Configuration Manual

MSc Research Project
MSCCLOUD1_A

Kurian George
Student ID: X22191437

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Kurian George

Student ID: ……x22191437

Programme: ……MSCCLOUD1_A **Year:** 2023-24

Module: ……Research Project

Lecturer: ……Punit Gupta

Submission Due Date:

……12-08-2024……..……

Project Title: Optimizing Resource Scheduling in Cloud Environments with Docker Containers and Advanced Auto Scaling Algorithms

Word Count: ……6074…… **Page Count:** ……21……..……

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: ……Kurian George

Date: ……12-08-2024……

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

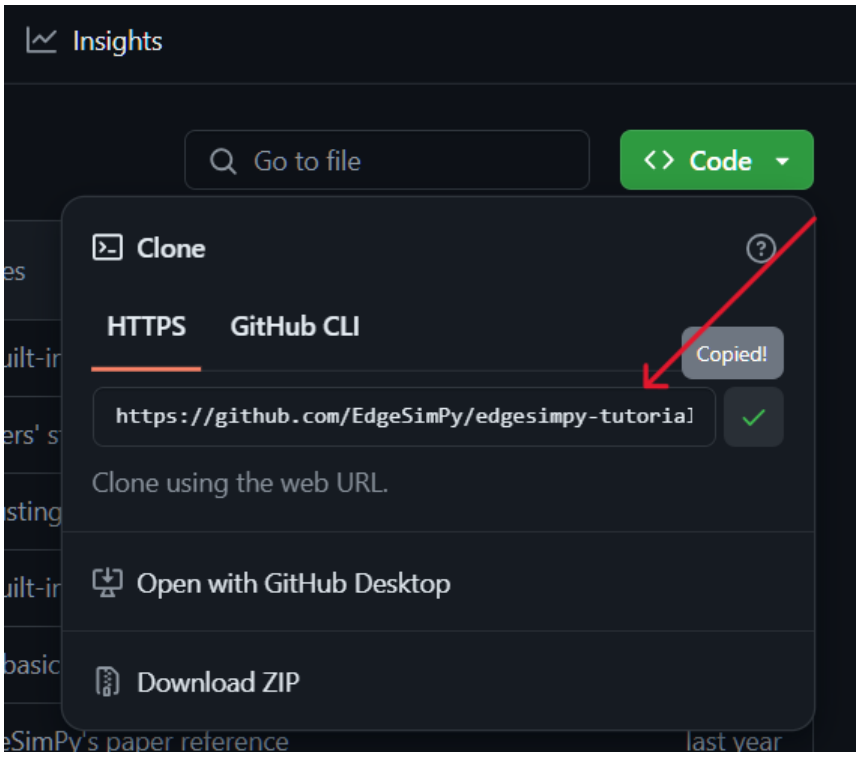
Kurian George
Student ID: x22191437

1 Setting Up EdgeSimPy in Google Colab

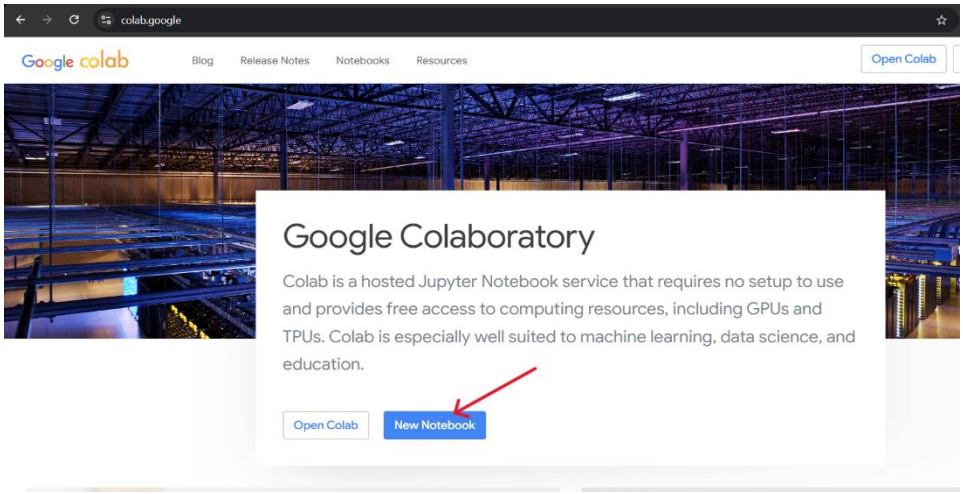
Step 1: Login into GitHub and search EdgeSimPy-tutorials or paste the below URL in Google https://github.com/EdgeSimPy/edgesimpy-tutorials

Step 2: Clone the repository by clicking the Code > and copying the HTTPS URL or copy the below URL:

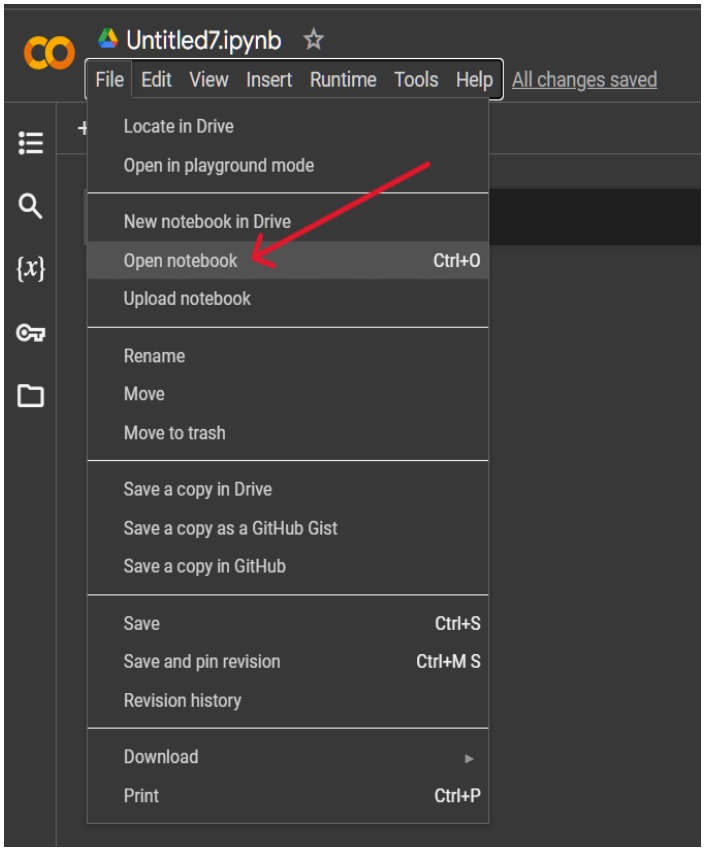
https://github.com/EdgeSimPy/edgesimpy-tutorials.git



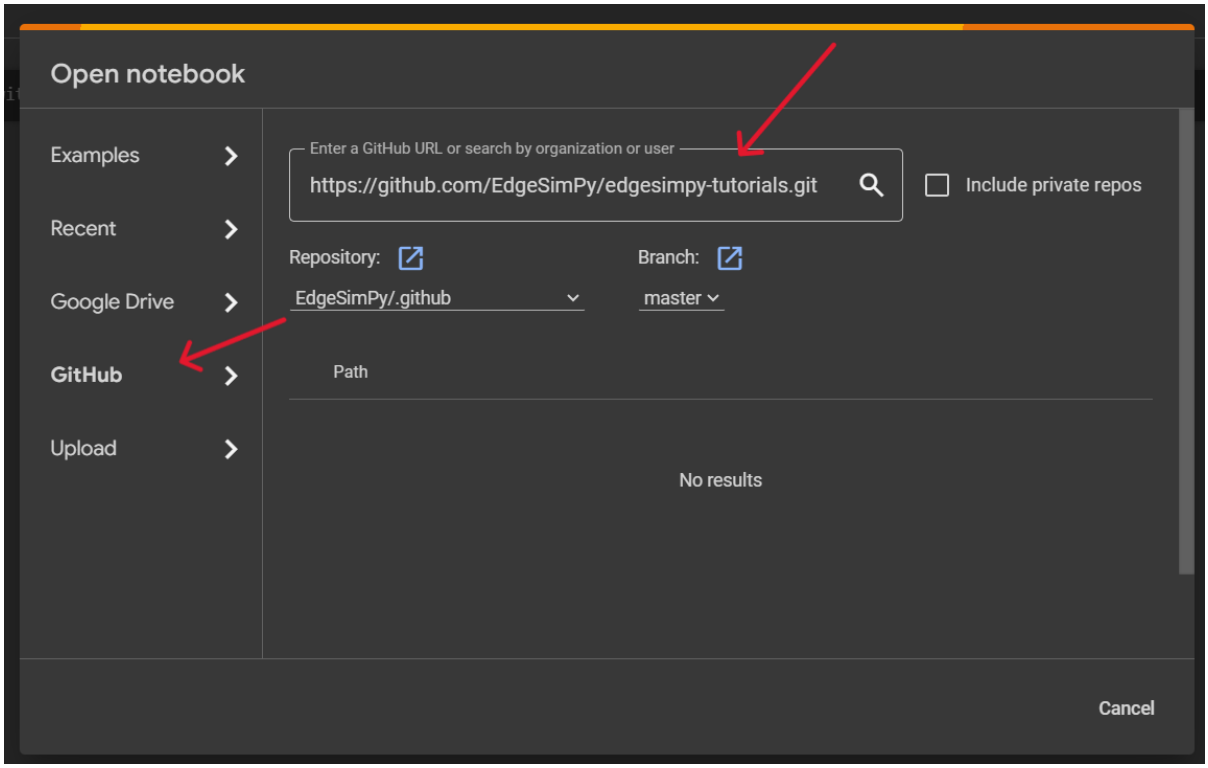
Step 3: Login into Google Colab and Select ‘New Notebook’ (You’ll probably need a gmail ID to login to the console.



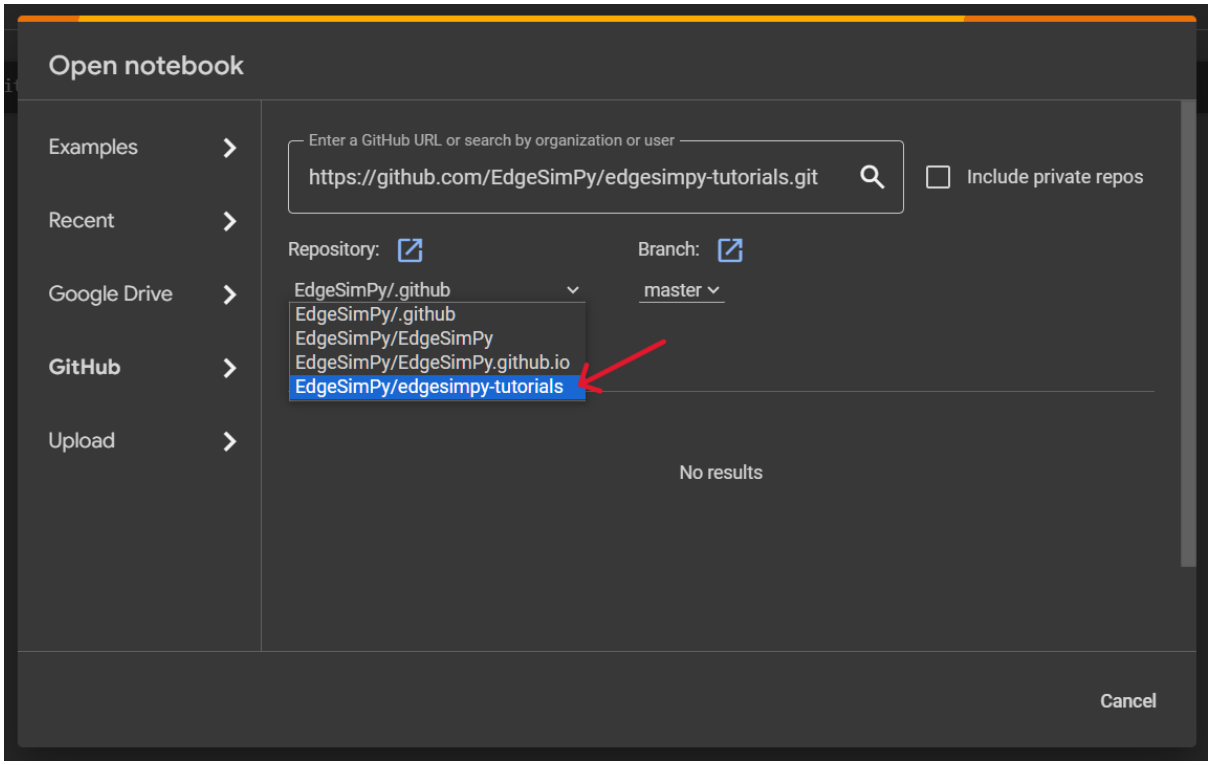
Step 4: After opening a New Notebook, select 'File' from the top left and Click on 'Open Notebook' or press 'Ctrl+O'.



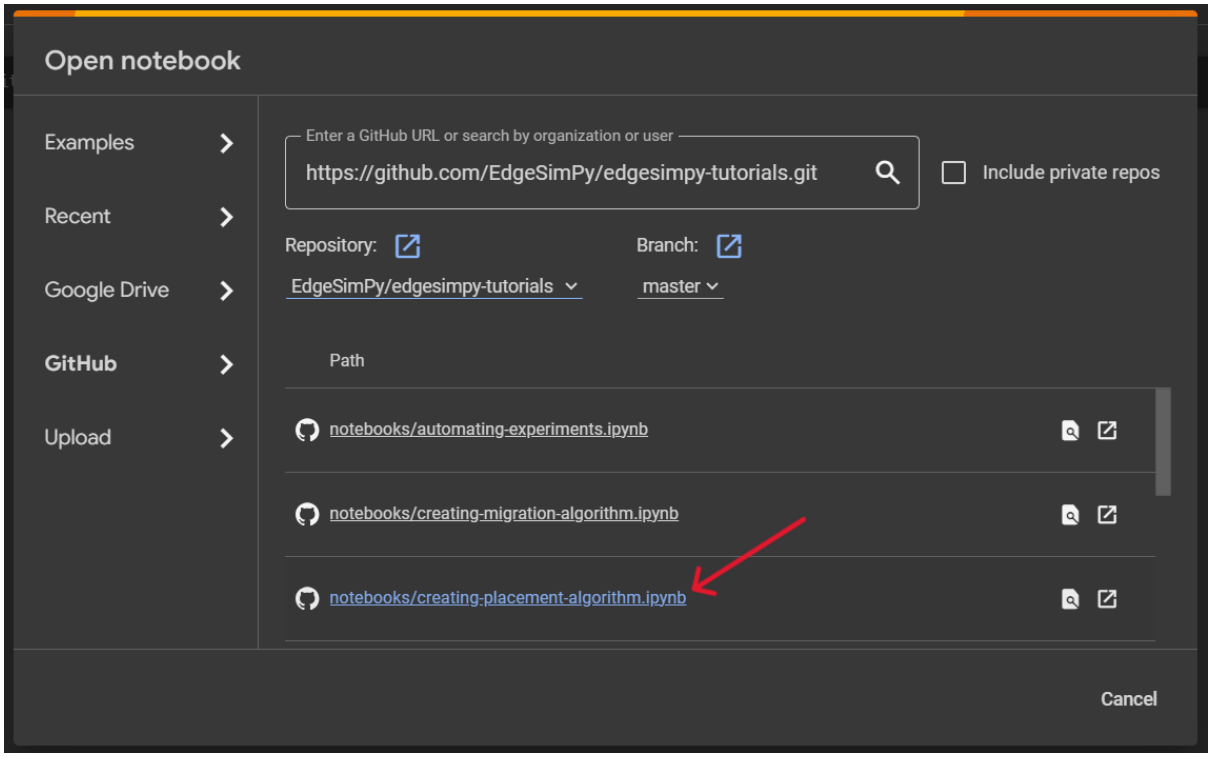
Step 5: In the Open Notebook tab, click on GitHub and paste the HTTPS URL (https://github.com/EdgeSimPy/edgesimpy-tutorials.git) in the search box.



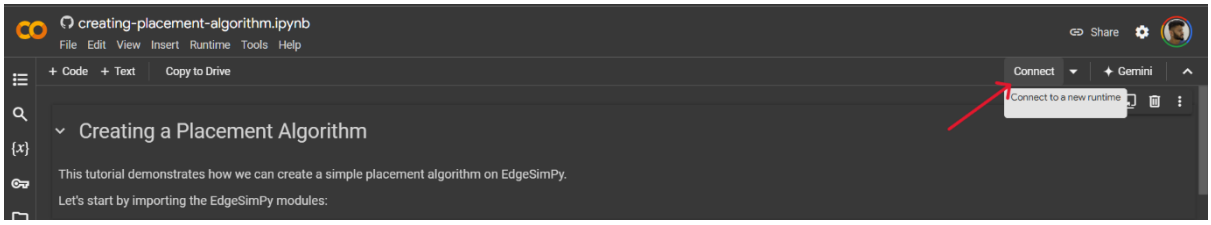
Step 6: In the Open notebook tab, under 'Repository', click on the drop-down arrow and select 'EdgeSimPy/edgesimpy-tutorial'.

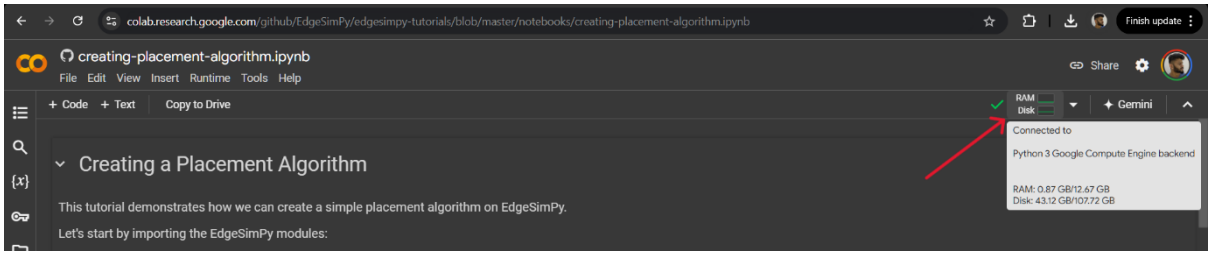


Step 7: In the Open notebook tab, under the 'Path' tab, select the 'notebooks/creating-placement-algorithm.ipynb' which will open the EdgeSimPy Placement Algorithm Workspace.



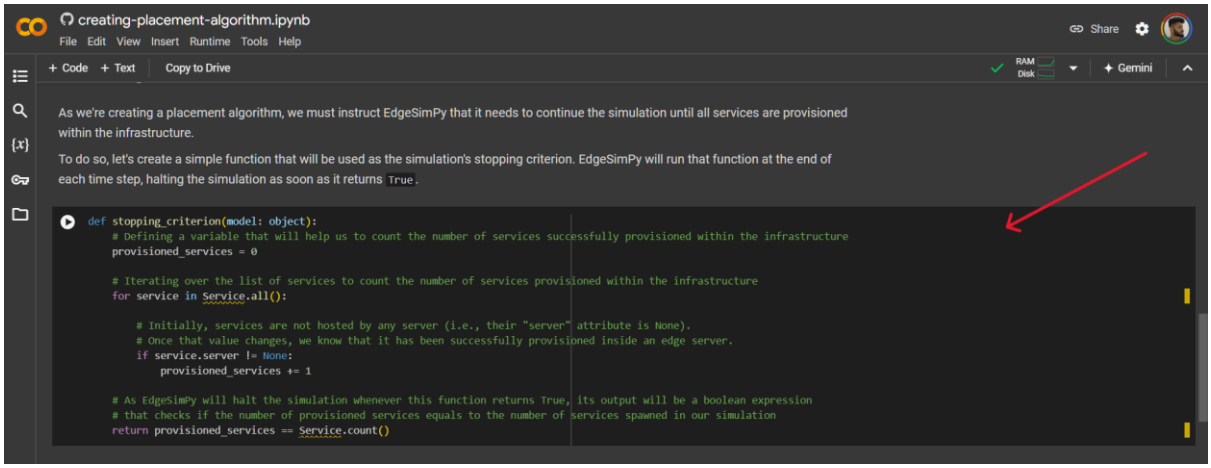
Step 8: On the Placement Algorithm Workspace, click on ‘Connect’ option at the top right corner of the page and it should change to ‘connected’ and display the RAM and Disk Symbols.





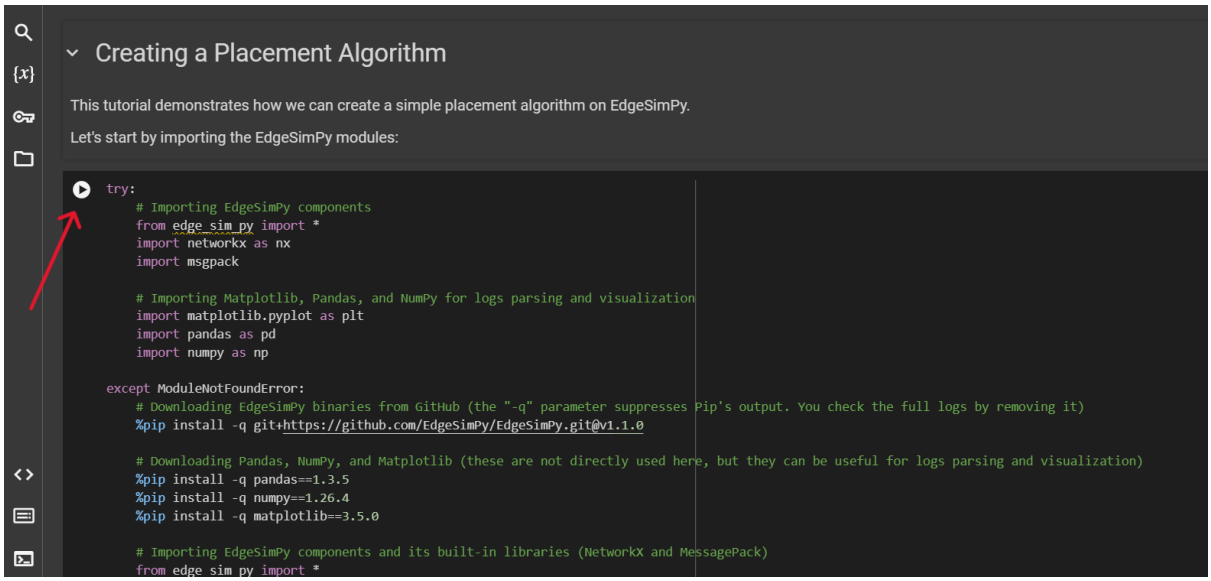
2 EdgeSimPy Workspace

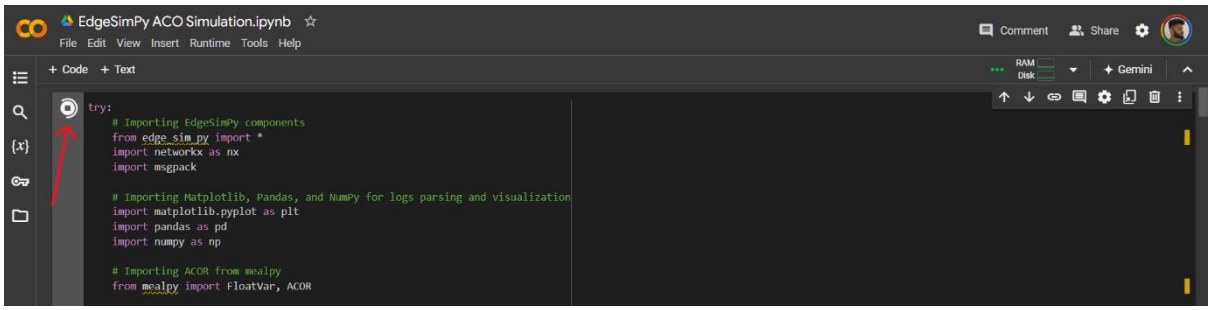
On the EdgeSimPy Placement Algorithm Workspace there will be 4 Code Cells. An example of a code cell is shown below.



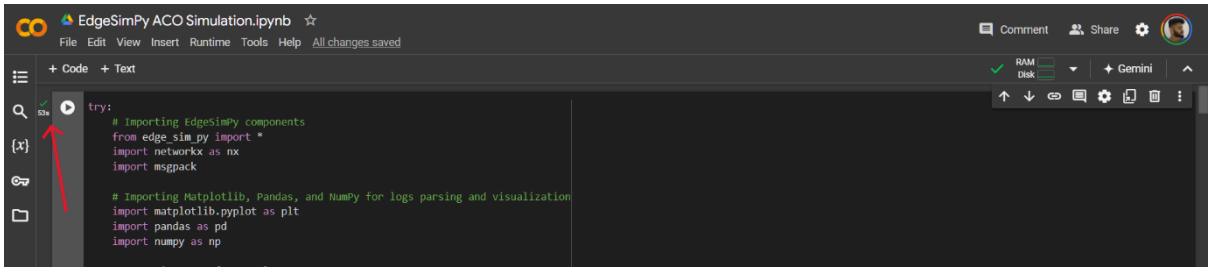
Step 1: The first Code Cell under the ‘Creating a Placement Algorithm’ heading is the Imports and setup Code Cell. This code cell will install all the dependencies required for the EdgeSimPy to function.

Step 2: To install all the dependencies, run this Code cell by pressing the ‘Run’ icon on the top left side of the Code cell.

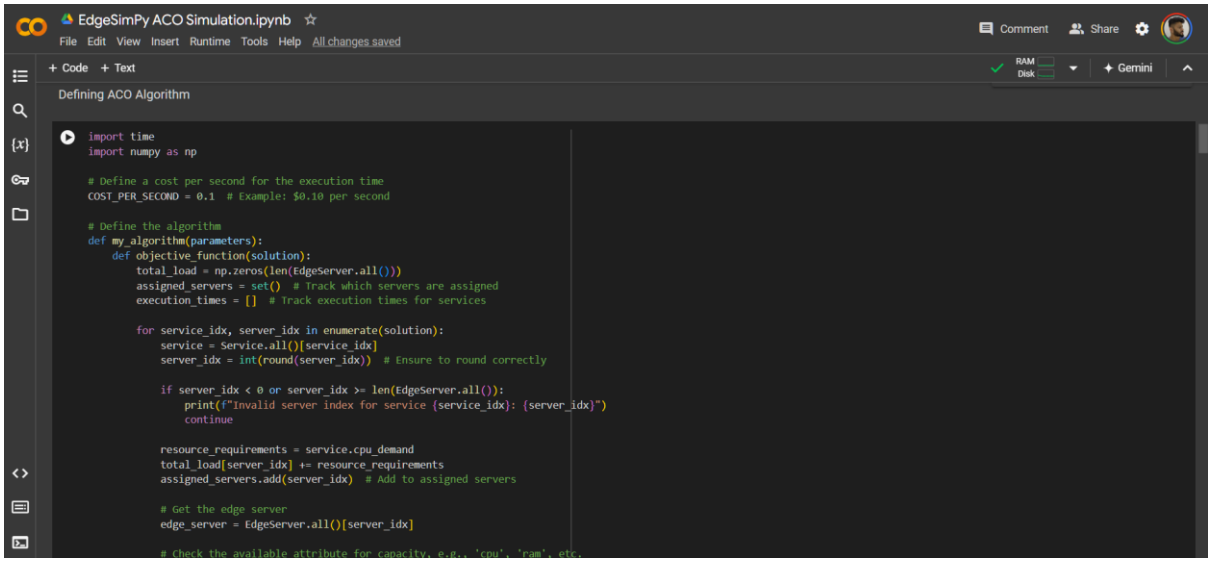




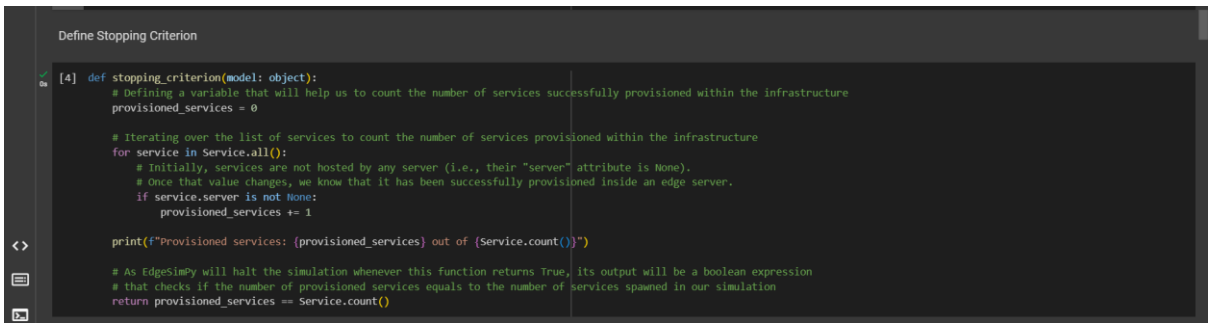
- Upon successful Running, it will display a Green Tick on the left side of the Run button



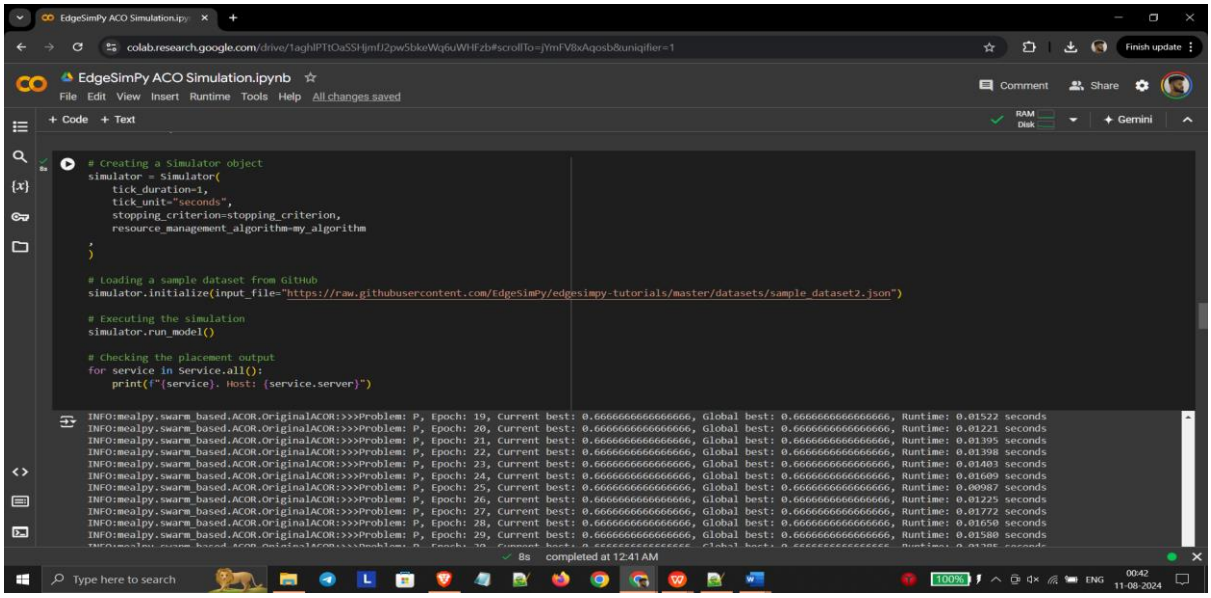
Step 3: The Second Code Cell is the ‘Placement Algorithm cell’; it is where the proposed code is implemented.



Step 4: The third Code Cell is the Stopping Criterion Code Cell which defines the stopping criteria for the iterations in the code.

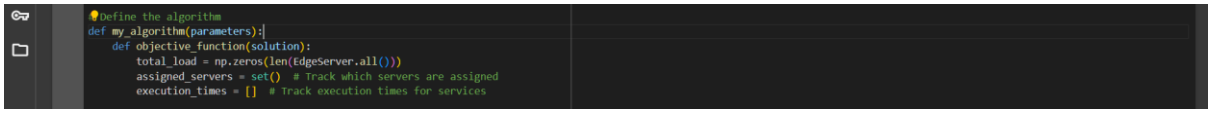


Step 5: The fourth Code Cell is used to run the simulation which will display the output of the code.

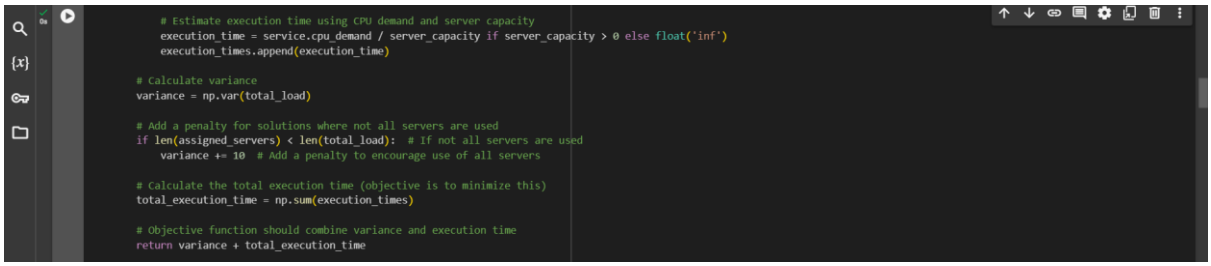


3 Code and Output

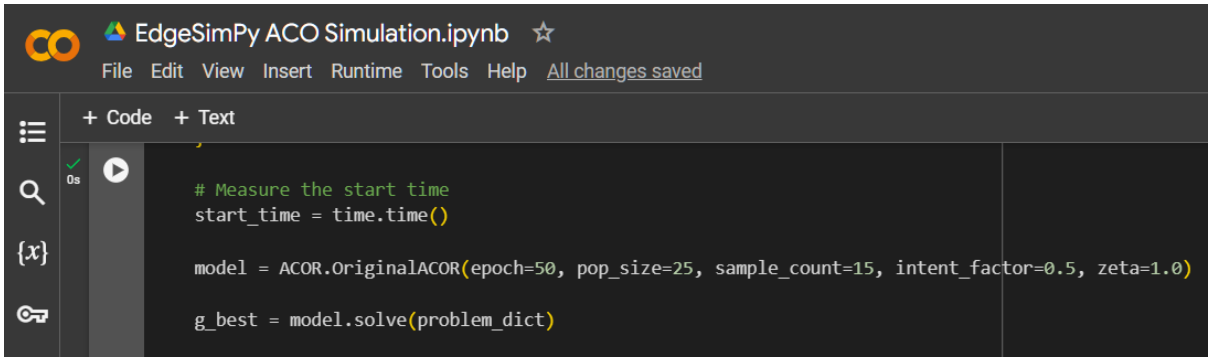
The proposed Ant Colony Optimization Code is written in Python language. It is having a main function ‘my_algorithm(parameters)’ and has a nested function ‘objective_function(solution)’ which contains all the logics and commands to print the output.



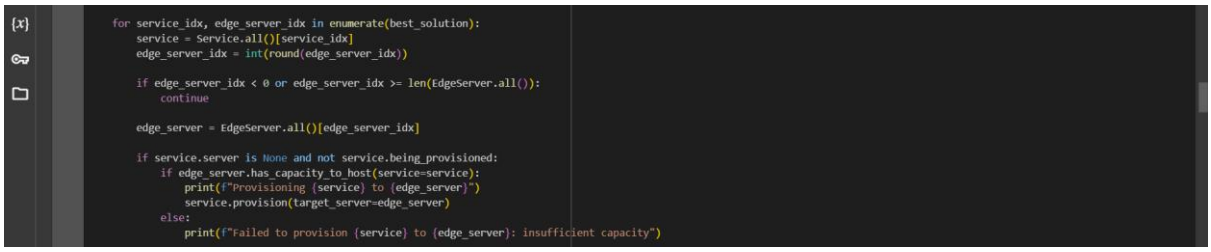
The logic to calculate the execution time using CPU demand and server capacity



The ACO algorithm parameters to find the solution

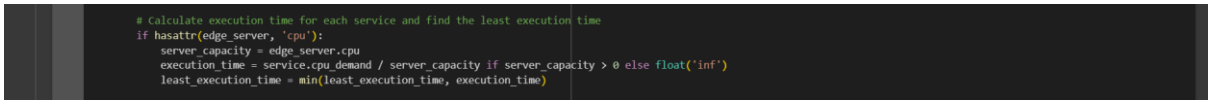


The logic to allocate servers to services.



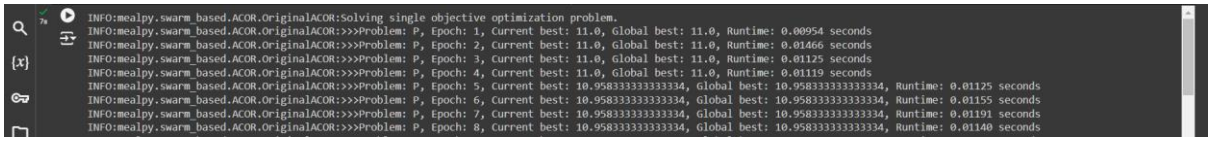
```
for service_idx, edge_server_idx in enumerate(best_solution):
    service = Service.all()[service_idx]
    edge_server_idx = int(round(edge_server_idx))

    if edge_server_idx < 0 or edge_server_idx >= len(EdgeServer.all()):
        continue

    edge_server = EdgeServer.all()[edge_server_idx]

    if service.server is None and not service.being_provisioned:
        if edge_server.has_capacity_to_host(service=service):
            print(f"Provisioning {service} to {edge_server}")
            service.provision(target_server=edge_server)
        else:
            print(f"Failed to provision {service} to {edge_server}: insufficient capacity")
```

The logic to find the execution time of each service and to find the least execution time.



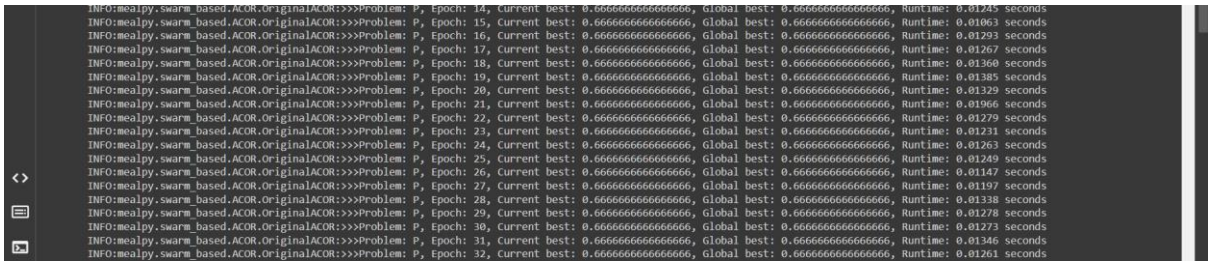
```
# Calculate execution time for each service and find the least execution time
if hasattr(edge_server, 'cpu'):
    server_capacity = edge_server.cpu
    execution_time = service.cpu_demand / server_capacity if server_capacity > 0 else float('inf')
    least_execution_time = min(least_execution_time, execution_time)
```

The output of the proposed algorithm before finding the shortest path, the current best and global best values are way higher



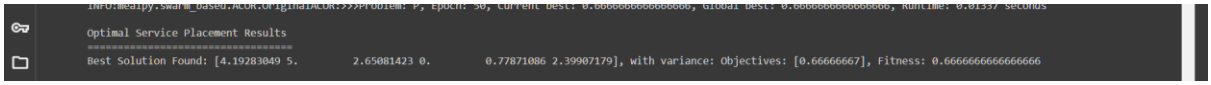
```
INFO:mealpy.swarm_based.ACOR.OriginalACOR:Solving single objective optimization problem.
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 1, Current best: 11.0, Global best: 11.0, Runtime: 0.00954 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 2, Current best: 11.0, Global best: 11.0, Runtime: 0.01466 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 3, Current best: 11.0, Global best: 11.0, Runtime: 0.01125 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 4, Current best: 11.0, Global best: 11.0, Runtime: 0.01119 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 5, Current best: 10.958333333333334, Global best: 10.958333333333334, Runtime: 0.01125 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 6, Current best: 10.958333333333334, Global best: 10.958333333333334, Runtime: 0.01155 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 7, Current best: 10.958333333333334, Global best: 10.958333333333334, Runtime: 0.01191 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 8, Current best: 10.958333333333334, Global best: 10.958333333333334, Runtime: 0.01140 seconds
```

The output of the proposed algorithm after finding the shortest path, the current best and global best values are at minimum and are constant for the rest of the iterations.



```
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 14, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01245 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 15, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01063 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 16, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01293 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 17, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01267 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 18, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01360 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 19, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01385 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 20, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01329 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 21, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01966 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 22, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01279 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 23, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01231 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 24, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01263 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 25, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01249 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 26, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01147 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 27, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01197 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 28, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01338 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 29, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01278 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 30, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01273 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 31, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01346 seconds
INFO:mealpy.swarm_based.ACOR.OriginalACOR:>>>Problem: P, Epoch: 32, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01261 seconds
```

The output showing the optimal placement results.



```
Optimal Service Placement Results
==================================
Best Solution Found: [4.19283049 5.         2.65081423 0.         0.77871086 2.39907179], with variance: Objectives: [0.66666667], Fitness: 0.6666666666666666
```

The output showing the ACO parameters for the current execution, least execution time, number of services assigned, and the service to server allocation.

```
ACO Parameters:
- Number of Ants: 25
- Pheromone Importance: 0.5
- Exploration Factor: 1.0

Environment Setup:
- Number of Services: 6
- Number of Edge Servers: 6

Execution Time: 0.12 seconds
Least Execution Time: 0.08 seconds
Cost Based on Execution Time: $0.08
Provisioned services: 6 out of 6
Service_1. Host: EdgeServer_5
Service_2. Host: EdgeServer_4
Service_3. Host: EdgeServer_6
Service_4. Host: EdgeServer_3
Service_5. Host: EdgeServer_2
Service_6. Host: EdgeServer_1
```

The proposed ACO algorithm in the 2nd Code Cell is replaced by the FCFS algorithm written in Python(Copy and paste the FCFS algorithm in the 2nd Cell).

```python
# Define the FCFS algorithm for service placement
def my_fcfs_algorithm(parameters):
    total_load = np.zeros(len(EdgeServer.all()))
    assigned_servers = set()  # Track which servers are assigned

    # Measure the start time
    start_time = time.time()

    for service in Service.all():
        server_idx = 0  # Start with the first server
        while server_idx < len(EdgeServer.all()):
            edge_server = EdgeServer.all()[server_idx]
            # Check if the server can host the service
            if edge_server.has_capacity_to_host(service):
                print(f"Provisioning {service} to {edge_server}")
                service.provision(target_server=edge_server)
                resource_requirements = service.cpu_demand
                total_load[server_idx] += resource_requirements
                assigned_servers.add(server_idx)  # Mark server as assigned
                break
            server_idx += 1  # Move to the next server

    # Measure the end time
    end_time = time.time()

    # Calculate the execution time
    execution_time = end_time - start_time

    # Calculate variance
    variance = np.var(total_load)
```

Change the function name in the Code Cell 4 (Run simulation cell) to ‘my_fcfs_algorithm’ and run the new simulation.

```
# Creating a Simulator object
simulator = Simulator(
    tick_duration=1,
    tick_unit="seconds",
    stopping_criterion=stopping_criterion,
    resource_management_algorithm=my_fcfs_algorithm
,
)

# Loading a sample dataset from GitHub
simulator.initialize(input_file="https://raw.githubusercontent.com/EdgeSimPy/edgesimpy-tutorials/master/datasets/sample_dataset2.json")

# Executing the simulation
simulator.run_model()

# Checking the placement output
for service in Service.all():
    print(f"{service}. Host: {service.server}")
```

Similarly, after executing the Code Cell 3 and 4, the output for the FCFS algorithm is obtained

```
Optimal Service Placement Results
==================================
Total Load on Servers: [0. 0. 0. 0. 6. 0.]
Variance of Server Load: 5.00
Provisioned services: 6 out of 6

Execution Time: 0.00 seconds
Service_1. Host: EdgeServer_1
Service_2. Host: EdgeServer_1
Service_3. Host: None
Service_4. Host: None
Service_5. Host: None
Service_6. Host: None
Provisioned services: 6 out of 6
Service_1. Host: EdgeServer_5
Service_2. Host: EdgeServer_5
Service_3. Host: EdgeServer_5
Service_4. Host: EdgeServer_5
Service_5. Host: EdgeServer_5
Service_6. Host: EdgeServer_5
```