Project Documentation: Cloud Management System (CMS)

Course: Cloud Computing & Networking – CSCI363

Student Name	ID
Adham Ahmed Mekky	231001296
Adham Ahmad Abdelaal	231000313
Amr Sherif Nabil	231000045
Belal Mohammed Amer	231000272
Youssef Helmy Ibrahim	231000834

1. Overview

The **Cloud Management System (CMS)** is a hybrid orchestration platform designed to streamline the management of cloud resources. Built on Python, the system leverages WSL 2 (Windows Subsystem for Linux) to interface directly with the Linux kernel, providing a unified dashboard for:

- **Infrastructure as a Service (IaaS):** Managing Virtual Machines via QEMU/KVM.
- **Container as a Service (CaaS):** Managing Docker containers via the Docker Engine.

The project features a modern Graphical User Interface (GUI) for primary operations and a legacy Command Line Interface (CLI) as a fail-safe backup.

2. System Architecture & Design

The system follows a modular Model-View-Controller (MVC**)** pattern to ensure separation of concerns and crash-proof stability.

2.1 Tech Stack

- **Language:** Python 3.10+
- **GUI Framework:** customtkinter (Modern, High-DPI aware interface).
- **Virtualization Backend:** subprocess module bridging to QEMU/KVM.

- **Container Backend:** Docker SDK for Python (docker library).
- **Environment:** Ubuntu 22.04 running on WSL 2.

2.2 File Structure

The project is organized to separate logic from presentation:

- **gui_main.py:** The primary entry point. Handles the GUI event loop, threading, and user input validation.
- **vm_manager.py:** Backend logic for IaaS. Handles disk creation (qemu-img) and VM execution (qemu-system-x86_64).
- **docker_manager.py:** Backend logic for CaaS. Handles API calls to the Docker Daemon.
- **cli_main.py (Backup):** A text-based menu system for headless environments.
- **VM_Storage:** Dedicated directory for storing Virtual Hard Drives (.qcow2).
- **Docker_Projects:** Workspace for saving generated Dockerfiles.

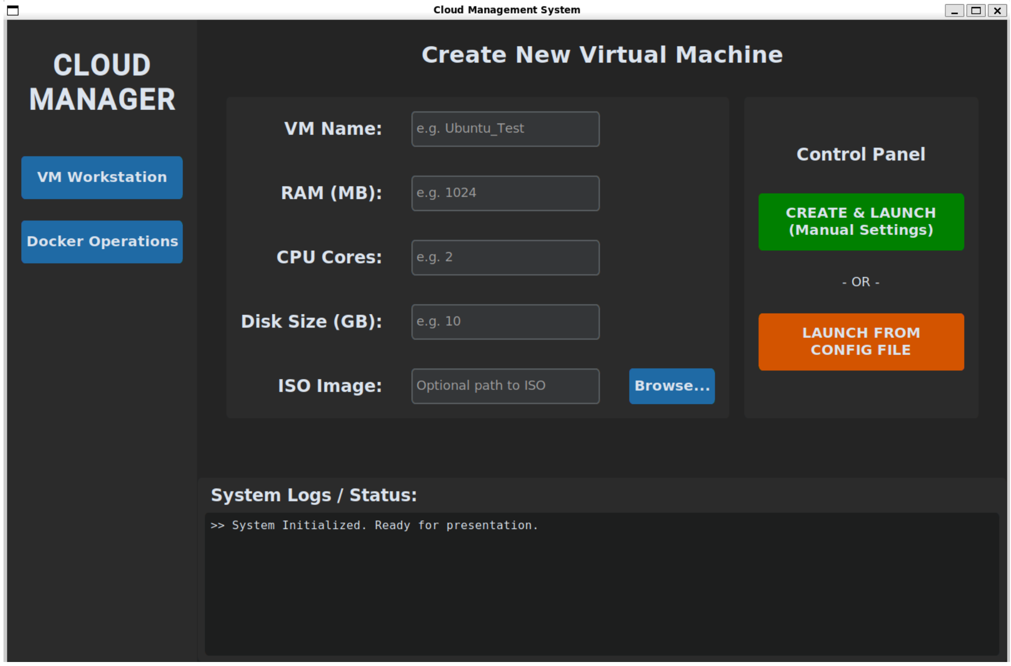
3. Feature Documentation (GUI)

The Graphical User Interface is the primary mode of operation, offering a "Single Pane of Glass" for all cloud tasks.

3.1 VM Workstation (IaaS)

This module provides full control over hardware emulation.

- **Manual Provisioning:** Users can specify exact hardware parameters:
  - **RAM:** Allocated memory (e.g., 2048 MB).
  - **CPU:** Processor core assignment (e.g., 2 Cores).
  - **Disk:** Dynamic storage allocation (e.g., 10 GB).
  - **ISO Boot:** Integrated file browser to select OS installers.
- **Automated Provisioning:** The "Launch from Config" feature reads a JSON file (vm_config.json) to instantly deploy a pre-defined VM standard, mimicking real-world Infrastructure as Code practices.



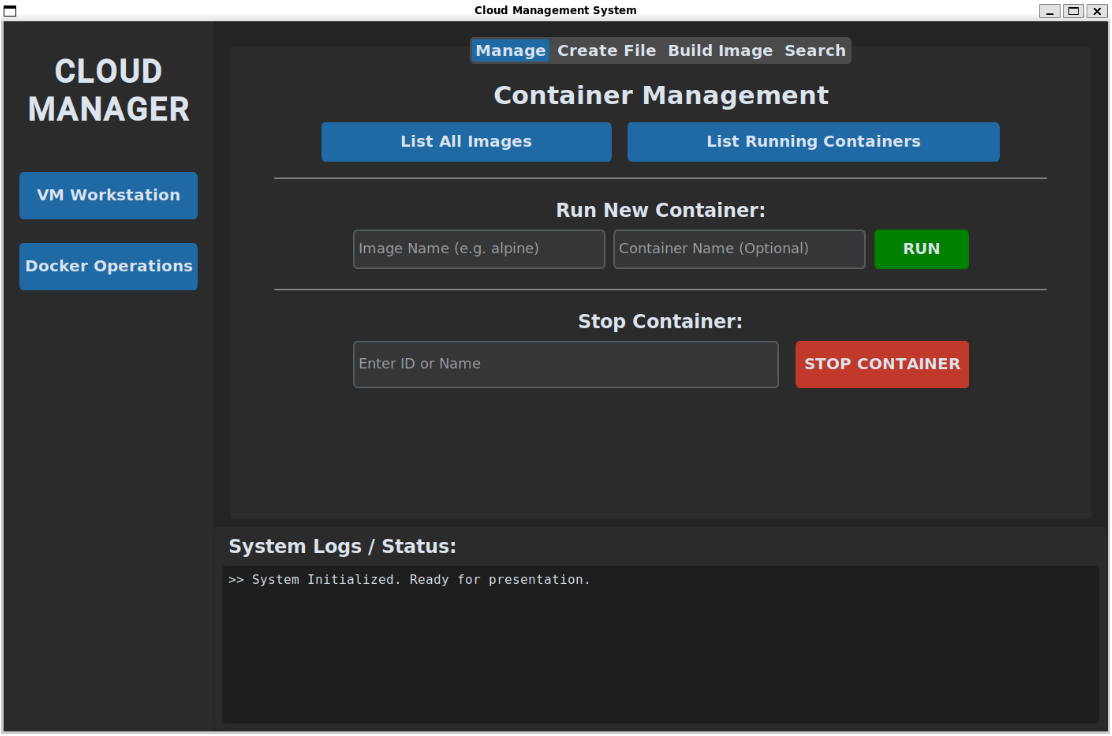
```
>> System Initialized. Ready for presentation.
```

3.2 Docker Operations (CaaS)

This module manages the full container lifecycle without requiring terminal access.

Tab 1: Management Dashboard

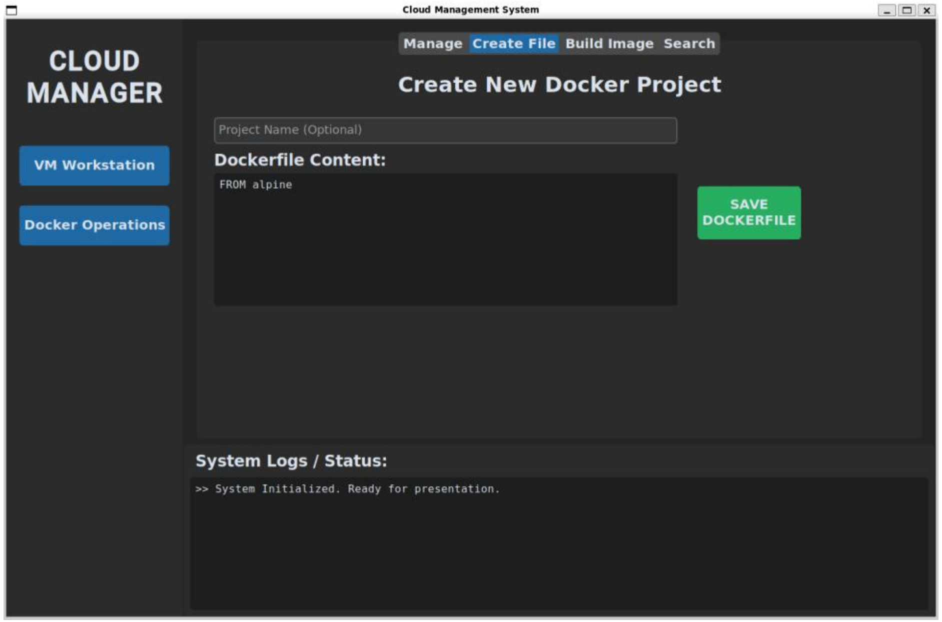
- **Real-Time Monitoring:** Displays tables of available Images and Running Containers with details (ID, Status, Size).
- **Lifecycle Control:**
  - **Run:** Launch containers in the background (-d) with custom names.
  - **Stop:** Forcefully halt containers using their ID or Name.



```
>> System Initialized. Ready for presentation.
```

Tab 2: Development Environment

- **Integrated Editor:** A text editor area to write Dockerfile code.
- **Project Management:** Automatically creates structured project folders inside Docker_Projects/ and saves the file.



Create New Docker Project

Project Name (Optional)

Dockerfile Content:

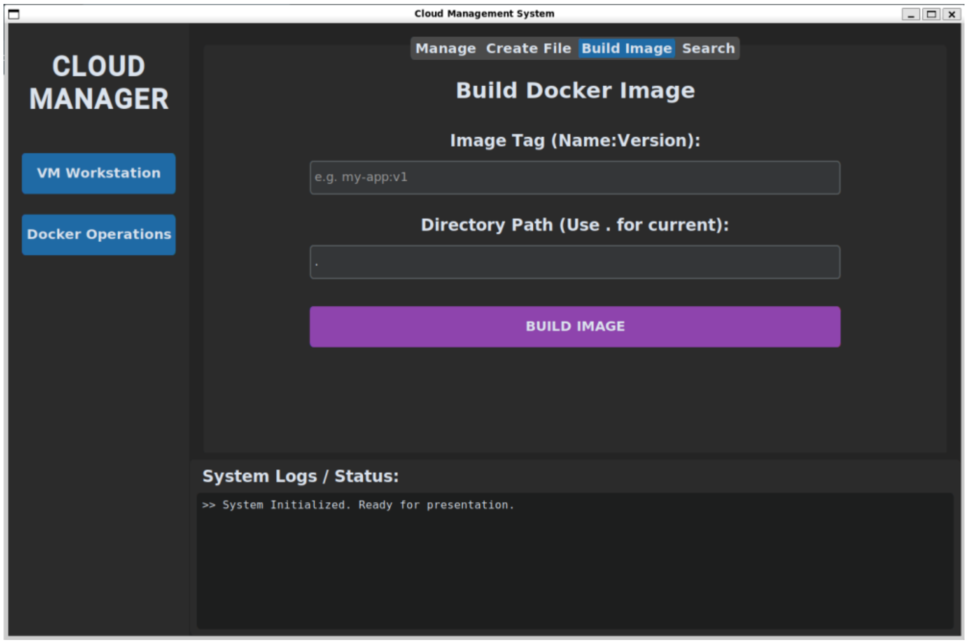
```
FROM alpine
```

System Logs / Status:

```
>> System Initialized. Ready for presentation.
```

Tab 3: Build System

- **Image Compilation:** Converts the project folder into a runnable Docker Image.
- **Live Logs:** Streams the build process (Step 1/2, Step 2/2) directly to the system console for debugging.



Build Docker Image

Image Tag (Name:Version):

e.g. my-app:v1

Directory Path (Use . for current):

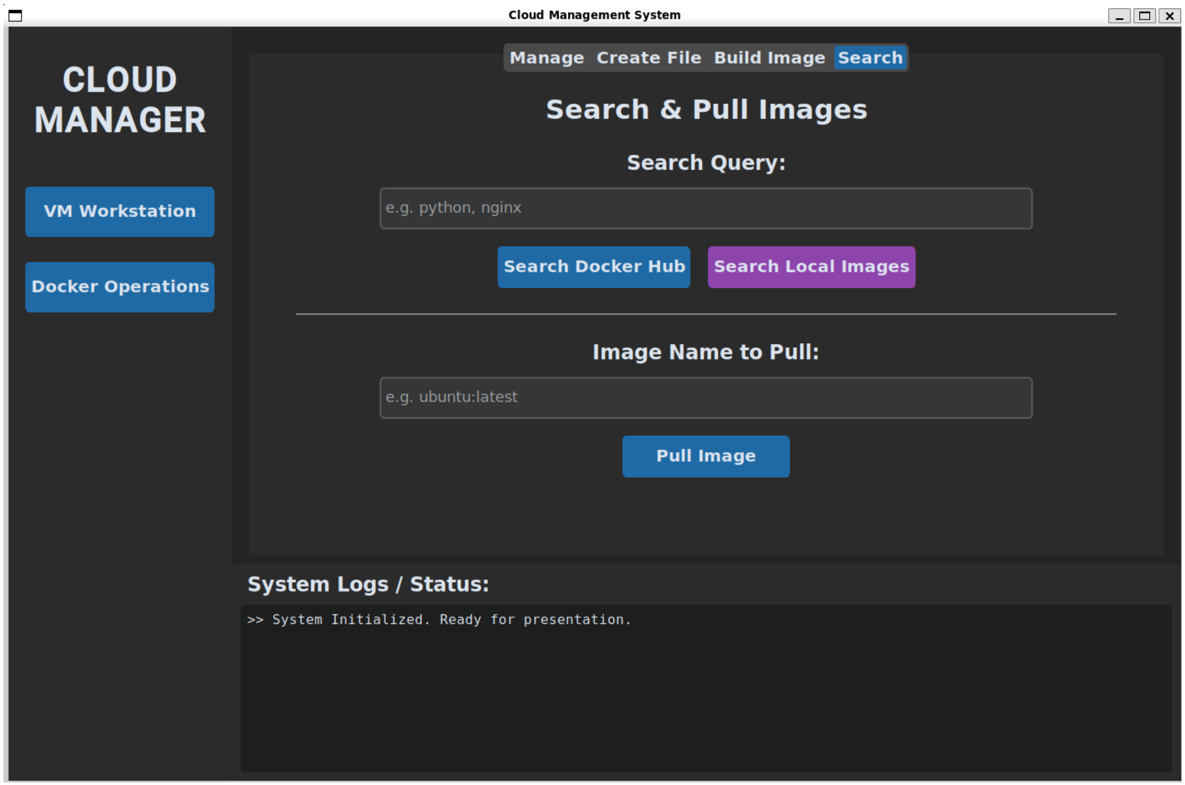
.

System Logs / Status:

```
>> System Initialized. Ready for presentation.
```

Tab 4: Registry & Search

- **Docker Hub Search:** Queries the global registry for public images.
- **Local Filter:** A "Smart Search" feature that filters the local library to find specific downloaded images.
- **Pull:** Downloads images from the cloud to the local machine.



```
>> System Initialized. Ready for presentation.
```

4. Testing & Validation

The system underwent rigorous testing to ensure stability and error handling.

Test Case 1: VM Crash Prevention

Scenario: Launching a VM with excessive RAM (e.g., 8GB on a 4GB laptop).

Result: The system catches the resource contention. The use of

Python threading prevents the GUI from freezing while the backend negotiates resources.

Test Case 2: Docker Build Workflow

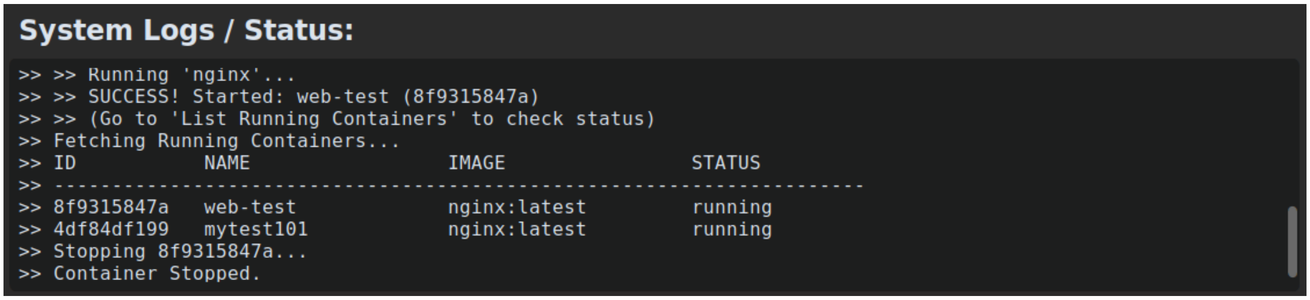
Action: Created a Dockerfile (FROM alpine) -> Saved to Project -> Built as test:v1.

Outcome: The build succeeded, logs displayed "Success," and the image immediately appeared at the top of the "List Images" table (verified by the date-sorting algorithm).

Test Case 3: Container Lifecycle

Action:

1. **Run:** Started nginx named web-test.
2. **Verify:** Confirmed web-test appeared in the "Running Containers" list.
3. **Stop:** Input web-test and clicked Stop.
4. **Result:** The container was successfully terminated and removed from the active list.



System Logs / Status:

```
>> >> Running 'nginx'...
>> >> SUCCESS! Started: web-test (8f9315847a)
>> >> (Go to 'List Running Containers' to check status)
>> Fetching Running Containers...
>> ID           NAME                 IMAGE                STATUS
>> ----------------------------------------------------------------------
>> 8f9315847a   web-test             nginx:latest         running
>> 4df84df199   mytest101            nginx:latest         running
>> Stopping 8f9315847a...
>> Container Stopped.
```

5. Backup Interface (CLI)

A legacy Command Line Interface (cli_mode.py) is included as a fail-safe measure.

- **Purpose:** Provides access to core functions in "Headless" environments where no display is available.
- **Capabilities:**
    - Create/Launch VMs.
    - List Images and Containers.
    - Pull Images.
- **Limitation:** Does not support the advanced "Run Container" or "File Editor" features found in the GUI.

6. Conclusion

The Cloud Management System successfully demonstrates the principles of cloud orchestration. By wrapping complex CLI tools (QEMU, Docker) in a user-friendly Python interface, it lowers the barrier to entry for managing virtualization technologies. The project meets all functional requirements, including IaaS/CaaS management, persistent storage organization, and robust error handling.