****

**CSE540 | Cloud Computing**

A Report on

**EC2 Replica: VM Management Using KVM**

**(An Infrastructure-as-a-Service (IaaS) Implementation)**

Submitted to

Prof. Sanjay Chaudhary

Group Members

| **Name** | **Email Address** | **Roll No.** |
| --- | --- | --- |
| Anvi Sheth | anvi.s@ahduni.edu.in | AU22L20001 |
| Harsh Pandya | harsh.p9@ahduni.edu.in | AU2140171 |
| Nevil Jobanputra | nevil.j@ahduni.edu.in | AU2140209 |
| Muskan Chauhan | muskan.c1@ahduni.edu.in | AU22L20003 |

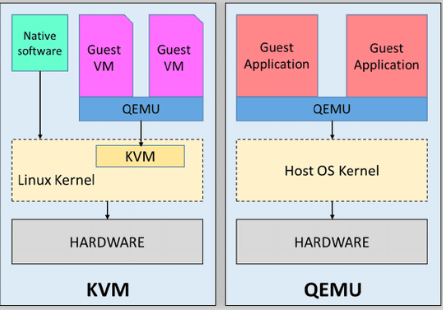
**Project Statement**

This project aims to design and implement an IaaS platform, inspired by Amazon EC2. With the use of KVM, this project will allow the creation of a highly scalable virtual machine environment for users. This will enable the foundational capability of hosting and managing virtualized instances, which involves hardware abstraction, resource allocation, and isolation. This project demonstrates the end-to-end lifecycle of creating, configuring, and managing virtual machines, along with user-friendly management interfaces to support operations like deployment, scaling, and deletion of virtual instances. This project, by using open-source tools such as QEMU, virt-manager, and Linux utilities, demonstrates the feasibility of building cloud infrastructure efficiently.

**Project Design**

1. Client-side: Users interact with the front-end graphical interface to create virtual machines.
2. Server-side: KVM and QEMU for virtual machine management and virtualization.
3. Virtualization Layer: KVM provides hardware-based virtualization for efficient resource utilization.
4. Storage and Networking: Virtual machines will be stored on local or network storage, and networking will be configured through bridge interfaces.

**Architecture**



KVM Architecture (Left Side):

* KVM is a hardware-assisted virtualization solution that integrates into the Linux kernel. It allows virtual machines (VMs) to run with near-native performance.

Components:

* QEMU: Provides device emulation and VM management. QEMU interacts with the guest VMs.
* KVM Module: A kernel module that enables CPU virtualization by using hardware support (Intel VT-x or AMD-V).
* Linux Kernel: The host operating system's kernel manages the hardware and provides KVM capabilities.
* Guest VMs: The virtual machines that run their own operating systems with hardware acceleration provided by KVM.

Key Feature:

* The CPU instructions of the guest VMs are executed directly on the hardware through KVM, which improves performance.

QEMU Architecture (Right Side):

* QEMU, on its own, acts as a software-based emulator and can fully emulate hardware, enabling virtual machines to run without hardware-assisted virtualization (slower than KVM).

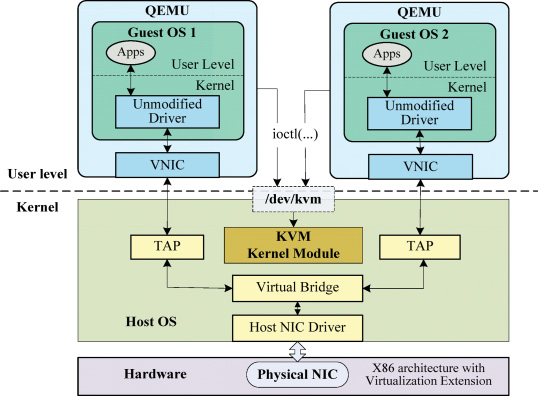
Components:

* QEMU: Handles both system emulation and hardware emulation for the guest applications.
* Host OS Kernel: The host operating system's kernel (not necessarily Linux).
* Guest Applications: Applications running inside the virtual environment emulated by QEMU.

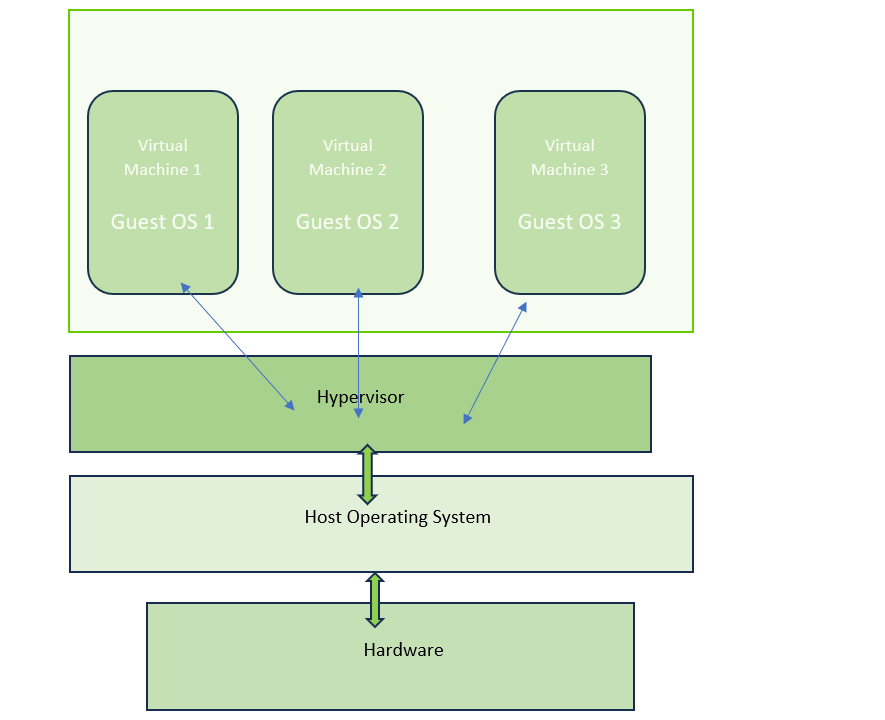
Key Feature:

* QEMU can emulate an entire system, including the CPU and peripherals, regardless of the host's hardware or architecture.
* KVM + QEMU: Best for performance and emulating VMs with hardware support.
* QEMU alone: Ideal for full system emulation when hardware virtualization is unavailable or cross-architecture emulation is needed.

Flowchart:

****

* This flowchart shows the integration of QEMU and KVM for hardware-assisted virtualization. QEMU offers device emulation and interacts with the KVM kernel module via the /dev/kvm interface to provide accelerated execution using hardware virtualization extensions, such as Intel VT-x/AMD-V. Guest OSes (VMs) run applications and unmodified drivers with virtual network interfaces (VNICs) offered by QEMU.
* On the kernel level, TAP interfaces connect VNICs to a virtual bridge within the host OS. From the virtual bridge, connections get propagated through the Host NIC Driver into the physical NIC to complete network access. The KVM module sits atop the guest OS to guarantee near-native performance, for device emulation by QEMU is allowed while handling this matter. This design enables both good virtualization and efficient communication among guests' virtualized network systems.

2. 

* This flowchart illustrates the architecture of virtualization using a hypervisor. At the bottom, the hardware layer provides physical resources like CPU, memory, and network interfaces. Above the hardware, the Host Operating System manages the hardware resources and interacts with the hypervisor. The hypervisor acts as a mediator, creating and managing multiple Virtual Machines (VMs). Each virtual machine runs its own Guest Operating System (Guest OS 1, Guest OS 2, and Guest OS 3) independently, enabling the isolation of applications and processes.
* The hypervisor abstracts and allocates hardware resources to each VM, allowing multiple guest operating systems to share the same physical hardware while operating as isolated environments. This architecture ensures efficient utilization of resources and supports simultaneous execution of multiple VMs.

**Logical design**

1. User Input (Streamlit).
2. Passes VM parameters (name, CPU, RAM, etc.) to Flask API.
3. Flask API interacts with libvirt to create a VM.
4. Fetch IP address via virsh domifaddr.
5. API response returned to Streamlit for display.
6. Include process flows such as:

* VM creation
* IP fetching

**Pseudocode**

**Pseudocode for Server-Side (Flask API)**

1. Setup Flask Application

* Initialize Flask app.

1. Define Helper Functions

* get\_vm\_ip(vm\_name):

1. Run virsh domifaddr command to get the IP of the VM.
2. Return output or error.

* create\_vm(vm\_name, iso\_path, vcpu, ram, disk\_size):

1. Use virt-install to create a VM with given specifications.
2. Return success or error message.
3. Define API Endpoints

* POST /create\_vm:

1. Parse input (VM name, vCPU, RAM, Storage).
2. Validate input (VM name required).
3. Call create\_vm() to create a VM.
4. Wait for 5 seconds and call get\_vm\_ip() to fetch IP.
5. Return success response with VM IP or warning.

* GET /get\_vm\_ip:

1. Parse input (VM name).
2. Validate input.
3. Call get\_vm\_ip() to fetch IP.
4. Return the IP address or warning.
5. Run Flask App

* Start the server on localhost.

**Pseudocode for Client-Side (Streamlit App)**

1. Setup Streamlit Application

* Define API endpoints for VM creation and IP retrieval.

1. VM Creation Form

* On form submission: Setup Streamlit Application
* Define API endpoints for VM creation and IP retrieval.
* VM Creation Form
* Display input fields for: VM Name, vCPU, RAM, and Storage.
* On form submission:

1. Validate VM name.
2. Send POST request to /create\_vm API.
3. Show success or error message.
4. Fetch VM IP
5. Add a button to fetch IP.

* On click:

1. Send GET request to /get\_vm\_ip API with VM name.
2. Display raw IP information output in a text area.
3. Sidebar (About Section)
4. Add a short description of the app's functionality.
5. Run Streamlit App
6. Launch the Streamlit UI.Setup Streamlit Application
7. Define API endpoints for VM creation and IP retrieval.
8. VM Creation Form

* Display input fields for: VM Name, vCPU, RAM, and Storage.
* On form submission:

1. Validate VM name.
2. Send POST request to /create\_vm API.
3. Show success or error message.
4. Fetch VM IP
5. Add a button to fetch IP.

* On click:

1. Send GET request to /get\_vm\_ip API with VM name.
2. Display raw IP information output in a text area.
3. Sidebar (About Section)
4. Add a short description of the app's functionality.
5. Run Streamlit App

**Hardware and Software Platform**

**Hardware Platform**

The project is hosted on a strong hardware infrastructure with smooth virtualization and robust operation. Specifications are;

1. CPU: A multi-core processor with hardware virtualization support, such as Intel VT-x or AMD-V, to enable efficient execution of virtual machines.
2. RAM: With at least 8 GB in memory, 16GB more is recommended for performance as in the case of the run of multiple virtual machines in parallel.
3. Storage: An SSD with a minimum of 50GB free, to ensure fast read/writes and enough storage space for images of the VMs and their corresponding data.
4. Network: Local environment with DHCP enabled for simple, easy IP assignment between VMs and the host.

**Software Platform**

The software stack implements virtualization and user-facing services using Linux-based tools and modern programming frameworks. These are:

1. Host OS: Ubuntu 20.04 or later, for stability and compatibility with KVM/QEMU, and a strong community.
2. Virtualization Tools:

* KVM/QEMU: The core virtualization engine that provides efficient hardware-based virtualization.
* Libvirt: A management layer that simplifies virtual machine operations.
* virt-install: A command-line utility for creating and managing VMs.

1. Programming Tools:

* Python 3.x: Used for scripting automation tasks and backend logic.
* Flask: Lightweight web framework for backend services to handle user requests and VM operations.
* Streamlit: A Python-based framework for creating an interactive web-based front end that intuitively manages the VMs.

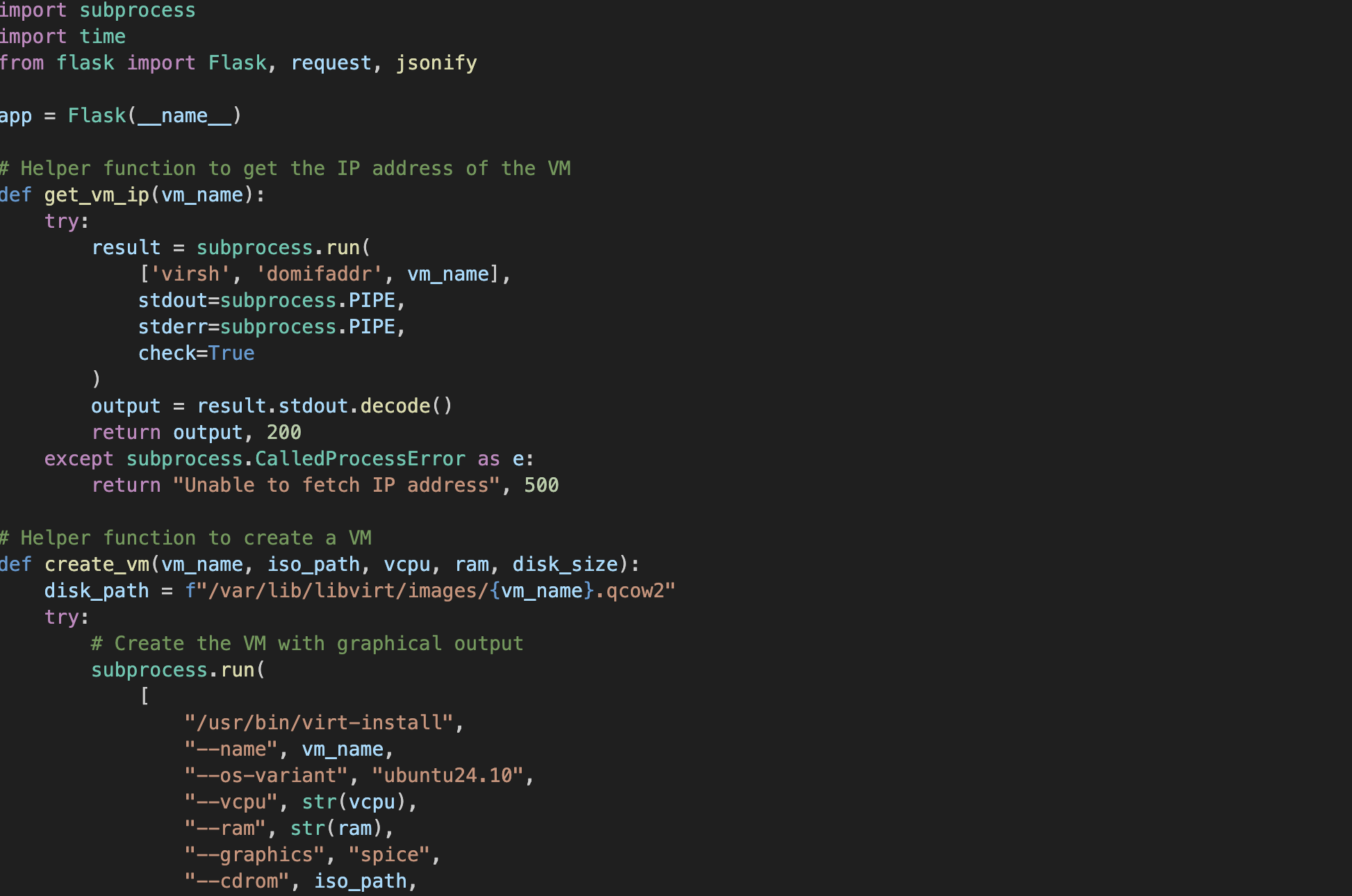
**List of Cloud Service (IaaS)**

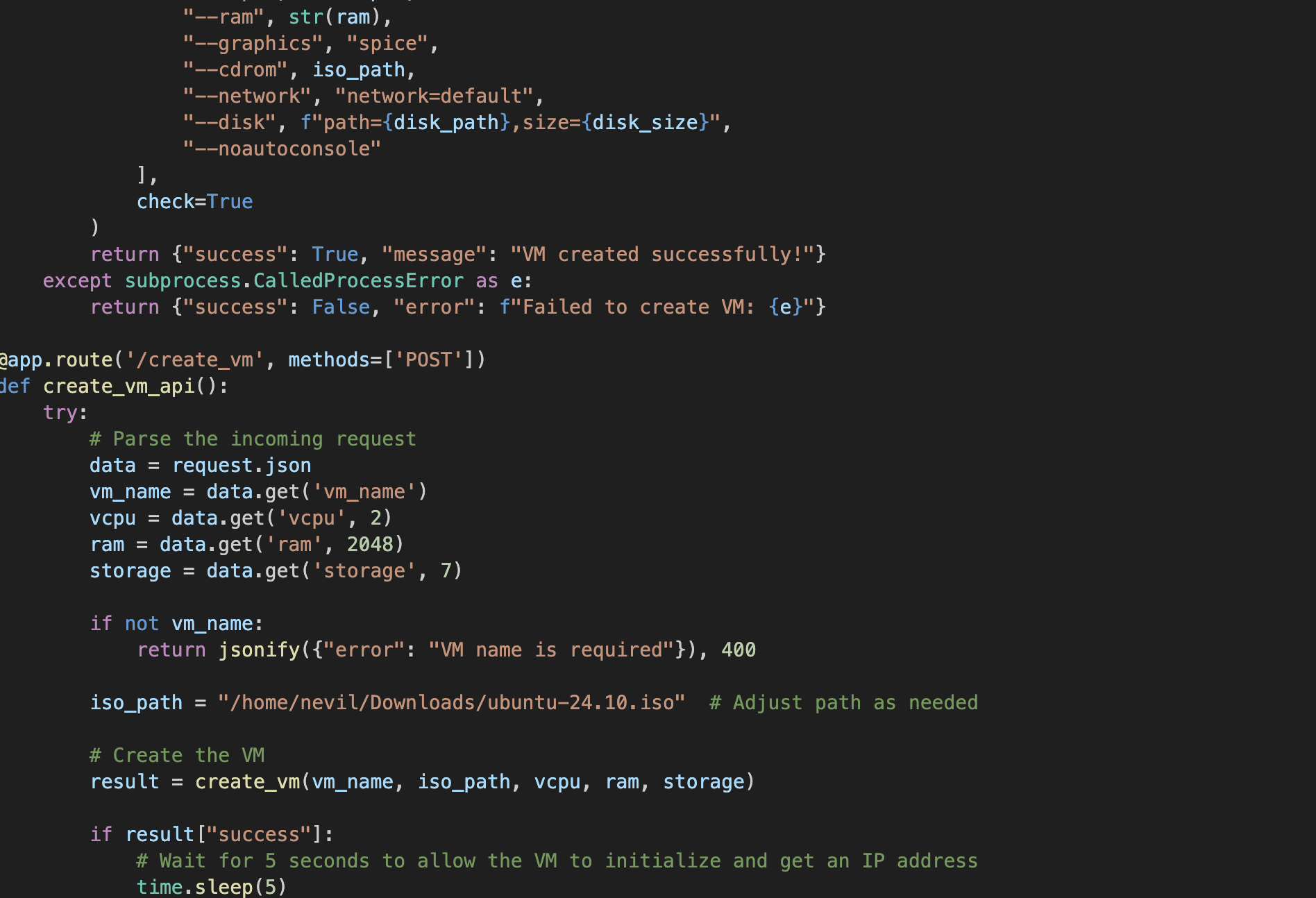
For this project, we have used Infrastructure as a Service (IaaS) as the predominant service model for the cloud. IaaS allows for dynamic provisioning of virtual machines, allowing its users to have scalable computing resources without having to spend and manage physical equipment in any way.

In IaaS, virtual machines can be configured to suit individual needs, for example, in terms of the number of CPU cores, the amount of memory, and the capacity of storage. This would ensure that the resources utilized by the VMs would be optimum and thereby efficiently support the various application and workload needs.

Our project demonstrates how one can create and manage virtual machines through IaaS in a cost-effective and scalable manner, thus aligned with the core principles of cloud computing.

**Sample Source Code**

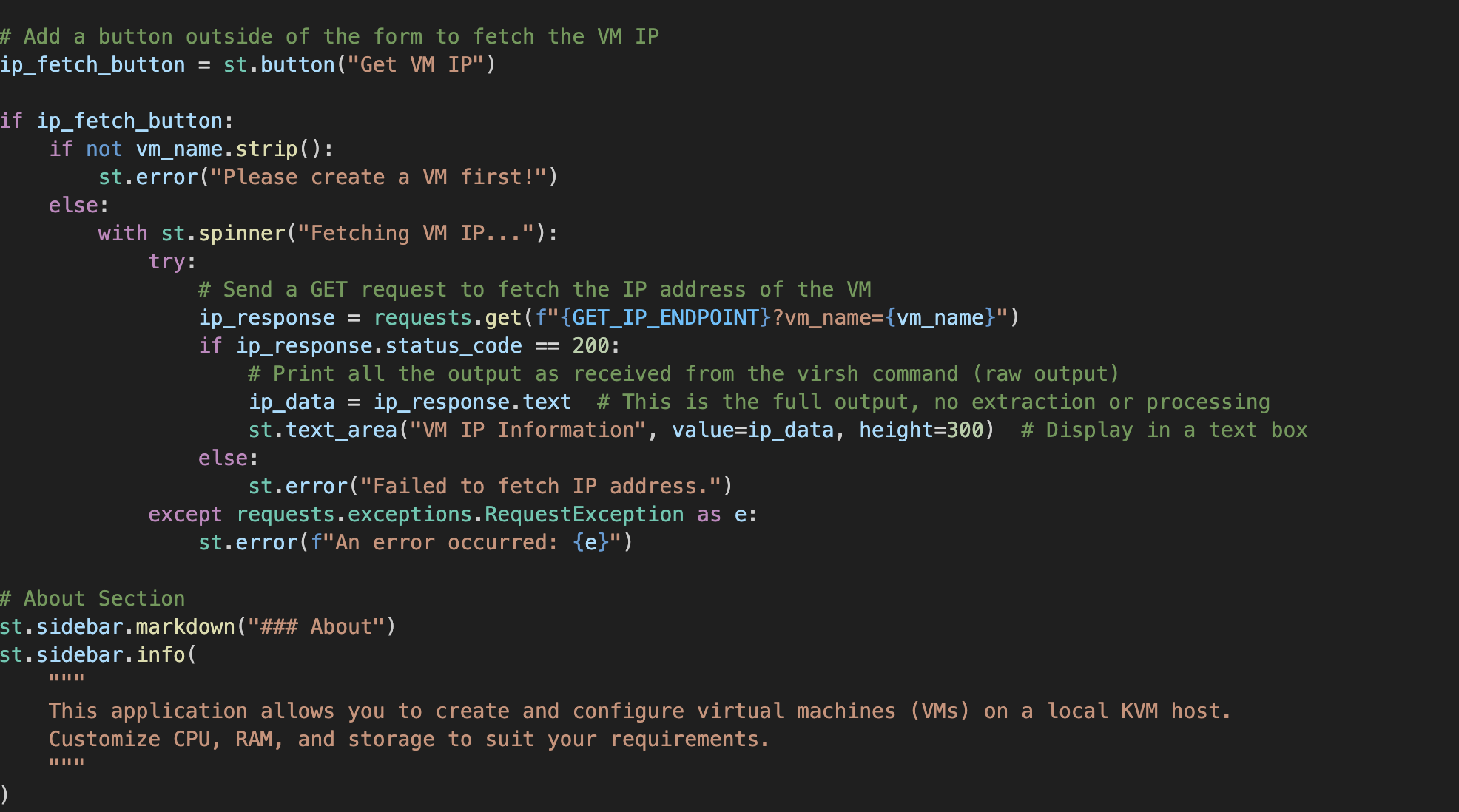




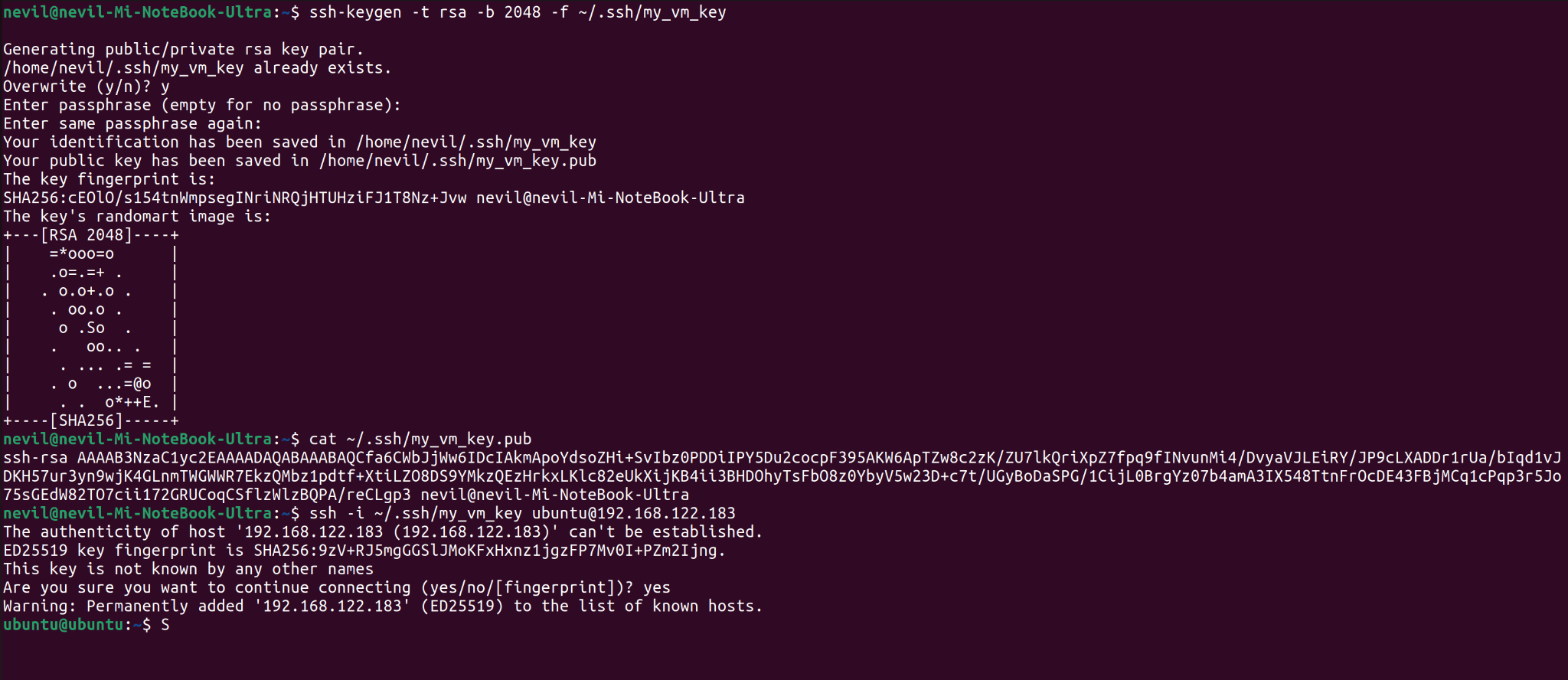




****



**Result**

****

**Challenges**

We faced a number of technical issues while implementing our project, which we had to change and adjust our strategy for:

1. Incompatible with Xen Hypervisor:

First, we tried to use the Xen Hypervisor for virtualization. However, because of hardware incompatibility and repeated system instability that caused reboots, we had to switch to KVM (Kernel-based Virtual Machine) as our virtualization platform. KVM's integration with the Linux kernel and compatibility with our hardware proved to be a more reliable choice.

1. Manual SSH Configuration:

We wanted to automate the installation and configuration of SSH, or Secure Shell, in the provisioning process of virtual machines for ease of remote access. But with the problems we had with scripting this automation within our environment, we fell back to manually installing and configuring SSH on each virtual machine. This added a whole extra layer of manual intervention to our workflow.

1. Difficulty in Finding Compatible ISO Images:

It was difficult to find the right ISO image for the guest operating systems. Some ISO images did not support our virtual machine configuration, while others were not optimized for virtualization. This led us to test several distributions and versions before finding the right ISO files, which delayed the setup process.

**Conclusion**

This project demonstrated the practical implementation of an Infrastructure as a Service (IaaS) platform, where KVM (Kernel-based Virtual Machine) is used for creating and managing virtual machines like services provided by Amazon EC2. Through this project, we gained invaluable insights into the underlying principles of cloud computing, virtualization, and resource allocation.

It is easy to overlook the problems while building a scalable and customizable virtual machine environment, but among the problems that cropped up were transitioning from Xen to KVM due to incompatibility with the hardware of the system, manually configuring SSH, and finding suitable ISO images.

The results validate the functionality and efficiency of the IaaS platform implemented by supporting the creation of a variety of virtual machines as desired, such as for different CPU, memory, or storage configurations. Therefore, this opens the great opportunity of IaaS-based delivery of flexible and cost-effective computing solutions in academic and professional scenarios alike.

In conclusion, this project not only met its goals but also improved our technical skills in virtualization technologies and cloud computing frameworks, thus serving as a good foundation for future ventures in cloud-based infrastructures.

**Reference**

(n.d.). Virtual Machine Manager. Retrieved December 16, 2024, from <https://virt-manager.org/>

Jain, S. (2022, March 26). *How to Create Virtual Machines in Linux Using KVM (Kernel-based Virtual Machine)?* GeeksforGeeks. Retrieved December 16, 2024, from <https://www.geeksforgeeks.org/how-to-create-virtual-machines-in-linux-using-kvm-kernel-based-virtual-machine/>

*Virtualisation with QEMU*. (n.d.). Ubuntu. Retrieved December 16, 2024, from <https://ubuntu.com/server/docs/virtualisation-with-qemu>

*What is IaaS (infrastructure-as-a-service)?* (n.d.). Cloudflare. Retrieved December 16, 2024, from <https://www.cloudflare.com/learning/cloud/what-is-iaas/>

<https://www.researchgate.net/figure/The-architecture-of-KVM-full-virtualized-network-I-O_fig2_294105897>

<https://www.geeksforgeeks.org/how-to-run-multiple-virtual-machines-windows-and-linux-inside-virtualbox/?ref=asr2>