CSE540 Cloud Computing

Serverless Computing

Pitch

Group 2

Vinay Kakkad

vinay.k@ahduni.edu.in

Henil Shah

shah.h@ahduni.edu.in

Sameep Vani

sameep.v@ahduni.edu.in

Samkit Kundalia

samkit.k@ahduni.edu.in

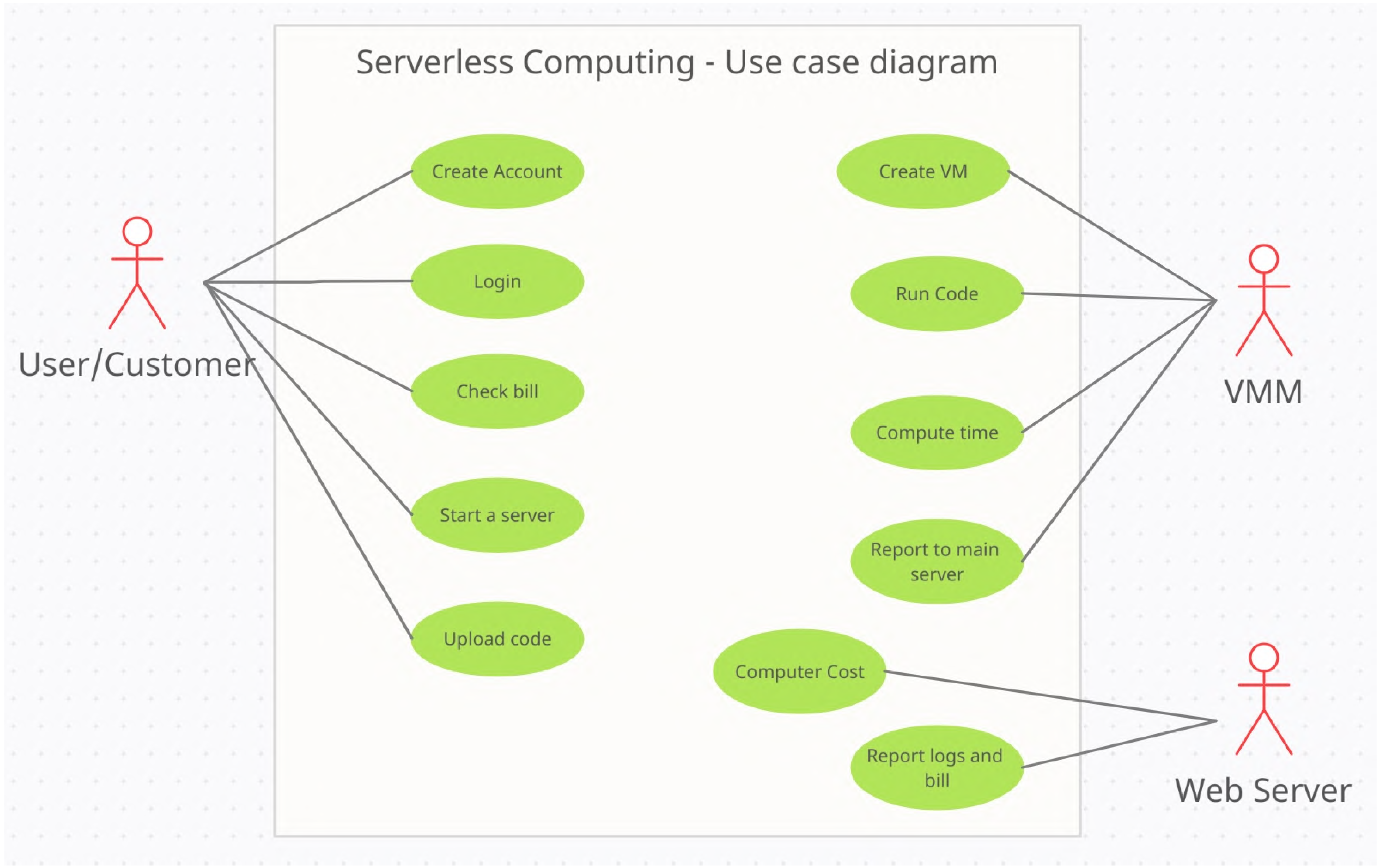
Goal and problem statement

Our main aim is to provide function-as-a-service (FaaS) – a cloud computing serivce that makes it easier for cloud developers to manage microservices.

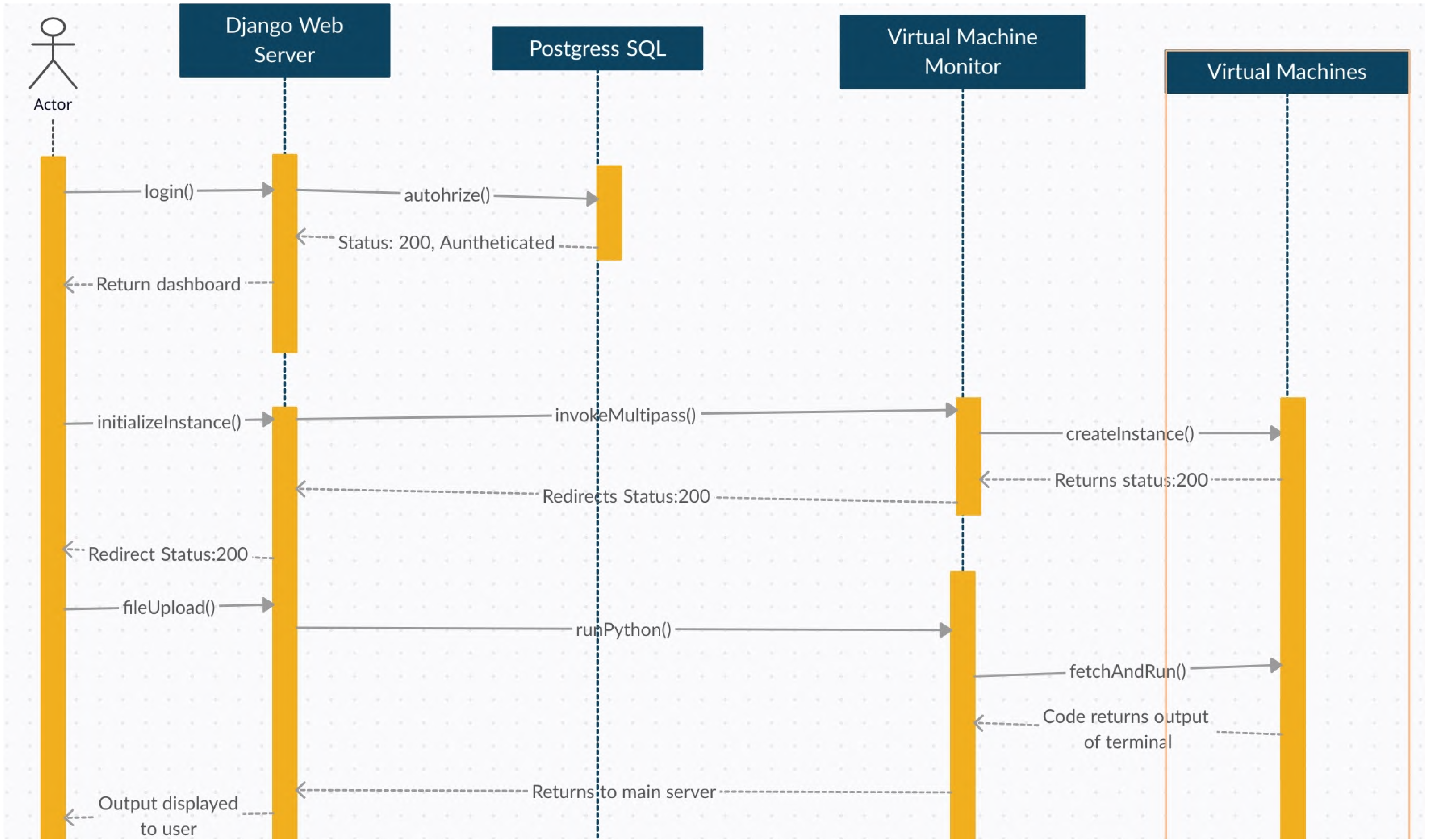
🎯 Milestones

- Virtualize the hardware to create micro VMs.
- Create a centralized server to handle multiple user requests.
- Create multipass scripts for the provided config files and save it to the centralized server.
- Provide output to the user and employ Pay-As-Use Model.
- Integrate the above components to get a full fledged serverless computing service.

Use–Case Diagram

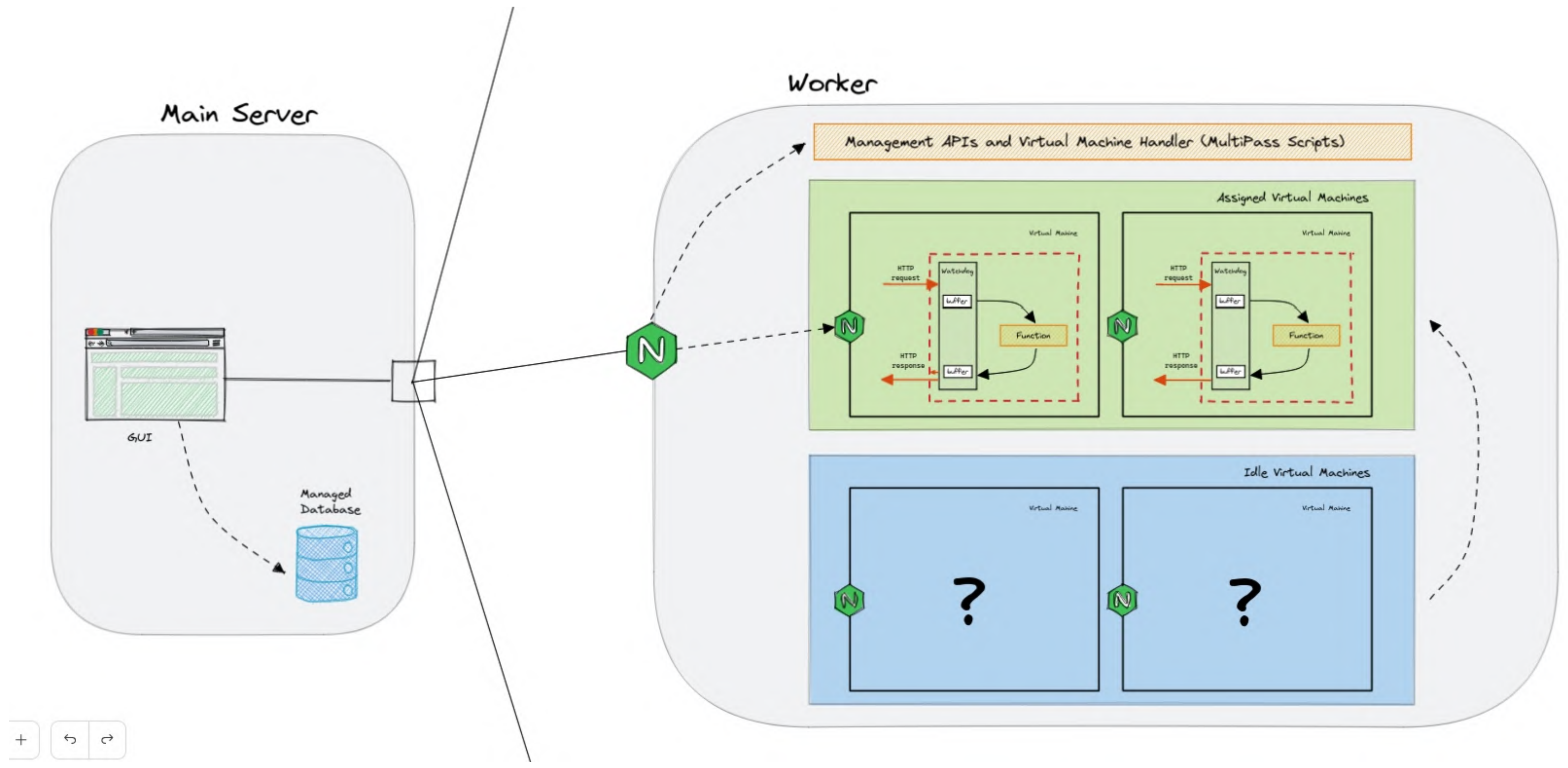


Sequence Diagram



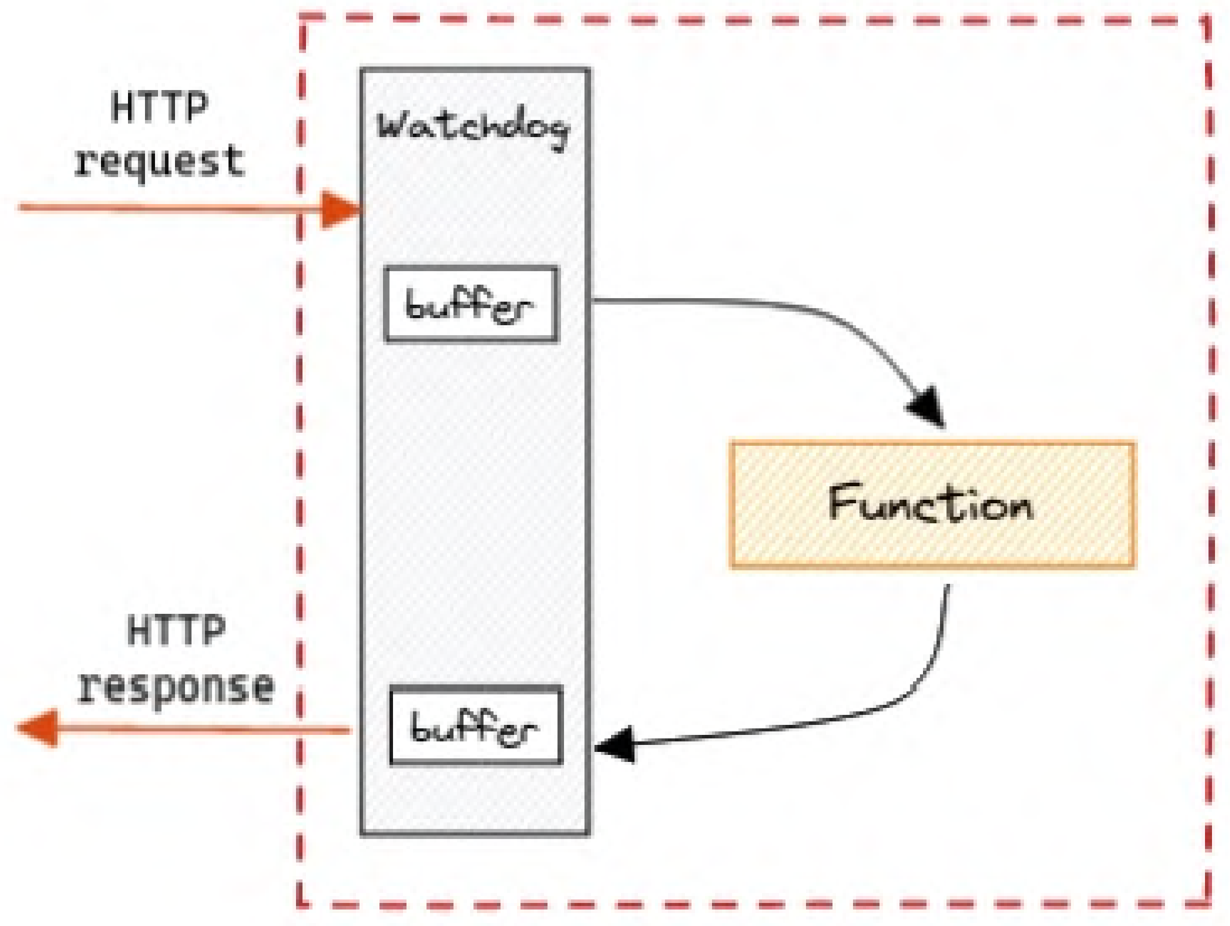
Pitch

Components



https://excalidraw.com/#room=d963a8603585514eed30,VN4nRH3BySwBaQ0YxdgEtA

Components



https://excalidraw.com/#room=d963a8603585514eed30,VN4nRH3BySwBaQ0YxdgEtA

List of Cloud Services Used

Multipass

DigitalOcean

Docker

PostgreSQL

Multipass

- Multipass is a VMM tool for creating and managing instances of Linux operating systems on a computer.
- It is based on the libvirt virtualization library and uses the KVM hypervisor to run the virtual machines.
- It also integrates with popular tools such as Docker and Kubernetes to ease out deployments and managing containarized application.

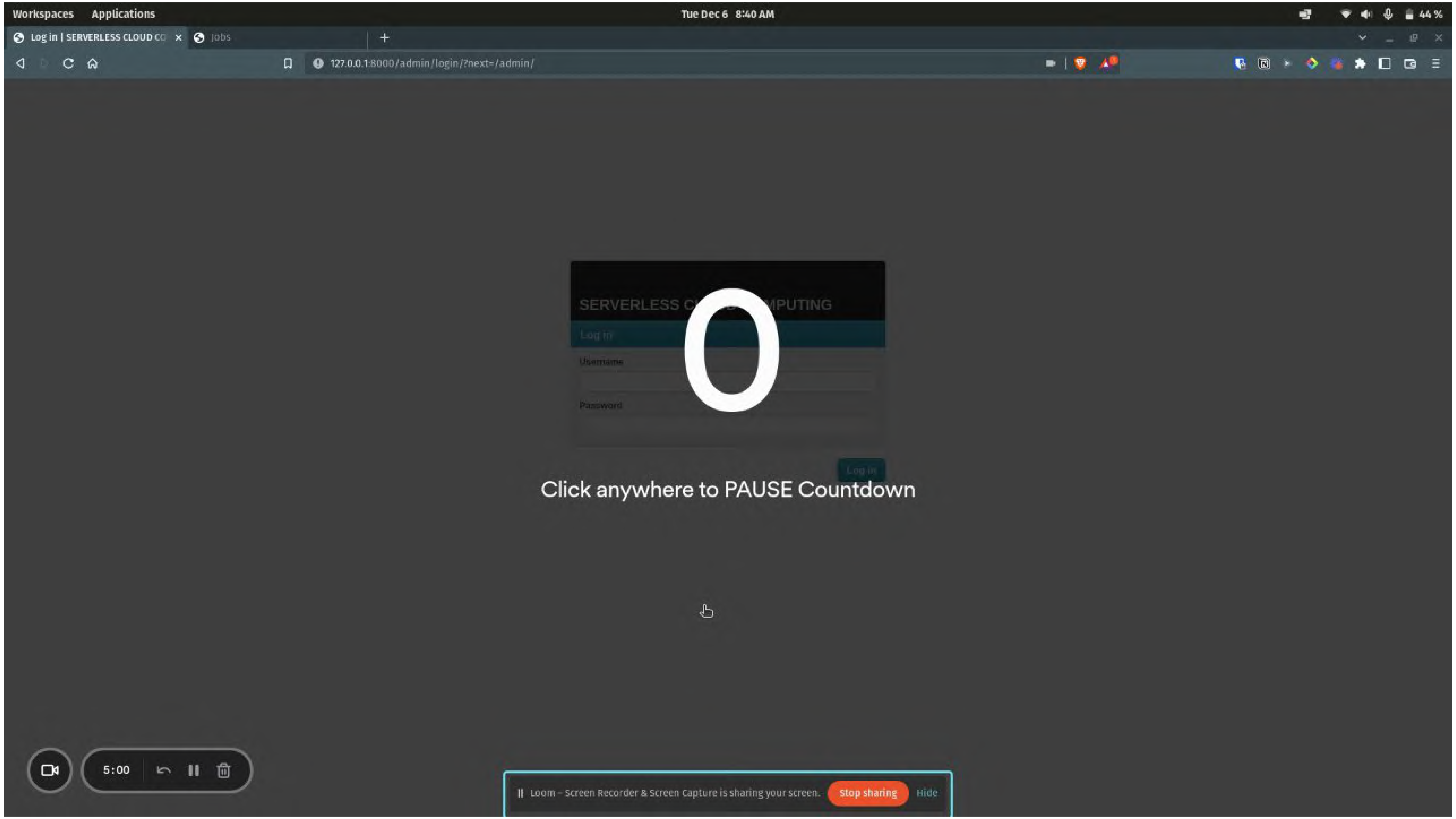
VMM setup in cclab

- Installed Ubuntu 20.04.5 LTS (Focal Fossa) on three lab computers, where cclab1 acts as Hyper-V , cclab2 and cclab3 act as VMs
- Set up the infrastructure using Mutipass Virtualisation Technology.

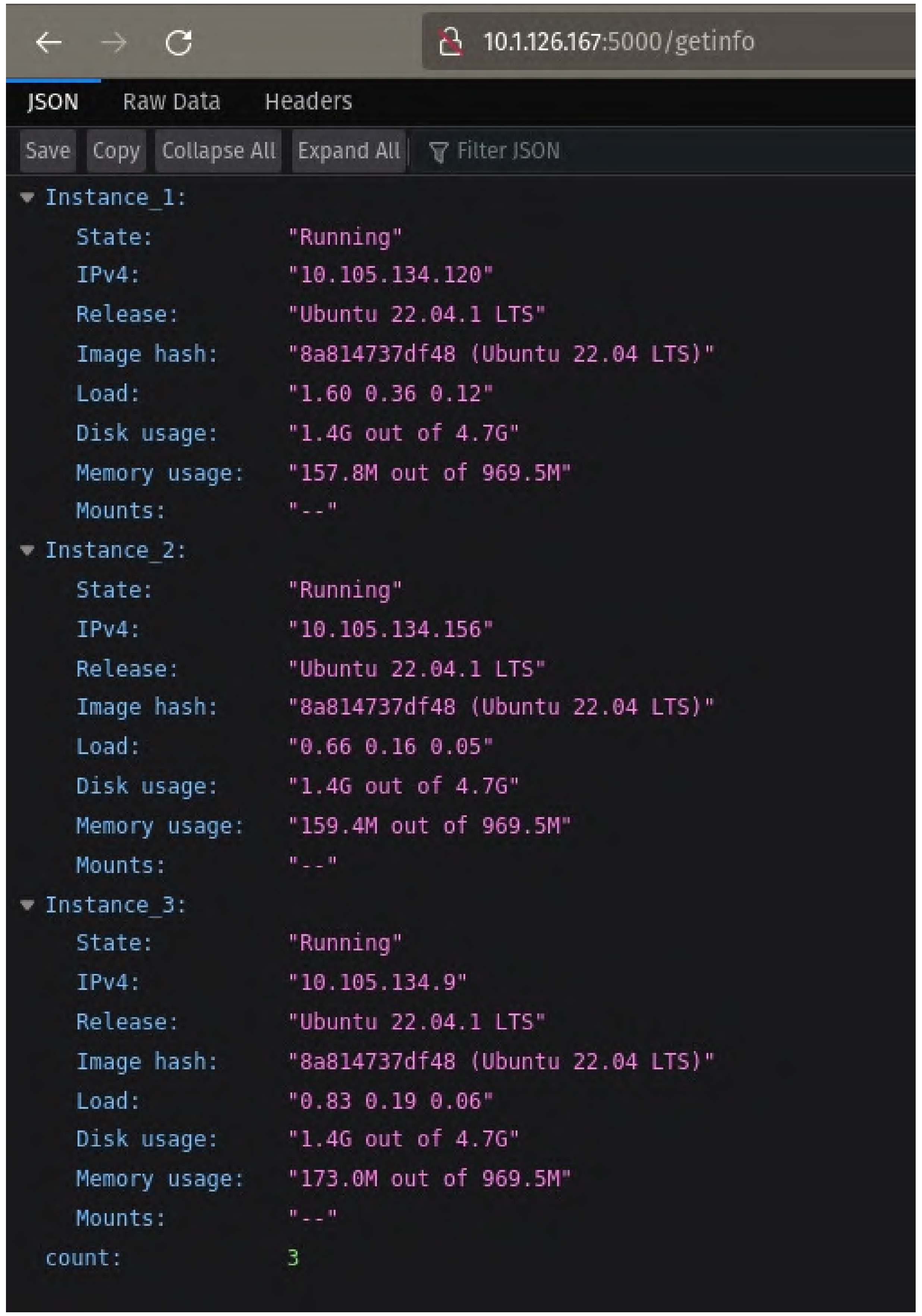
THOUGHT PROCESSES INVOLVED

- Debugging and setting up firecracker was very elaborate process, hence, we decided to use multipass as our VMM.

Results


Workspaces
Applications
Tue Dec 6 8:40 AM
44 %
Log in | SERVERLESS CLOUD CO
Jobs
127.0.0.1:8000/admin/login/?next=/admin/
SERVERLESS C MPUTING
Log in
Username
Password
Log in
0
Click anywhere to PAUSE Countdown
5:00
Loom – Screen Recorder & Screen Capture is sharing your screen.
Stop sharing
Hide

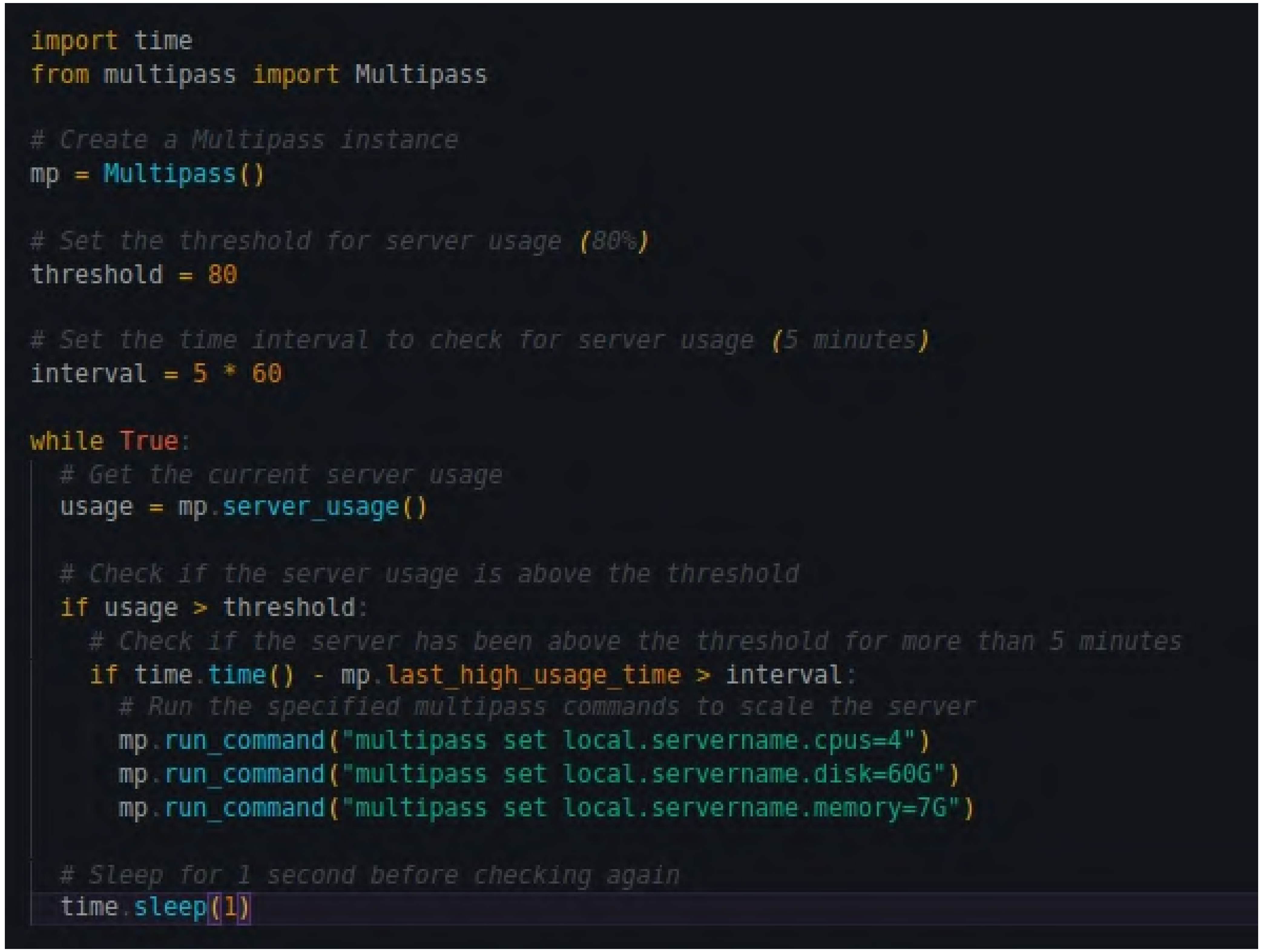
Results



```
▾ Instance_1:
    State:          "Running"
    IPv4:           "10.105.134.120"
    Release:        "Ubuntu 22.04.1 LTS"
    Image hash:     "8a814737df48 (Ubuntu 22.04 LTS)"
    Load:           "1.60 0.36 0.12"
    Disk usage:     "1.4G out of 4.7G"
    Memory usage:   "157.8M out of 969.5M"
    Mounts:         "--"
▾ Instance_2:
    State:          "Running"
    IPv4:           "10.105.134.156"
    Release:        "Ubuntu 22.04.1 LTS"
    Image hash:     "8a814737df48 (Ubuntu 22.04 LTS)"
    Load:           "0.66 0.16 0.05"
    Disk usage:     "1.4G out of 4.7G"
    Memory usage:   "159.4M out of 969.5M"
    Mounts:         "--"
▾ Instance_3:
    State:          "Running"
    IPv4:           "10.105.134.9"
    Release:        "Ubuntu 22.04.1 LTS"
    Image hash:     "8a814737df48 (Ubuntu 22.04 LTS)"
    Load:           "0.83 0.19 0.06"
    Disk usage:     "1.4G out of 4.7G"
    Memory usage:   "173.0M out of 969.5M"
    Mounts:         "--"
  count:            3
```

- Instances running on multipass
- Currently there are three running
- Specifications of each is different based on what is required

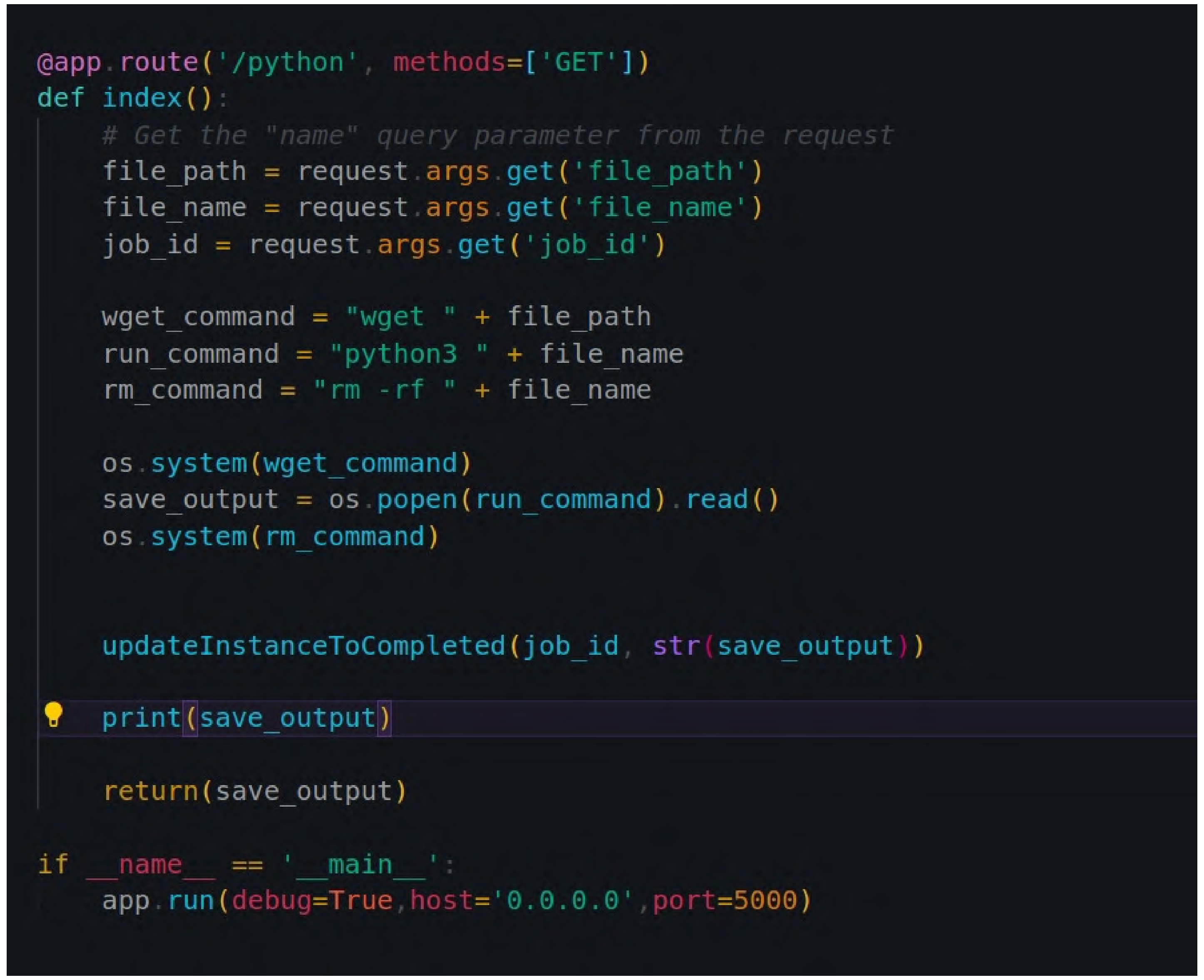
Code Snippets



```python
import time
from multipass import Multipass

# Create a Multipass instance
mp = Multipass()

# Set the threshold for server usage (80%)
threshold = 80

# Set the time interval to check for server usage (5 minutes)
interval = 5 * 60

while True:
  # Get the current server usage
  usage = mp.server_usage()

  # Check if the server usage is above the threshold
  if usage > threshold:
    # Check if the server has been above the threshold for more than 5 minutes
    if time.time() - mp.last_high_usage_time > interval:
      # Run the specified multipass commands to scale the server
      mp.run_command("multipass set local.servername.cpus=4")
      mp.run_command("multipass set local.servername.disk=60G")
      mp.run_command("multipass set local.servername.memory=7G")

  # Sleep for 1 second before checking again
  time.sleep(1)
```

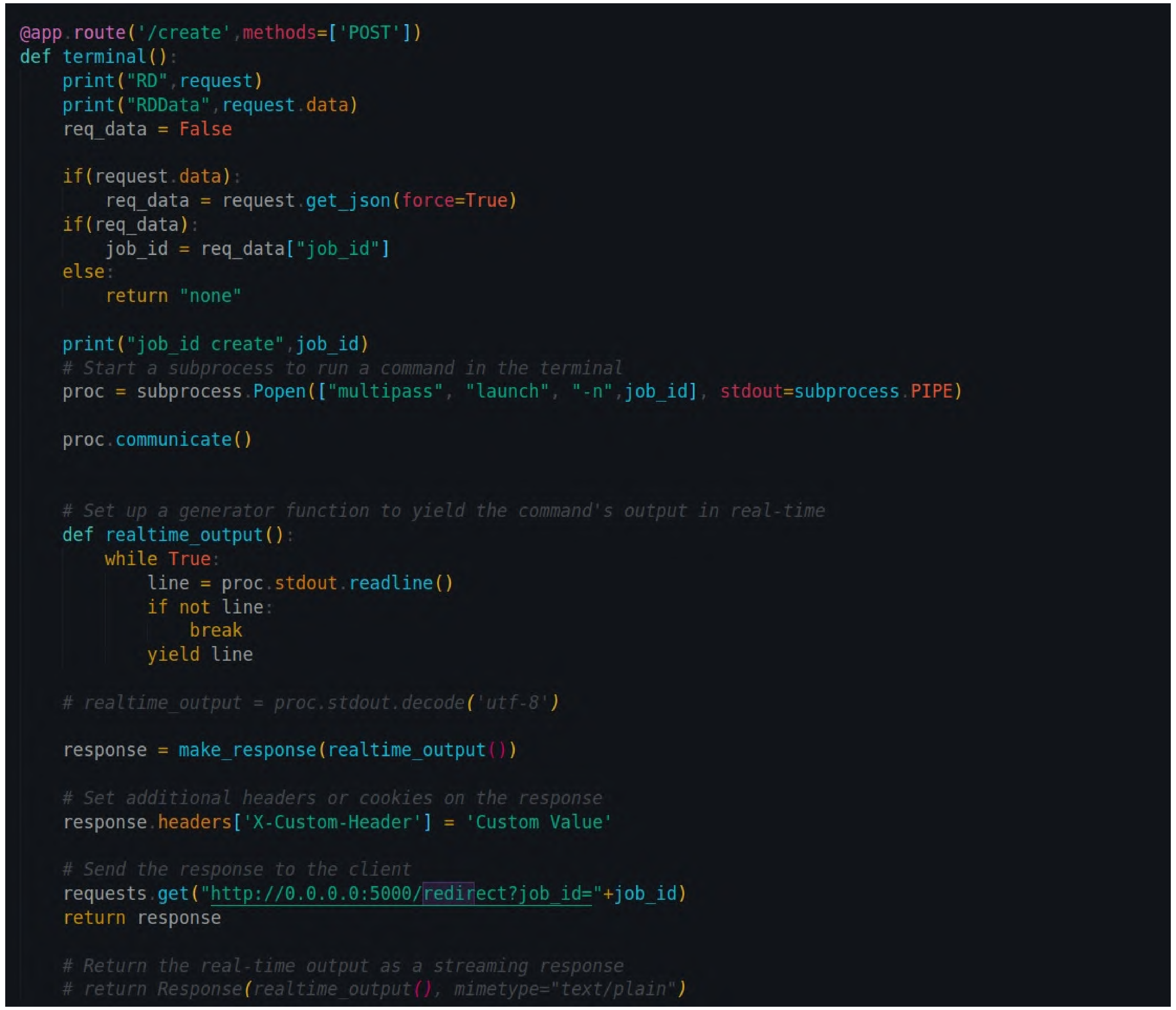
- The moment an instance starts, we run a script for that VM that checks if the usage > 80% for more than 5 minutes (configurable) and then autoscales the instance to double it.

Code Snippets



```python
@app.route('/python', methods=['GET'])
def index():
    # Get the "name" query parameter from the request
    file_path = request.args.get('file_path')
    file_name = request.args.get('file_name')
    job_id = request.args.get('job_id')

    wget_command = "wget " + file_path
    run_command = "python3 " + file_name
    rm_command = "rm -rf " + file_name

    os.system(wget_command)
    save_output = os.popen(run_command).read()
    os.system(rm_command)


    updateInstanceToCompleted(job_id, str(save_output))

    print(save_output)

    return(save_output)

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=5000)
```

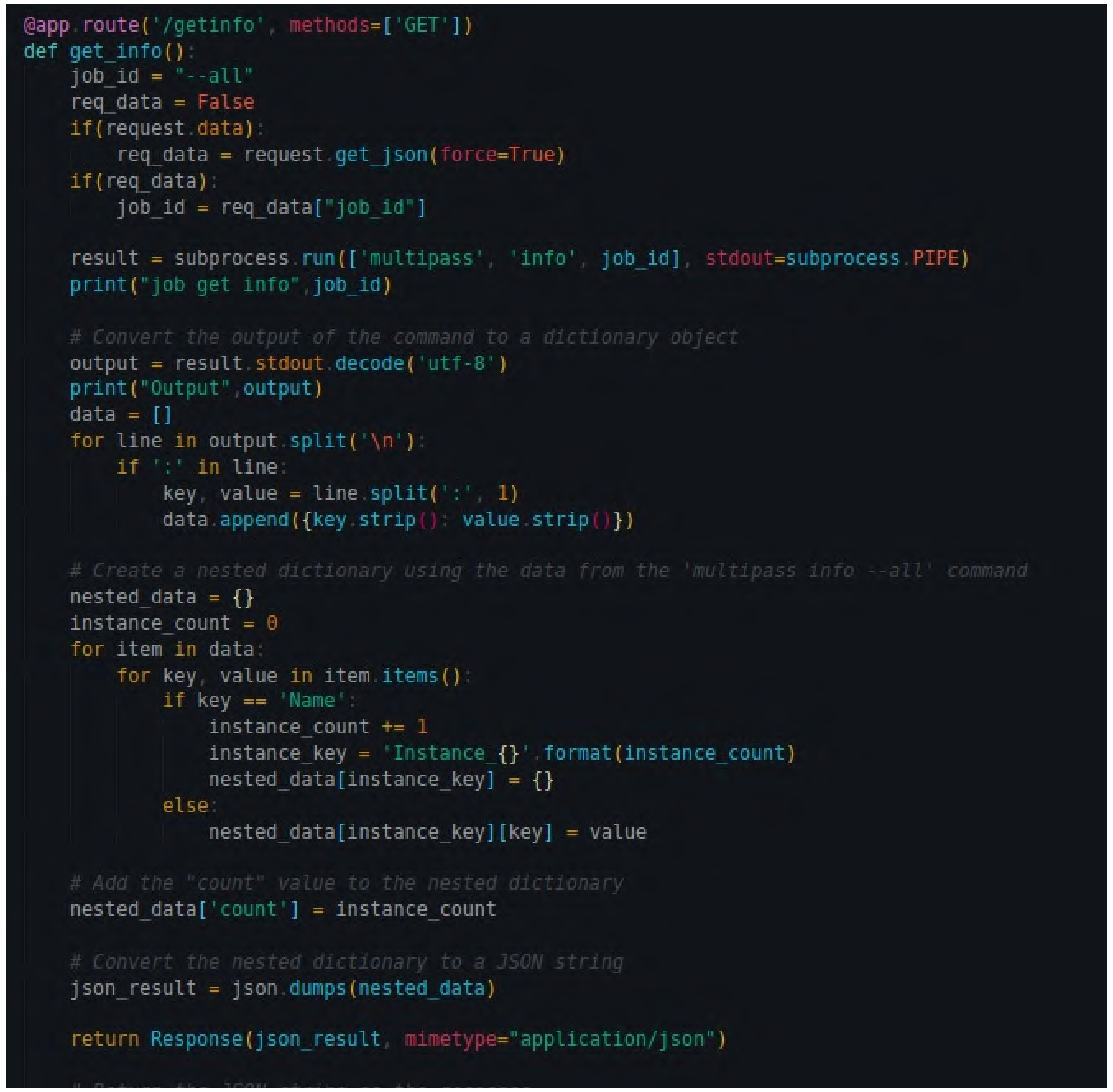
Runs uploaded file in python



```python
@app.route('/create',methods=['POST'])
def terminal():
    print("RD",request)
    print("RDData",request.data)
    req_data = False

    if(request.data):
        req_data = request.get_json(force=True)
    if(req_data):
        job_id = req_data["job_id"]
    else:
        return "none"

    print("job_id create",job_id)
    # Start a subprocess to run a command in the terminal
    proc = subprocess.Popen(["multipass", "launch", "-n",job_id], stdout=subprocess.PIPE)

    proc.communicate()


    # Set up a generator function to yield the command's output in real-time
    def realtime_output():
        while True:
            line = proc.stdout.readline()
            if not line:
                break
            yield line

    # realtime_output = proc.stdout.decode('utf-8')

    response = make_response(realtime_output())

    # Set additional headers or cookies on the response
    response.headers['X-Custom-Header'] = 'Custom Value'

    # Send the response to the client
    requests.get("http://0.0.0.0:5000/redirect?job_id="+job_id)
    return response

    # Return the real-time output as a streaming response
    # return Response(realtime_output(), mimetype="text/plain")
```

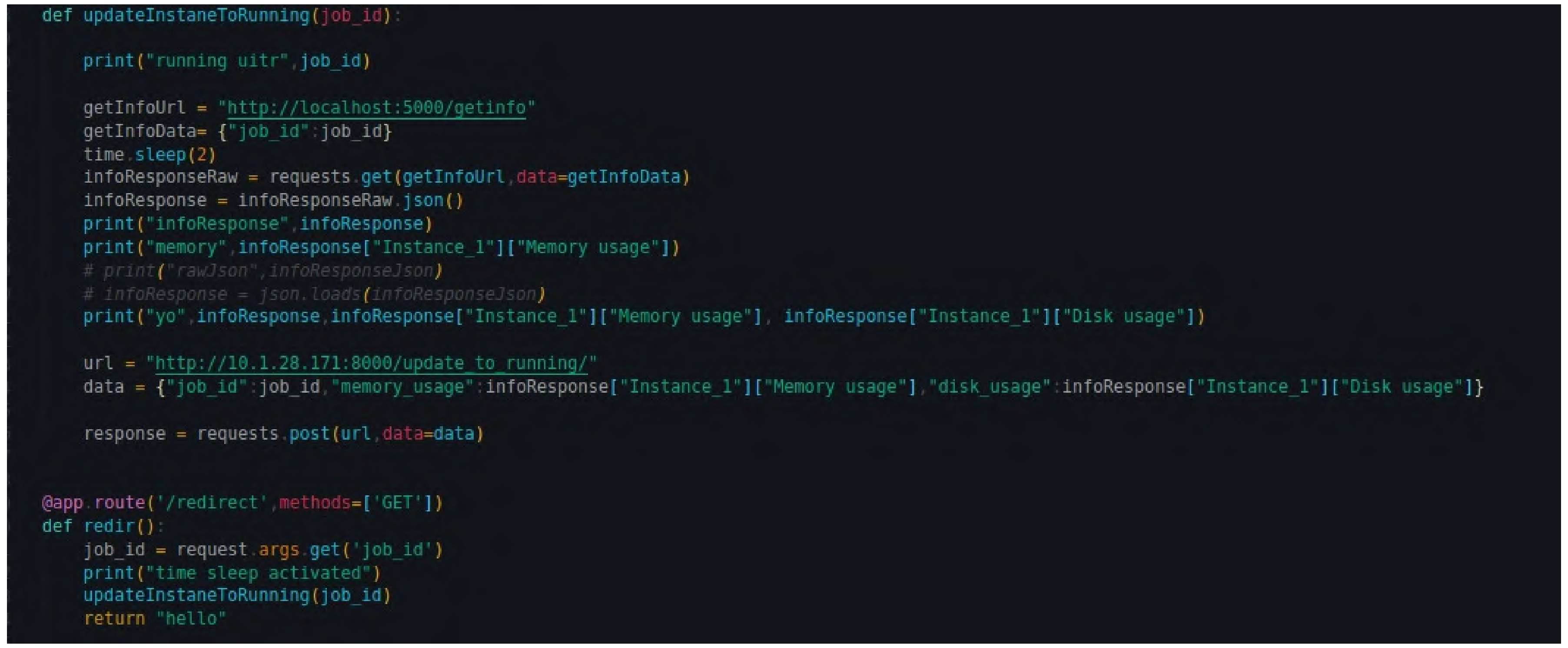
Instance Initialization

Code Snippets



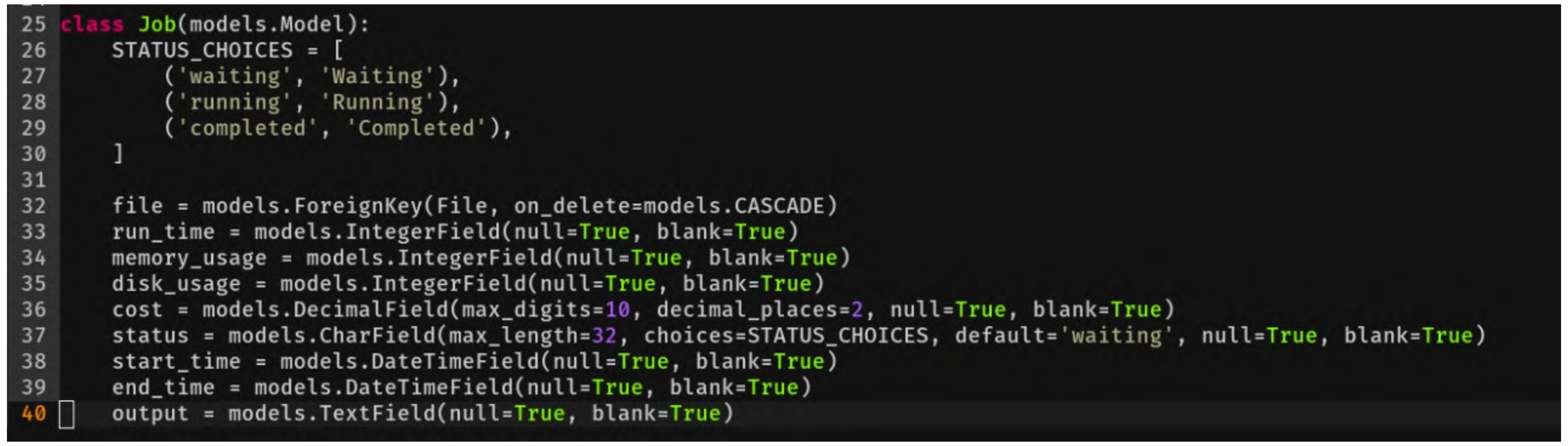
```python
@app.route('/getinfo', methods=['GET'])
def get_info():
    job_id = "--all"
    req_data = False
    if(request.data):
        req_data = request.get_json(force=True)
    if(req_data):
        job_id = req_data["job_id"]

    result = subprocess.run(['multipass', 'info', job_id], stdout=subprocess.PIPE)
    print("job get info",job_id)

    # Convert the output of the command to a dictionary object
    output = result.stdout.decode('utf-8')
    print("Output",output)
    data = []
    for line in output.split('\n'):
        if ':' in line:
            key, value = line.split(':', 1)
            data.append({key.strip(): value.strip()})

    # Create a nested dictionary using the data from the 'multipass info --all' command
    nested_data = {}
    instance_count = 0
    for item in data:
        for key, value in item.items():
            if key == 'Name':
                instance_count += 1
                instance_key = 'Instance_{}'.format(instance_count)
                nested_data[instance_key] = {}
            else:
                nested_data[instance_key][key] = value

    # Add the "count" value to the nested dictionary
    nested_data['count'] = instance_count

    # Convert the nested dictionary to a JSON string
    json_result = json.dumps(nested_data)

    return Response(json_result, mimetype="application/json")
```

VMM Stats



```python
def updateInstaneToRunning(job_id):

    print("running uitr",job_id)

    getInfoUrl = "http://localhost:5000/getinfo"
    getInfoData= {"job_id":job_id}
    time.sleep(2)
    infoResponseRaw = requests.get(getInfoUrl,data=getInfoData)
    infoResponse = infoResponseRaw.json()
    print("infoResponse",infoResponse)
    print("memory",infoResponse["Instance_1"]["Memory usage"])
    # print("rawJson",infoResponseJson)
    # infoResponse = json.loads(infoResponseJson)
    print("yo",infoResponse,infoResponse["Instance_1"]["Memory usage"], infoResponse["Instance_1"]["Disk usage"])

    url = "http://10.1.28.171:8000/update_to_running/"
    data = {"job_id":job_id,"memory_usage":infoResponse["Instance_1"]["Memory usage"],"disk_usage":infoResponse["Instance_1"]["Disk usage"]}

    response = requests.post(url,data=data)


@app.route('/redirect',methods=['GET'])
def redir():
    job_id = request.args.get('job_id')
    print("time sleep activated")
    updateInstaneToRunning(job_id)
    return "hello"
```

Update status



```python
class Job(models.Model):
    STATUS_CHOICES = [
        ('waiting', 'Waiting'),
        ('running', 'Running'),
        ('completed', 'Completed'),
    ]

    file = models.ForeignKey(File, on_delete=models.CASCADE)
    run_time = models.IntegerField(null=True, blank=True)
    memory_usage = models.IntegerField(null=True, blank=True)
    disk_usage = models.IntegerField(null=True, blank=True)
    cost = models.DecimalField(max_digits=10, decimal_places=2, null=True, blank=True)
    status = models.CharField(max_length=32, choices=STATUS_CHOICES, default='waiting', null=True, blank=True)
    start_time = models.DateTimeField(null=True, blank=True)
    end_time = models.DateTimeField(null=True, blank=True)
    output = models.TextField(null=True, blank=True)
```

Tracks metrics and generates bill

Future Work

- Use docker, pre-built containers, idle virtual machines and solve the problem of cold start
- Extends services like functions which can take in additional input parameters
- Using multiple slave nodes for scalability

Conclusion

- Implemented FaaS by just taking the input of the function file, and the managed all the things.
- Also, we charge for resources used and the time taken for the function to run, hence following the pay-as-you-use model.
- Also, used docker images for environments to help reduce the setup time for virtual machines.

References

- *Behind the scenes, lambda* (2021) */home/bruno*. Available at: https://www.bschaatsbergen.com/behind-the-scenes-lambda/
- *What is KVM? Red Hat - We make open source technologies for the enterprise*. Available at: https://www.redhat.com/en/topics/virtualization/what-is-KVM
- *Developer blog Fusebit*. Available at: https://fusebit.io/blog/2018/03/25/how-to-build-your-own-serverless-platform/
- *Whitepapers Amazon*. Earthpledge Foundation. Available at: https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/lambda-executions.html
- Anaibol (no date) *Anaibol/awesome-serverless: A curated list of Awesome Services, Solutions and resources for serverless / NOBACKEND applications.*, *GitHub*. Available at: https://github.com/anaibol/awesome-serverless

References

- Velichko, I. (2021) *OpenFaaS – run containerized functions on your own terms, Ivan on Containers, Kubernetes, and Server-Side*. Available at: https://iximiuz.com/en/posts/openfaas-case-study/
- *Multipass documentation* (no date) *Multipass Documentation | Multipass documentation*. Available at: https://multipass.run/docs

Thank you