Cloud Management System – Phase 1 Report

Malak Mohamed (223841), Hossam Abdelreheem (224527), Abdelrahman Elshazly (224657), Tarek Amr (220509), Farah Walid (222307), Linah Mohamed (234877)

Submission Date: 04/05/2025

Project Overview

The Cloud Management System project is designed to give users an intuitive and powerful platform to manage virtualization tasks such as creating and operating virtual machines (VMs) and working with Docker containers. In Phase 1, the system focuses on:

- Allowing users to create and manage virtual disks that serve as storage backends for virtual machines.
- Enabling users to create virtual machines with customized configurations, including CPU, memory, disk selection, and ISO files.

By combining QEMU, a well-known virtualization tool, with Python(PyQt5) for graphical user interface, the system bridges advanced virtualization capabilities with user-friendly interaction. This approach prepares students for real-world cloud management scenarios.

Implemented Features

- Virtual Disk Management

1. Purpose: To provide users the ability to create storage disks of different formats that can later be attached to virtual machines.

2. Features Implemented:

- Support for dynamic or fixed disk allocation.
- Broader disk format support: qcow2, raw, img, vmdk, vdi, vhd, vhdx, qcow, cow, cloop, dmg.
- Automatically updates disk extension when disk type changes.
- Disk Creation: Users can specify the size (with units like GB, MB, or TB) and path.
- Disk Resizing: Disks can be expanded in size (shrinking is prohibited for safety reasons).

3. Technical Details:
- Uses ***qemu-img*** commands through Python.
- Input validation ensures that users provide valid sizes, formats, and paths.
- Error messages guide users when invalid inputs are detected.

4. User Benefits: Users with little or no experience with command-line virtualization tools can perform complex disk management operations with just a few clicks.

- Virtual Machine Management

1. Purpose: To allow users to create fully customizable virtual machines that can boot from ISO images and use previously created virtual disks.

2. Features Implemented:
- Configuration Options: Users can specify: VM name, number of CPU cores (1 to 4), memory size (between 512 MB and 32 GB), virtual disk selection, and the ISO file for operating system installation.

3. Technical Details:
- Uses Python to interact with QEMU.
- Performs input validation to prevent misconfiguration.
- Handles missing file paths, invalid numeric values, and empty fields properly.

4. User Benefits: Even without prior experience in virtualization, users can create fully functional virtual machines suitable for operating system testing, software development, and educational purposes.

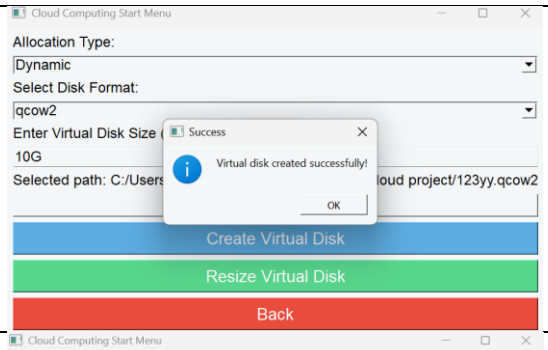
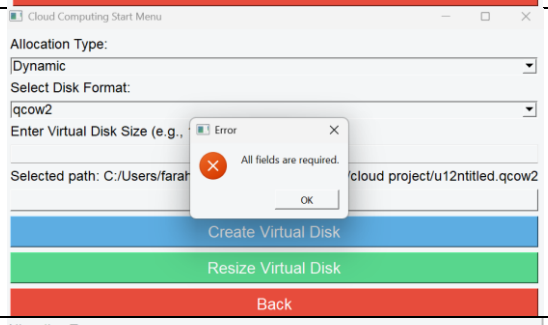
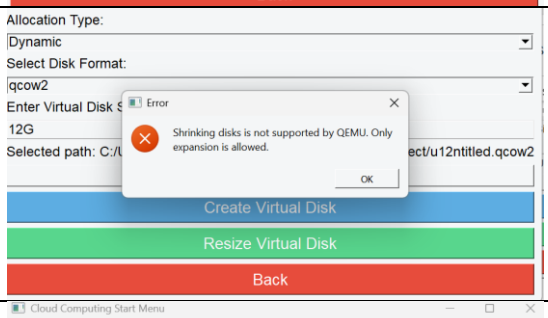
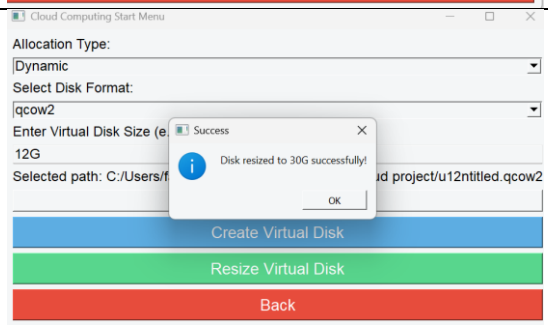
GUI Overview

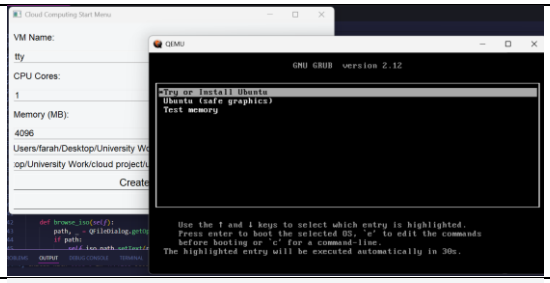
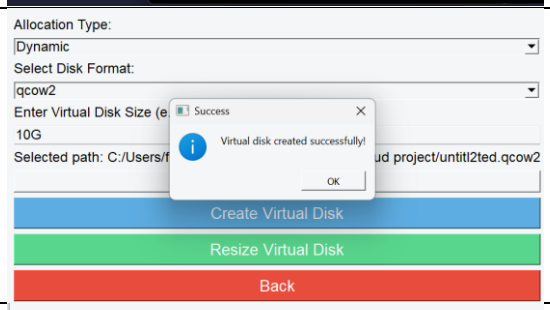
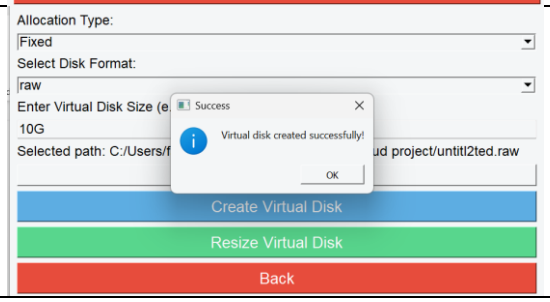
The GUI is built using PyQt5, offering a quick and responsive design, eliminating the need for users to interact with the command line.

Page	Purpose
Start Menu	Welcome screen to start or exit the application.
Project Menu	Hub to access Virtual Disk Management and VM Configuration.

Virtual Disk Management	Interface to create and resize virtual disks.
Virtual Machine Configuration	Interface to specify VM parameters and create new virtual machines.

Testing Section

Test Case	Action	Expected Result	Screenshot
Disk Creation –Valid Input	10G qcow2 disk creation	Disk created successfully	
Disk Creation –Valid Input	10G qcow2 disk creation	Disk created successfully	
Disk Creation – Missing Size	Attempted creation with no size	Error message displayed	
Disk Resize – Expansion	Increased 10G disk to 30G	Disk resized successfully	

VM Creation – Complete Parameters	Created VM with 2 cores, 4GB RAM, disk, and ISO	VM launched successfully	
Disk Creation – Dynamic Allocation	Created a qcow2 disk	Disk created successfully	
Disk Creation – Fixed Allocation	Created a raw disk with fixed allocation	Disk created successfully	

User Manual

1. **System Requirements**

- **Operating System:** Windows 10/11
- **Python Version:** 3.8 or higher
- **QEMU Executables Location:**
  C:/msys64/ucrt64/bin/qemu-img.exe
  C:/msys64/ucrt64/bin/qemu-system-x86_64.exe

2. **Starting the Application**

- Launch the script CloudPhase1.py.
- The **Start Menu** will appear, offering the option to begin or exit.

3. **Virtual Disk Management**

Creating a Virtual Disk:

1. Input desired size (e.g., 10G for 10 gigabytes).
2. Select the disk type (qcow2, raw, vmdk).
3. Choose a destination file path.
4. Click Create Virtual Disk.
5. Allocation Type:

Dynamic → Space allocated as needed (default for formats like qcow2).

Fixed → Space fully allocated upfront (supported for raw, vmdk, vdi).

Resizing a Disk:

1. Select an existing disk.
2. Enter a larger size.
3. Click Resize Virtual Disk: Shrinking is not permitted to prevent data loss.

4. **Virtual Machine Configuration**

Steps to Create a VM:

1. Enter a VM name.
2. Specify CPU cores (1-4).
3. Set memory (512 MB – 32 GB).
4. Select a virtual disk from the list.
5. Browse for an ISO file.
6. Click Create Virtual Machine.

The VM will launch in a separate QEMU window.

Challenges

- Challenges Faced and Solutions for Virtual Disk Creation:

1. Incorrect or Missing Disk Path

➔ Challenge:

Users might not provide a valid or correctly formatted file path.

➔ Solution:

GUI uses a file dialog (QFileDialog) to guide path selection.

2. Invalid Disk Size Input

➔ Challenge:

Users might enter a size without units or with special characters.

➔ Solution:

Size is validated to include a unit (G, M, or T).

Custom logic checks for and blocks special characters using special_chars.

Displays an error dialog (QMessageBox) if validation fails.

- Challenges Faced and Solutions for Virtual Machine Creation:

1. Validating User Inputs for VM Configuration

➔ Challenge:

Users might enter invalid VM names (e.g., starting with digits or including special characters), invalid memory sizes, or forget to provide ISO/virtual disk paths.

➔ Solution:

Implemented robust input validation checks:

Disallowed VM names starting with digits or containing special characters.

Checked for required fields like ISO and disk path before proceeding.

Provided real-time user feedback via QMessageBox for better usability.

2. Supporting User-Friendly File Selection

➔ Challenge:

Manually typing ISO or virtual disk paths is error-prone.

➔ Solution:

Implemented file browsers using QFileDialog.getOpenFileName() for both ISO and disk selection, making it user-friendly and reducing file path errors.

3. Error Handling for VM Launch Failures

➔ Challenge:

If QEMU fails to start (due to missing files or permission issues), it wasn't clear to users why it failed.

➔ Solution:

Wrapped the QEMU launch command in a try-except block and showed the exception message via a message box to assist in debugging: except Exception as e:QMessageBox.critical(self, "Error", f"Failed to launch VM:\n{e}")