

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)	<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

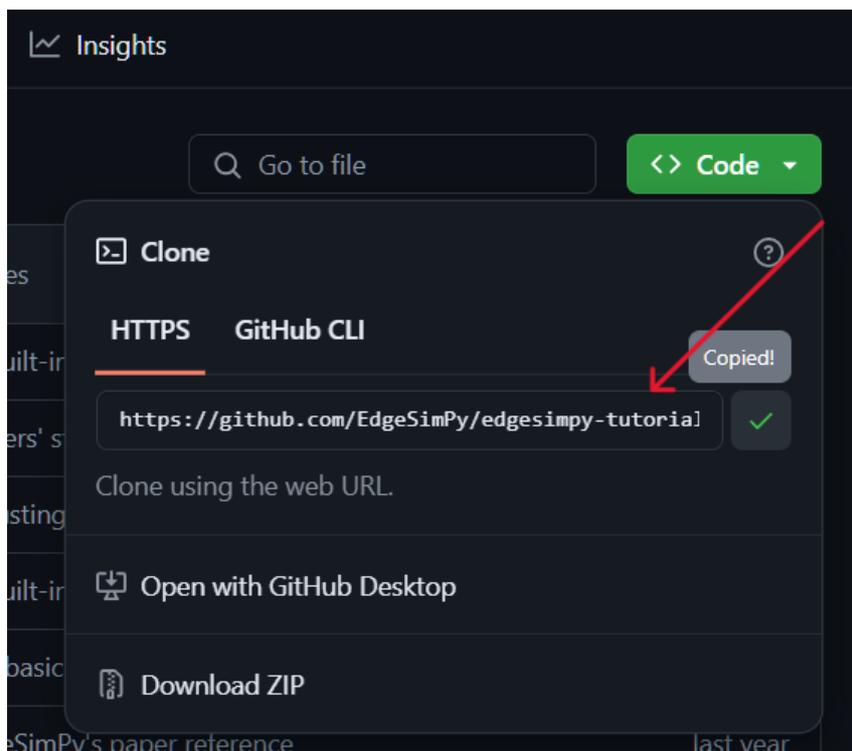
Kurian George
Student ID: x22191437

1 Setting Up EdgeSimPy in Google Colab

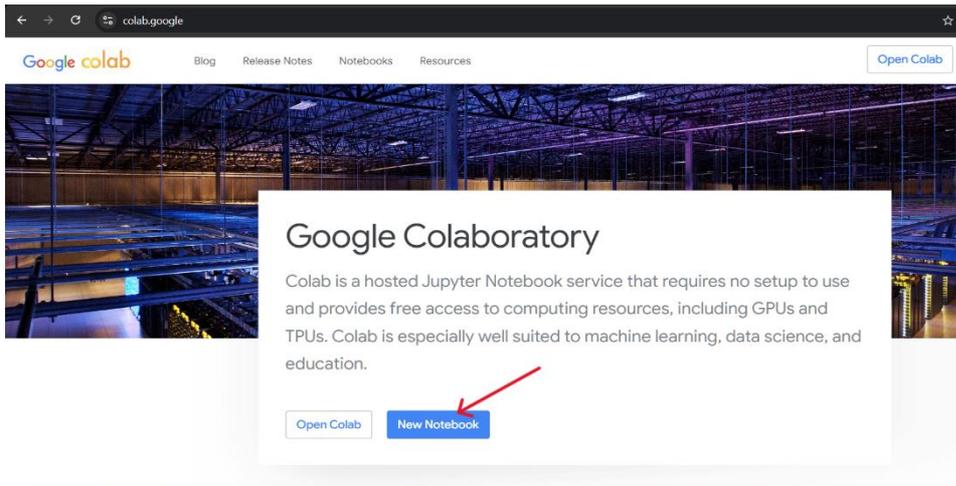
Step 1: Login into GitHub and search EdgeSimPy-tutorials or paste the below URL in Google <https://github.com/EdgeSimPy/edgesimpy-tutorials>

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

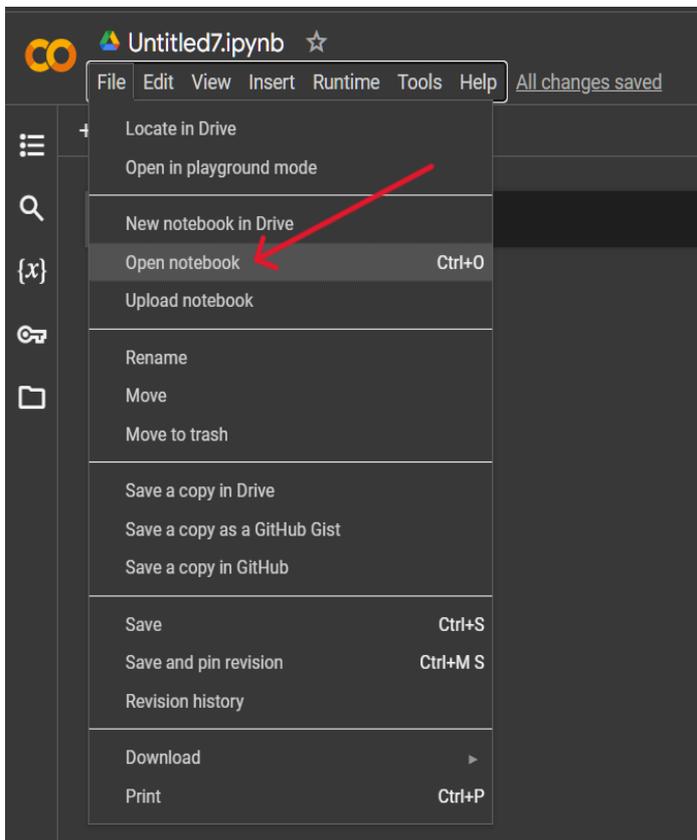
<https://github.com/EdgeSimPy/edgesimpy-tutorials.git>



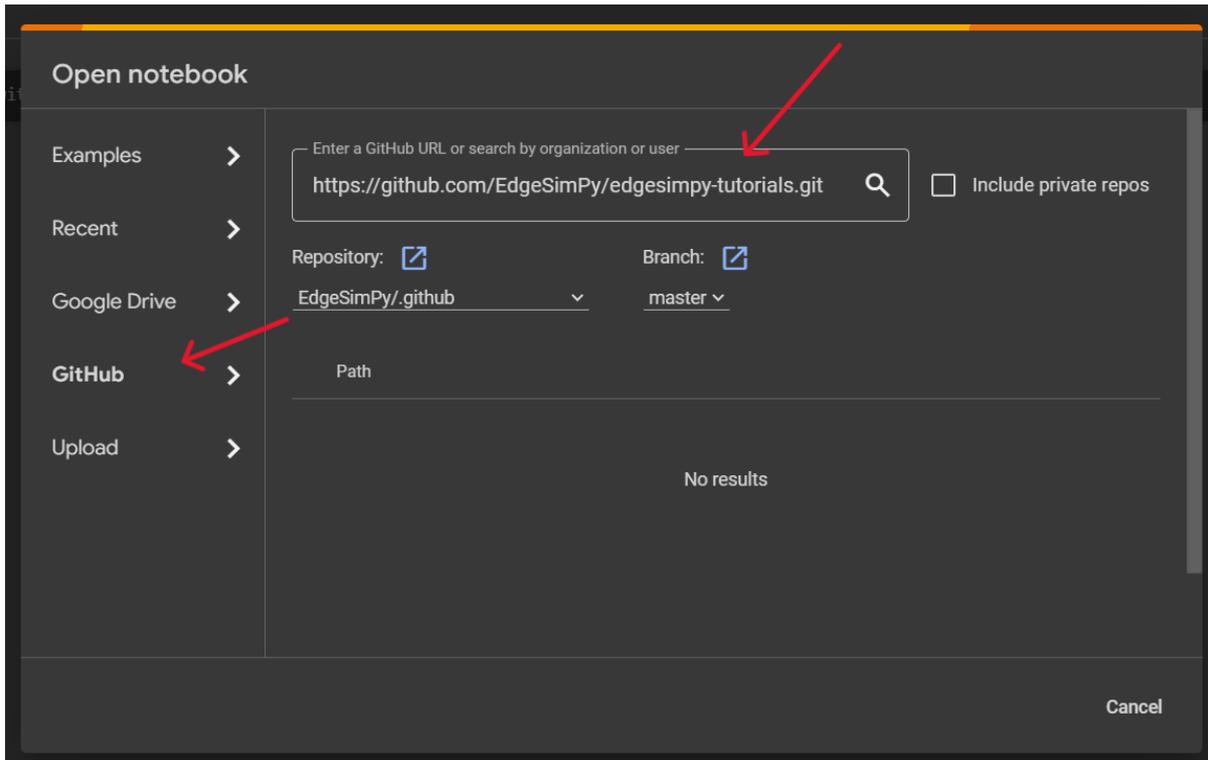
Step 3: Login into Google Colab and Select 'New Notebook' (You'll probably need a gmail ID to login to the console).



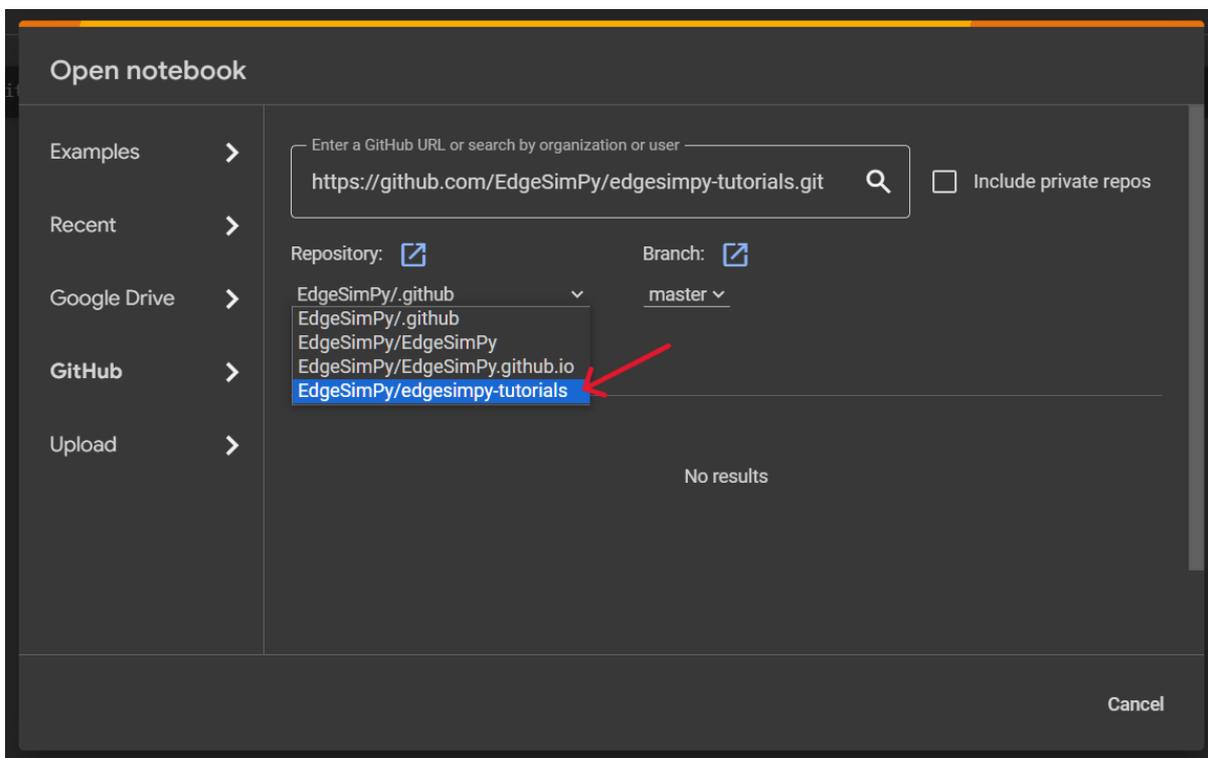
Step 4: After opening a New Notebook, select 'File' from the top left and Click on 'Open Notebook' or press 'Ctrl+O'.



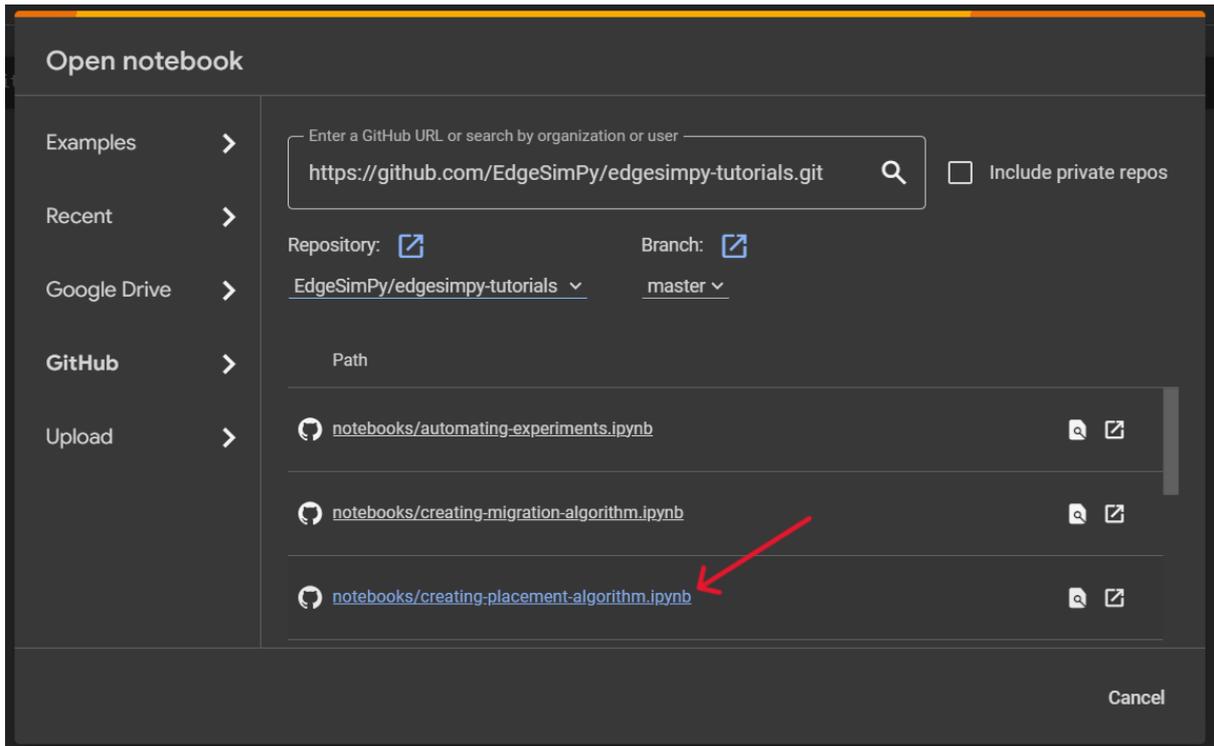
Step 5: In the Open Notebook tab, click on GitHub and paste the HTTPS URL (<https://github.com/EdgeSimPy/edgesimpy-tutorials.git>) in the search box.



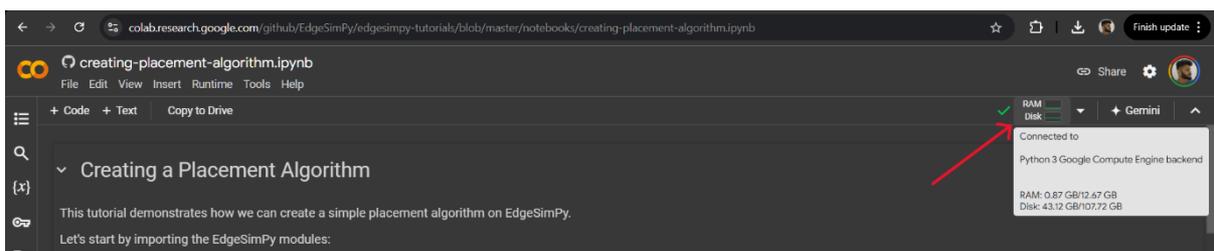
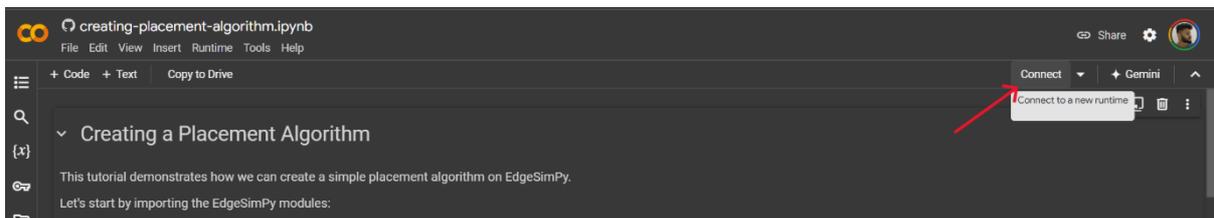
Step 6: In the Open notebook tab, under 'Repository', click on the drop-down arrow and select 'EdgeSimPy/edgesimpy-tutorial'.



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.

As we're creating a placement algorithm, we must instruct EdgeSimPy that it needs to continue the simulation until all services are provisioned within the infrastructure.

To do so, let's create a simple function that will be used as the simulation's stopping criterion. EdgeSimPy will run that function at the end of each time step, halting the simulation as soon as it returns `True`.

```
def stopping_criterion(model: object):
    # Defining a variable that will help us to count the number of services successfully provisioned within the infrastructure
    provisioned_services = 0

    # Iterating over the list of services to count the number of services provisioned within the infrastructure
    for service in Service.all():

        # Initially, services are not hosted by any server (i.e., their "server" attribute is None).
        # Once that value changes, we know that it has been successfully provisioned inside an edge server.
        if service.server != None:
            provisioned_services += 1

    # As EdgeSimPy will halt the simulation whenever this function returns True, its output will be a boolean expression
    # that checks if the number of provisioned services equals to the number of services spanned in our simulation
    return provisioned_services == Service.count()
```

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.

Creating a Placement Algorithm

This tutorial demonstrates how we can create a simple placement algorithm on EdgeSimPy.

Let's start by importing the EdgeSimPy modules:

```
try:
    # Importing EdgeSimPy components
    from edge_sim_py import *
    import networkx as nx
    import msgpack

    # Importing Matplotlib, Pandas, and NumPy for logs parsing and visualization
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    except ModuleNotFoundError:
        # Downloading EdgeSimPy binaries from GitHub (the "-q" parameter suppresses Pip's output. You check the full logs by removing it)
        %pip install -q git+https://github.com/EdgeSimPy/EdgeSimPy.git@v1.1.0

        # Downloading Pandas, NumPy, and Matplotlib (these are not directly used here, but they can be useful for logs parsing and visualization)
        %pip install -q pandas==1.3.5
        %pip install -q numpy==1.26.4
        %pip install -q matplotlib==3.5.0

        # Importing EdgeSimPy components and its built-in libraries (NetworkX and MessagePack)
        from edge_sim_py import *
```

```
try:
    # Importing EdgeSimPy components
    from edge_sim_py import *
    import networkx as nx
    import msgpack

    # Importing Matplotlib, Pandas, and NumPy for logs parsing and visualization
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    # Importing ACOR from mealpy
    from mealpy import FloatVar, ACOR
```

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

```

try:
    # Importing EdgeSimPy components
    from edge_sim_py import *
    import networkx as nx
    import msgpack

    # Importing Matplotlib, Pandas, and NumPy for logs parsing and visualization
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

```

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

```

Defining ACO Algorithm

import time
import numpy as np

# Define a cost per second for the execution time
COST_PER_SECOND = 0.1 # Example: $0.10 per second

# Define the algorithm
def my_algorithm(parameters):
    def objective_function(solution):
        total_load = np.zeros(len(EdgeServer.all()))
        assigned_servers = set() # Track which servers are assigned
        execution_times = [] # Track execution times for services

        for service_idx, server_idx in enumerate(solution):
            service = Service.all()[service_idx]
            server_idx = int(round(server_idx)) # Ensure to round correctly

            if server_idx < 0 or server_idx >= len(EdgeServer.all()):
                print(f"Invalid server index for service {service_idx}: {server_idx}")
                continue

            resource_requirements = service.cpu_demand
            total_load[server_idx] += resource_requirements
            assigned_servers.add(server_idx) # Add to assigned servers

            # Get the edge server
            edge_server = EdgeServer.all()[server_idx]

            # Check the available attribute for capacity, e.g., 'cpu', 'ram', etc.

```

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

```

Define Stopping Criterion

[4] def stopping_criterion(model: object):
    # Defining a variable that will help us to count the number of services successfully provisioned within the infrastructure
    provisioned_services = 0

    # Iterating over the list of services to count the number of services provisioned within the infrastructure
    for service in Service.all():
        # Initially, services are not hosted by any server (i.e., their "server" attribute is None).
        # Once that value changes, we know that it has been successfully provisioned inside an edge server.
        if service.server is not None:
            provisioned_services += 1

    print(f"Provisioned services: {provisioned_services} out of {Service.count()}")

    # As EdgeSimPy will halt the simulation whenever this function returns True, its output will be a boolean expression
    # that checks if the number of provisioned services equals to the number of services spawned in our simulation
    return provisioned_services == Service.count()

```

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

```

# Creating a Simulator object
simulator = Simulator(
    tick_duration=1,
    tick_units="seconds",
    stopping_criterion=stopping_criterion,
    resource_management_algorithm=my_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}")

INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 19, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01522 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 20, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01221 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 21, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01395 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 22, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01398 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 23, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01403 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 24, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01609 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 25, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.00987 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 26, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01225 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 27, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01772 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 28, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01650 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 29, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01580 seconds
INFO:meaply.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 30, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01095 seconds

```

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.

```

def my_algorithm(parameters):
    def objective_function(solution):
        total_load = np.zeros(len(EdgeServer.all()))
        assigned_servers = set() # Track which servers are assigned
        execution_times = [] # Track execution times for services

        # Estimate execution time using CPU demand and server capacity
        execution_time = service.cpu_demand / server_capacity if server_capacity > 0 else float('inf')
        execution_times.append(execution_time)

        # Calculate variance
        variance = np.var(total_load)

        # Add a penalty for solutions where not all servers are used
        if len(assigned_servers) < len(total_load): # If not all servers are used
            variance += 10 # Add a penalty to encourage use of all servers

        # Calculate the total execution time (objective is to minimize this)
        total_execution_time = np.sum(execution_times)

        # Objective function should combine variance and execution time
        return variance + total_execution_time

```

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

```

# Estimate execution time using CPU demand and server capacity
execution_time = service.cpu_demand / server_capacity if server_capacity > 0 else float('inf')
execution_times.append(execution_time)

# Calculate variance
variance = np.var(total_load)

# Add a penalty for solutions where not all servers are used
if len(assigned_servers) < len(total_load): # If not all servers are used
    variance += 10 # Add a penalty to encourage use of all servers

# Calculate the total execution time (objective is to minimize this)
total_execution_time = np.sum(execution_times)

# Objective function should combine variance and execution time
return variance + total_execution_time

```

The ACO algorithm parameters to find the solution

```

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

model = ACOR.OriginalACOR(epoch=50, pop_size=25, sample_count=15, intent_factor=0.5, zeta=1.0)

g_best = model.solve(problem_dict)

```

The logic to allocate servers to services.

```
[x]
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.

```
INFO:mealpy.swarm_based.ACOR.OriginalACOR>>>Problem: P, Epoch: 30, Current best: 0.6666666666666666, Global best: 0.6666666666666666, Runtime: 0.01337 seconds
Optimal Service Placement Results
Best Solution Found: [4.19283049 5. 2.65981423 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).

```
# 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("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.

```
[X] 4
# 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
```