Cloud Management and Virtualization System - User Manual

Table of Contents

Introduction

The **Cloud Management System** is a web-based application built with Streamlit that provides an intuitive interface for managing both Virtual Machines (VMs) and Docker containers. This system simplifies complex cloud infrastructure management tasks through a user-friendly dashboard.

Key Features:

- **VM Management**: Create and manage QEMU-based virtual machines
- **Docker Management**: Build, pull, search, and manage Docker images and containers
- **Interactive UI**: Easy-to-use web interface with clear navigation
- **Configuration Support**: Both interactive and file-based configuration options

System Requirements

Prerequisites:

- **Python 3.8+**
- **Docker Engine** (for Docker management features)
- **QEMU/KVM** (for VM management features)

- **Streamlit** (will be installed automatically)
- **Linux-based OS** (recommended for full functionality)

Installation:

```
# Clone the repository
git clone https://github.com/MennaSherieff/Cloud-Management-and-Virtualization-Sys
tem
cd Cloud-Management-and-Virtualization-System

# Install dependencies
pip install -r requirements.txt

# Ensure Docker is running
sudo systemctl start docker

# Verify QEMU installation
qemu-system-x86_64 --version
```

Getting Started

Launching the Application:

```
streamlit run app.py
```

Once launched, open your web browser and navigate to:

```
http://localhost:8501
```

First-Time Setup:

1. Ensure Docker service is running
2. Verify QEMU is installed and accessible
3. Check user permissions for Docker operations
4. Review network connectivity for Docker Hub access

Navigation

The application uses a sidebar navigation system with two main categories:

Sidebar Layout:

```
☁ Navigation
├── 🖥️ VM (Expandable)
│   ├── Create VM (QEMU)
│   └── Create VM from Config File
│
└── 🐳 Docker (Expandable)
    ├── 📝 Create Dockerfile
    ├── 🛠️ Build Docker Image
    ├── 📦 List Local Docker Images
    ├── 🔍 Search Local Docker Image
    ├── 🌐 Search Image on DockerHub
    ├── ⬇️ Pull Docker Image
    ├── 📋 List Running Containers
    └── ⏹️ Stop Container
```

Navigation Tips:

- Click on "🖥️ VM" or "🐳 Docker" to expand/collapse their options
- Click any button to select that feature
- The selected feature will appear in the main content area
- The system remembers your last selection during the session

Virtual Machine Management

1. Create VM (QEMU) - Interactive Mode

Create a virtual machine with custom specifications.

Steps:

1. Select "🖥️ VM" → "Create VM (QEMU)"
2. Configure parameters:
    - **CPU cores**: Number of virtual CPUs (1-16)
    - **Memory (MB)**: RAM allocation (256-32768 MB)
    - **Disk Size**: Virtual disk size (e.g., "10G" for 10GB)

3. Click "Create VM"

What happens:

- Creates a QCOW2 disk image named `interactive_vm.qcow2`
- Validates all inputs
- Starts QEMU with specified configuration
- Outputs to terminal/log in non-graphical mode

Example:

```
CPU cores: 2
Memory (MB): 2048
Disk Size: 10G
```

Creates a VM with 2 CPUs, 2GB RAM, and 10GB disk.

2. Create VM from Config File

Create a VM using predefined configuration.

Prerequisites:

- Create `vm/config.txt` file with the following format:

```
cpu=2
memory=2048
disk_size=10G
```

Steps:

1. Ensure `vm/config.txt` exists with valid parameters
2. Select "🖥️ VM" → "Create VM from Config File"
3. Click "Create VM from Config"

What happens:

- Reads configuration from `vm/config.txt`
- Creates a QCOW2 disk image named `config_vm.qcow2`
- Validates configuration parameters
- Starts QEMU with loaded configuration

Docker Management

1. Create Dockerfile

Generate a Dockerfile in a specified directory.

Steps:

1. Select "🐳 Docker" → "📝 Create Dockerfile"
2. Enter:
    - **Directory path**: Absolute or relative path where Dockerfile should be created
    - **Dockerfile content**: Multi-line Dockerfile instructions
3. Click "Create Dockerfile"

Requirements:

- Directory must exist and be writable
- Content cannot be empty
- File will be created with 644 permissions

Example:

```
Directory path: /home/user/myapp
Content:
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y python3
COPY . /app
WORKDIR /app
CMD ["python3", "app.py"]
```

2. Build Docker Image

Build a Docker image from an existing Dockerfile.

Steps:

1. Select "🐳 Docker" → "🛠️ Build Docker Image"
2. Enter:
    - **Dockerfile Path**: Full path to Dockerfile
    - **Image Name:Tag**: Name and tag for the image (e.g., `myapp:v1.0`)
3. Click "Build Image"

Requirements:

- Dockerfile must exist
- Docker daemon must be running
- Sufficient permissions to run Docker commands

3. List Local Docker Images

Display all Docker images available locally.

Steps:

1. Select "🐳 Docker" → "📦 List Local Docker Images"
2. Click "List Images"

Output Format:

```
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        20.04     abc123def456   2 weeks ago    72.8MB
nginx         latest    def456ghi789   3 weeks ago    133MB
```

4. Search Local Docker Image

Search for specific images in the local registry.

Steps:

1. Select "🐳 Docker" → "🔍 Search Local Docker Image"
2. Enter search query (partial names/tags supported)
3. Click "Search Local Images"

Example:

```
Query: ubuntu
Output: Lists all local images containing "ubuntu" in name or tag
```

5. Search Image on DockerHub

Search for images in Docker Hub registry.

Steps:

1. Select "🐳 Docker" → "🌐 Search Image on DockerHub"
2. Enter search query

3. Click "Search DockerHub"

Requirements:

- Internet connectivity
- Docker Hub API accessibility

6. Pull Docker Image

Download an image from Docker Hub to local registry.

Steps:

1. Select "🐳 Docker" → "⬇️ Pull Docker Image"
2. Enter image name (e.g., `ubuntu:20.04`, `nginx:latest`)
3. Click "Pull Image"

7. List Running Containers

Display all currently running Docker containers.

Steps:

1. Select "🐳 Docker" → "📋 List Running Containers"
2. Click "List Containers"

Output Format:

```
CONTAINER ID   IMAGE        COMMAND       CREATED       STATUS       PORTS     NAMES
a1b2c3d4e5f6   nginx:latest   "nginx -g..." 2 hours ago   Up 2 hours   80/tcp    web-server
```

8. Stop Container

Stop a running Docker container.

Steps:

1. Select "🐳 Docker" → "⏹️ Stop Container"
2. Enter Container ID or name
3. Click "Stop Container"

Note: Use Container ID or name from the "List Running Containers" output.

Troubleshooting

Common Issues and Solutions:

1. "Docker is not installed" Error

```
# Install Docker
sudo apt-get update
sudo apt-get install docker.io
sudo systemctl start docker
sudo usermod -aG docker $USER  # Log out and back in
```

2. "Permission denied" for Docker Commands

```
# Add user to docker group
sudo usermod -aG docker $USER

# Apply changes (log out and back in or)
newgrp docker
```

3. QEMU Not Found

```
# Install QEMU
sudo apt-get install qemu-system-x86
```

4. VM Creation Fails

- Check QEMU installation: `qemu-system-x86_64 --version`
- Verify disk space availability
- Ensure KVM is enabled in BIOS
- Check `/dev/kvm` permissions

5. Docker Build Fails

- Verify Dockerfile syntax
- Check network connectivity for pulling base images
- Ensure sufficient disk space

- Verify Docker daemon is running: `sudo systemctl status docker`

6. Image Pull Fails

- Check internet connectivity
- Verify image name is correct
- Check Docker Hub status
- Ensure no firewall blocking Docker Hub

Error Messages Guide:

- **"ValueError: Dockerfile content cannot be empty"**: Provide content in the text area
- **"PermissionError: No write permission in directory"**: Check directory permissions
- **"FileNotFoundError: Dockerfile not found"**: Verify Dockerfile path is correct
- **"RuntimeError: Docker Hub search failed"**: Check network connectivity
- **"subprocess.CalledProcessError"**: Underlying command failed, check system logs

Appendix

File Structure:

```
cloud-management-system/
├── app.py                  # Main application
├── vm/
│   ├── __init__.py
│   ├── vm_manager.py       # VM management functions
│   ├── utils.py            # VM utilities
│   └── config.txt          # VM configuration template
├── docker/
│   ├── __init__.py
│   ├── dockerfile_manager.py
│   ├── image_manager.py
│   └── container_manager.py
└── requirements.txt        # Dependencies
```

Configuration File Format (`vm/config.txt`):

```
cpu=2
memory=2048
disk_size=10G
```

Dockerfile Requirements:

- Must be valid Dockerfile syntax
- Each instruction on a new line
- Supports all standard Dockerfile instructions
- File is created with UTF-8 encoding

QEMU Parameters:

- **CPU**: Virtual CPU cores (1-16)
- **Memory**: RAM in MB (256-32768)
- **Disk**: QCOW2 format, size specified with 'G' suffix

Support:

For additional assistance, please refer to:

- Docker Documentation: https://docs.docker.com
- QEMU Documentation: https://www.qemu.org/docs/
- Streamlit Documentation: https://docs.streamlit.io

Version: 1.0

Last Updated: 28 December, 2025

Compatibility: Python 3.8+, Docker 20.10+, QEMU 5.2+