**Project and Testing Report: Cloud Management System (CMS)**

**Introduction**

The Cloud Management System (CMS) is a Python-based GUI application designed to streamline the management of virtual machines and Docker containers. It provides features for creating and managing virtual machines using QEMU and facilitates Docker container lifecycle operations, including creating Dockerfiles, building images, managing containers and interacting with DockerHub. This document provides an overview of the design choices, challenges faced, solutions implemented, testing methodologies, and performance evaluation of the system.

**1. Design Choices**

**1.1 Technology Stack**

* **Programming Language**: Python was chosen for its extensive libraries and ease of integration with external tools like QEMU and Docker.
* **GUI Framework**: Tkinter was selected for its simplicity and native support in Python, ensuring platform-independent functionality.
* **Virtualization**: QEMU was utilized for VM creation due to its open-source nature and robust features.
* **Containerization**: Docker SDK for Python enabled seamless Docker management within the application.

**1.2 Features**

The following core features were implemented based on user requirements:

1. **Virtual Machine Management**:
   * Creation of new VMs with user-defined configurations.
   * Management of existing VMs using QCOW2 disk images.
2. **Docker Management**:
   * Creation of Dockerfiles.
   * Building Docker images.
   * List Docker images and running containers.
   * Stop running containers.
   * Search for Docker images locally and on DockerHub.
   * Download Docker images from DockerHub.
   * Run a new Docker container from an existing image.
3. **Logging**:
   * Integrated a logging module for capturing user interactions and system errors, enhancing traceability.

**1.3 User Experience Enhancements**

* Clear navigation through submenus and windows for different tasks.
* Dynamic input validation for user-provided data.
* Dynamic field enabling/disabling based on user’s selection.
* Use of file dialog windows for selecting paths and files, improving usability.

**2. Challenges Faced and Solutions Implemented**

**2.1 Management of QEMU**

* **Challenge**: Executing QEMU commands and managing disk images required careful validation of user inputs.
* **Solution**: Input fields were validated dynamically, and error messages were displayed using Tkinter’s messagebox. QEMU command execution was streamlined with proper error handling.
  1. **Integrating QEMU and Docker**
* **Challenge**: Managing two different virtualization technologies required careful coordination of dependencies and system resources.
* **Solution**: Modularized the code to separate QEMU and Docker functionalities, ensuring independent operation without conflicts.

**2.3 Logging Implementation**

* **Challenge**: Debugging required a robust logging mechanism.
* **Solution**: The logging module was configured to log messages at different levels (INFO, WARNING, ERROR) into a file for later debugging and analysis.

**2.4 File Path Selection for Disk Images**

* **Challenge**: Ensuring users can intuitively select paths for saving and accessing disk images.
* **Solution**: Introduced new file dialog windows for:
  + Choosing paths to save newly created disk images.
  + Browsing for existing QCOW2 or ISO files.

**3. Testing Methodologies**

**3.1 Unit Testing**

* **Objective**: Validate individual components, such as file path handling and input validation.
* **Test Cases**:
  + Validating numerical inputs for CPU count, memory size, and disk size.
  + Checking file existence and correct file type for disk images.

**3.2 Integration Testing**

* **Objective**: Ensure seamless interaction between GUI components, QEMU, and Docker SDK.
* **Process**:
  + Tested workflows such as VM creation and Docker image management.
  + Verified the transition between different GUI windows.
  + Verified that Docker and QEMU commands executed as expected without conflicts.

**3.3 System Testing**

* **Objective**: Test the complete application under real-world scenarios.
* **Approach**:
  + Simulated user interactions for creating VMs and managing Docker containers.

**3.4 Logging Verification**

* **Objective**: Confirm the accuracy of logged events.
* **Process**:
  + Triggered various actions to generate logs.
  + Reviewed log files for consistency and completeness.

**3.5 Performance Testing**

* **Approach**:
  + Measured system responsiveness and resource utilization during VM creation and Docker image builds.

**4. System Performance Evaluation**

**4.1 Test Environment**

* **Hardware**: Intel i7 processor, 16GB RAM, SSD storage.
* **Operating Systems**: Ubuntu 20.04.6 and Windows 11.

**4.2 Results**

1. **Virtual Machine Creation**:
   * New VM creation completed within 301.39 seconds.
   * Existing image-based VM creation executed flawlessly.
2. **Docker Operations**:
   * Dockerfile creation and image building tasks performed without errors.
   * Container listing and stopping operations were instantaneous.
3. **Error Handling**:
   * Errors (e.g., invalid inputs, missing dependencies) were captured accurately in logs and displayed to the user.
4. **GUI Responsiveness**:
   * Smooth navigation between windows.
   * No significant delays during interactions.

**5. Conclusion**

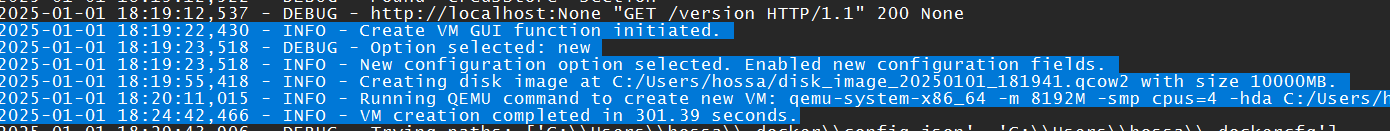
The Cloud Management System successfully combines virtual machine and Docker management into a unified GUI application. Despite challenges, the project achieved its objectives by leveraging robust design principles, careful testing, and iterative improvements. Future enhancements will focus on expanding functionality and further improving user experience.

**6. Appendices**

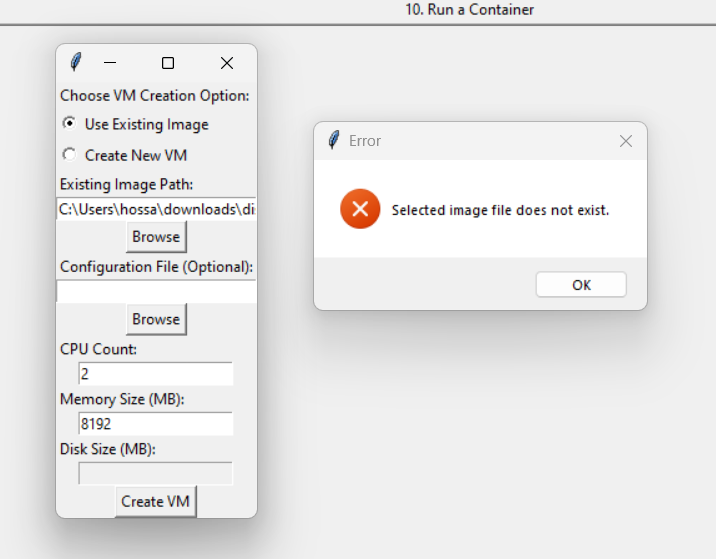
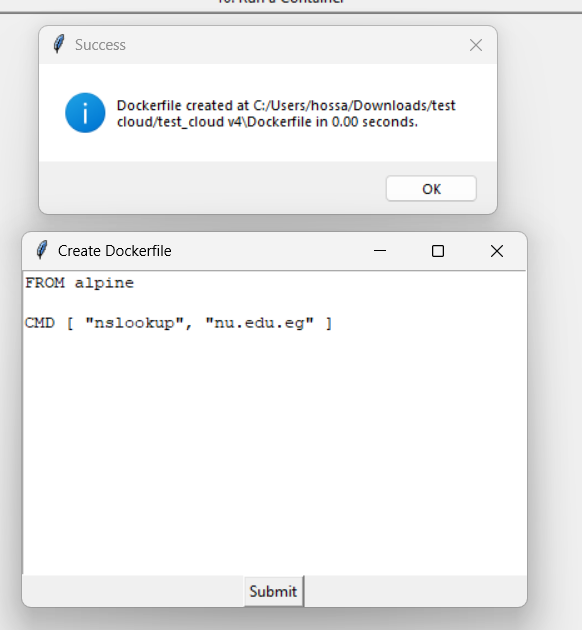
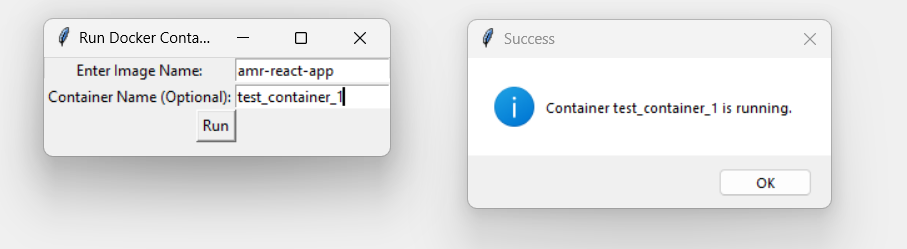
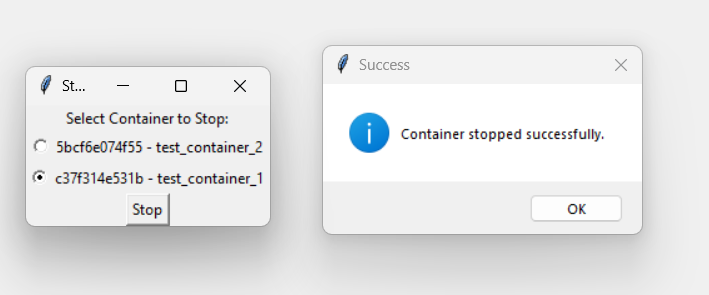
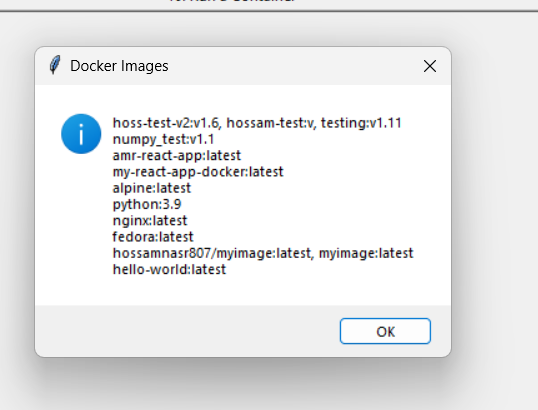
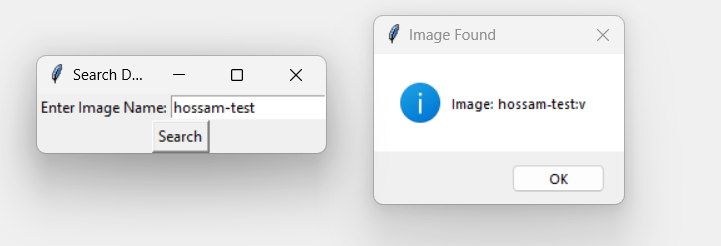
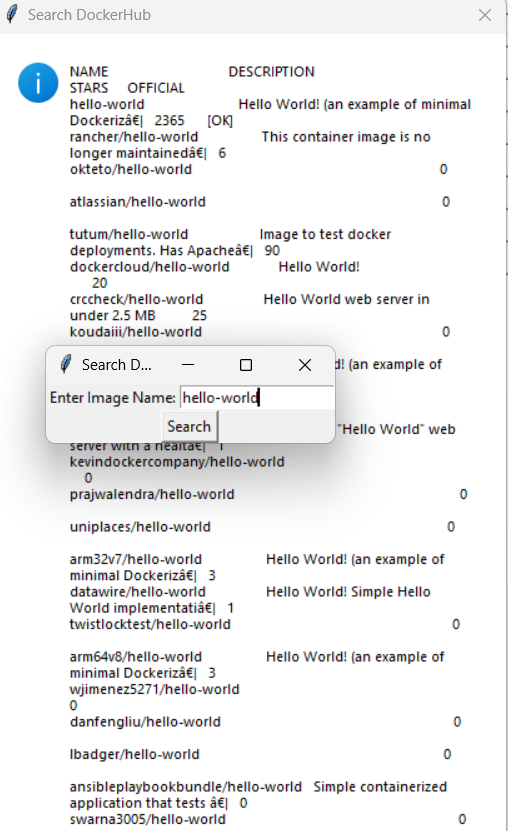
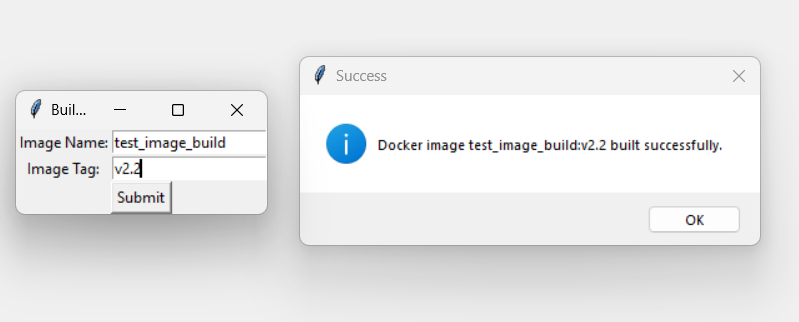
6.1 Sample Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Expected Result** | **Actual Result** | **Status** |
| TC-001 | Input CPU as a negative number | Error message displayed | As Expected | Passed |
| TC-002 | Select non-existent QCOW2 file for VM creation | Error message displayed | As Expected | Passed |
| TC-003 | Create a Dockerfile in a valid directory | Dockerfile saved successfully | As Expected | Passed |
| TC-004 | Stop a running Docker container | Container stopped successfully | As Expected | Passed |
| TC-005 | Transition between main menu and submenus | Smooth transition without errors | As Expected | Passed |
| TC-006 | Execute QEMU command for new VM creation | VM created successfully | As Expected | Passed |
| TC-007 | List available Docker images | All images listed correctly | As Expected | Passed |
| TC-008 | Search for a specific Docker image locally | Correct image displayed or "not found" message | As Expected | Passed |
| TC-009 | Search for Docker images on DockerHub | Relevant results displayed | As Expected | Passed |
| TC-0010 | Download a Docker image from DockerHub | Image downloaded successfully | As Expected | Passed |
| TC-011 | Run a new Docker container from an existing image | Container started successfully | As Expected | Passed |
| TC-012 | Verify logs for a completed VM creation | Logs accurately reflect the creation process | As Expected | Passed |
| TC-013 | Build a docker image from an existing Dockerfile | Image downloaded successfully | As Expected | Passed |
| TC-014 | Measure performance during Docker image build | Build completed within acceptable time limits | As Expected | Passed |
| TC-015 | Verify usability of file selection dialogs | File dialogs open correctly and accept valid inputs | As Expected | Passed |
| TC-016 | Measure performance during new VM creation | VM created successfully within 301.39 seconds | As Expected | Passed |
| TC-017 | Measure performance during existing image VM creation | VM created successfully within 124.88 seconds | As Expected | Passed |

6.2 log samples



6.2 Screenshots of Test Cases:



A screenshot of a computer error

Description automatically generated

A screenshot of a computer

Description automatically generated

