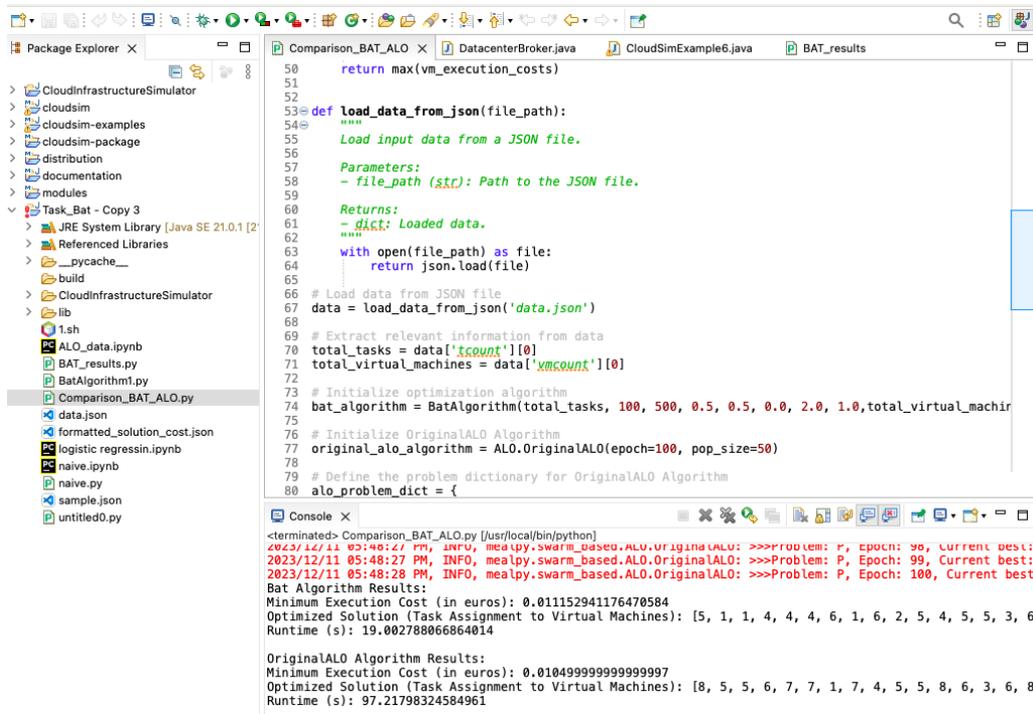
```

<terminated> BAT_results.py [usr/local/bin/python]
Minimum Execution Cost (in euros): 0.01084285714285714
Optimized Solution (Task Assignment to Virtual Machines): [0, 1, 1, 6, 3, 6, 6, 4, 6, 6, 6, 5, 6, 7, 7, 0

```

Figure 20: Comparison code snippet of both BAT and ALO

Here are the execution cost and runtime comparison results of both the BAT and ALO algorithm.



```

50     return max(vm_execution_costs)
51
52
53 def load_data_from_json(file_path):
54     """
55     Load input data from a JSON file.
56
57     Parameters:
58     - file_path (str): Path to the JSON file.
59
60     Returns:
61     - dict: Loaded data.
62     """
63     with open(file_path) as file:
64         return json.load(file)
65
66 # Load data from JSON file
67 data = load_data_from_json('data.json')
68
69 # Extract relevant information from data
70 total_tasks = data['tcount'][0]
71 total_virtual_machines = data['vmcount'][0]
72
73 # Initialize optimization algorithm
74 bat_algorithm = BatAlgorithm(total_tasks, 100, 500, 0.5, 0.5, 0.0, 2.0, 1.0, total_virtual_machin
75
76 # Initialize OriginalALO Algorithm
77 original_alo_algorithm = ALO.OriginalALO(epoch=100, pop_size=50)
78
79 # Define the problem dictionary for OriginalALO Algorithm
80 alo_problem_dict = {

```

```

<terminated> Comparison_BAT_ALO.py [Jusr/local/bin/python]
2023/12/11 05:48:27 PM, INFO, mealpy.swarm_based.ALO.OriginalALO: >>>Problem: P, Epoch: 98, Current best:
2023/12/11 05:48:27 PM, INFO, mealpy.swarm_based.ALO.OriginalALO: >>>Problem: P, Epoch: 99, Current best:
2023/12/11 05:48:28 PM, INFO, mealpy.swarm_based.ALO.OriginalALO: >>>Problem: P, Epoch: 100, Current best
Bat Algorithm Results:
Minimum Execution Cost (in euros): 0.011152941176470584
Optimized Solution (Task Assignment to Virtual Machines): [5, 1, 1, 4, 4, 4, 6, 1, 6, 2, 5, 4, 5, 5, 3, 6,
Runtime (s): 19.002788066864014

OriginalALO Algorithm Results:
Minimum Execution Cost (in euros): 0.010499999999999997
Optimized Solution (Task Assignment to Virtual Machines): [8, 5, 5, 6, 7, 7, 1, 7, 4, 5, 5, 8, 6, 3, 6, 8,
Runtime (s): 97.21798324584961

```

Figure 21: Execution cost and runtime comparison of both BAT and ALO algorithm