

**Project and Testing Report for the virtual machine management system**

CSCI363: Cloud Computing and Networking-2024FALL

Farah mohamed ibrahem 221001592

Jana ayman 221000410

Amenah medhat 221001792

Shorouk sherif 22100064

Salma Essam Mahdy 211001778

**Introduction**

The Virtual Machine Management System is a Python-based application leveraging Docker and QEMU to provide a user-friendly interface for managing virtual machines and Docker containers. The system is designed to assist users in efficiently handling cloud resources through an intuitive GUI built with Tkinter. This report details the design choices, challenges faced, implemented solutions, testing methodologies, and system performance evaluation.

**Design Choices**

**Technology Stack**

1. **Programming Language**: Python, chosen for its ease of use and the many libraries it has for us to use in the management system.
2. **Docker Integration**:
   * Utilized the Docker Python library to manage containers and images.
   * Included functionalities such as creating Dockerfiles, building and pulling images, and listing running containers.
3. **QEMU Integration**:
   * Leveraged OS system commands to integrate QEMU functionalities for creating and booting virtual machine images.
4. **GUI Framework**: Tkinter, selected for its simplicity and rapid development capabilities.

**Challenges Faced and Solutions Implemented**

**1. Challenge: Searching in the local images and Docker Hub**

**Issue**: Miscommunication with the Docker daemon due to incorrect configurations when using the Docker Python API.

**Solution**: Using shell commands Instead of Docker API directly.

**2. Challenge: Pulling an image from Docker Hub**

**Issue**: We couldn’t pull an image from Docker Hub and download it in our local images.

**Solution**: Using shell commands Instead of Docker API directly

**3. Challenge: File and Path Management**

**Issue**: Saving files like Dockerfiles to fixed paths was prone to errors.

**Solution**: Allowed dynamic path selection using file dialogs and added error handling to ensure file operations are secure.

**Testing Methodologies**

**1. Unit Testing**

* Verified individual functionalities such as image creation, container listing, and QEMU boot commands.
* Used mock inputs to simulate user actions.

**2. System Testing**

* Conducted end-to-end testing by performing real-world tasks:
  + Creating a Dockerfile, building an image, and running containers.
  + Booting a virtual machine using QEMU.

**Test Cases and Evidence**

**Test Case 1: Create and Save Dockerfile**

* **Input**: Write sample content in the Dockerfile editor and save.
* **Expected Output**: File is saved at the specified location with correct content.
* **Result**: Passed (validated by checking file content).

**Test Case 2: Build Docker Image**

* **Input**: Select a Dockerfile and specify an image name.
* **Expected Output**: Image is built successfully with logs confirming the operation.
* **Result**: Passed.

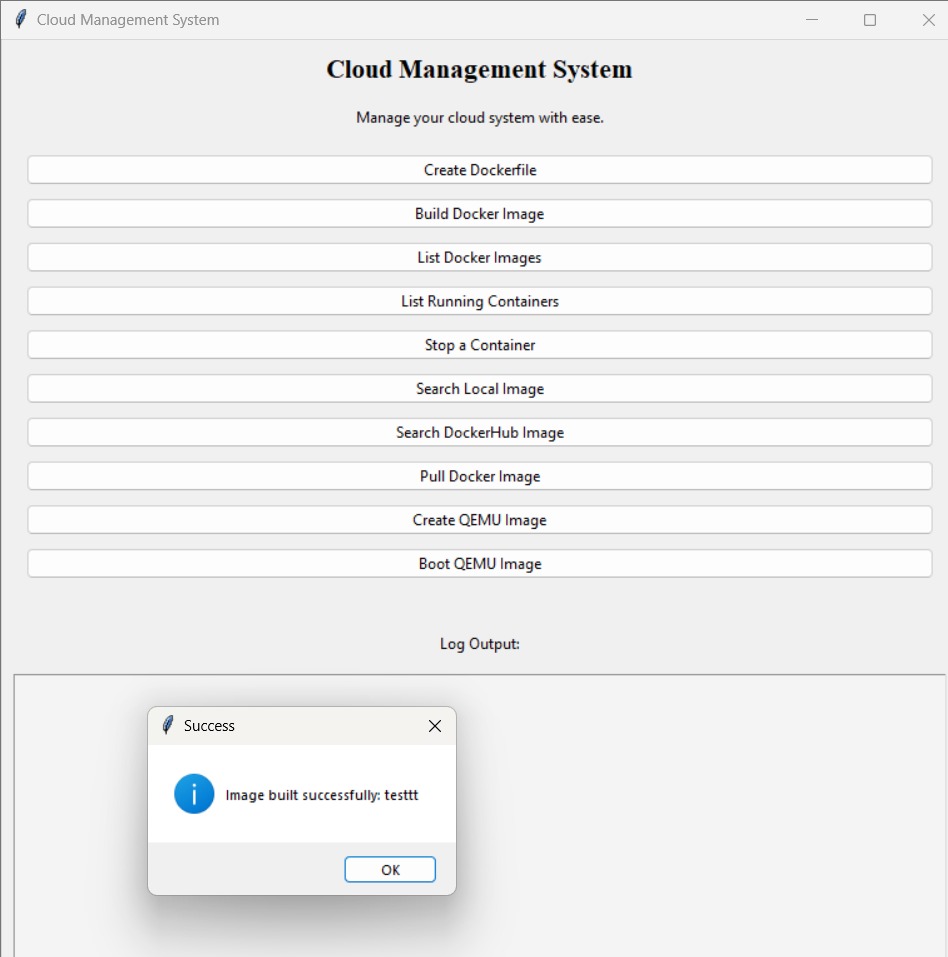
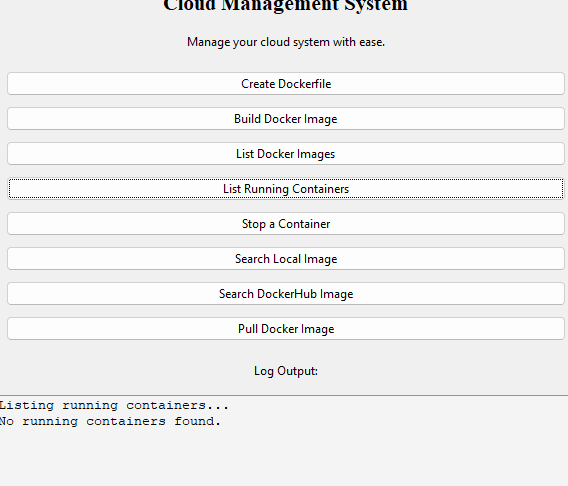
**Test Case 3: List Running Containers**

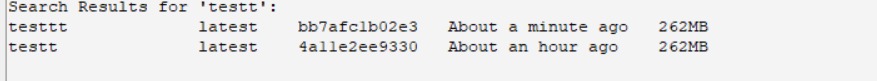
* **Input**: Click the "List Running Containers" button.
* **Expected Output**: Display of active containers with IDs and names.
* **Result**: Passed.

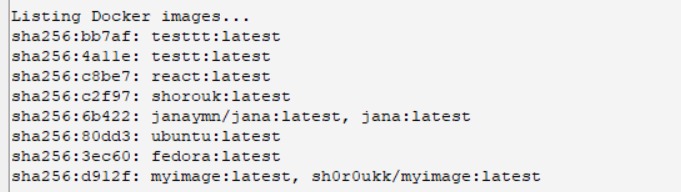
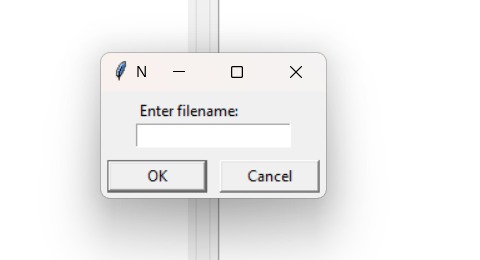
**Test Case 4: Create and Boot QEMU Image**

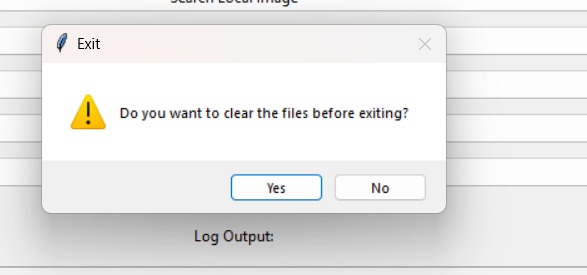
* **Input**: Specify RAM, cores, and ISO file for a virtual machine.
* **Expected Output**: Virtual machine boots successfully.
* **Result**: Passed (observed VM booting up with specified parameters).

**Screenshots/Evidence:**

1.  Building an image from the dockerfile we built.
2. Running containers listed in the GUI.
3. Search local image.



1. Running image listed in the GUI.
2. Adding file in custom mode dockerfile
3. On existing the program



**System Performance Evaluation**

#### Performance Metrics

1. **Execution Time**:

* Docker Image Building: The system averages a completion time of approximately 5 seconds for basic Dockerfiles. For more complex configurations, the time may increase slightly depending on the build steps.
* QEMU Boot: Default configurations achieve boot times within 10 seconds, showcasing efficient initialization of virtual environments.
* Docker Image Pulling: Typically requires 10-15 seconds to pull standard images. Larger or less commonly used images may take longer depending on network speed and image size.
* Container Operations: Listing running containers and stopping containers occur almost instantaneously, with sub-second response times under normal workloads.

**Error Handling**:

* + Proactively identifies user input errors (e.g., missing files or incorrect paths) and provides detailed, actionable feedback via logs.

#### Limitations

1. Limited support for advanced QEMU configurations.
2. Dependency on Docker and QEMU installations.
3. GUI responsiveness occasionally slowed during intensive operations (e.g., building large images).

#### Strengths

1. User-friendly interface for using the system.
2. Robust error handling and informative logs.
3. Simplified integration of Docker and QEMU functionalities.
4. Lightweight and responsive for standard tasks, maintaining stability even during extended sessions.
5. Seamless combination of Docker and QEMU functionalities, enabling users to manage containerized and virtualized environments from a single platform.