**Cloud Management System Documentation**

**Link for Source code:** [**Source Code**](https://drive.google.com/file/d/1dMK6zXV3XWqZWidteGfnBrxHJHirHe2D/view?usp=sharing)

**Overview**

The Cloud Management System (CMS) is a Python-based GUI application designed to manage virtual machines and Docker containers. It provides a user-friendly interface to perform tasks such as creating and managing VMs using QEMU, building Docker images, managing containers, and interacting with DockerHub. The application is built using the Tkinter library for the GUI and Docker SDK for Python for Docker-related operations.

**Features**

1. **Create Virtual Machines**
   * Use existing QCOW2 disk images.
   * Create new VMs with custom configurations.
2. **Docker Management**
   * Create Dockerfiles.
   * Build 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**
   * Log significant actions and errors for debugging and traceability.

**Prerequisites**

1. **Python Libraries**:
   * tkinter
   * docker
   * os
   * subprocess
   * json
   * datetime
   * logging
   * time
2. **External Dependencies**:
   * QEMU: Install using package managers like apt (Linux) or Homebrew (MacOS).
3. **System Requirements**:
   * Python 3.x
   * Ensure Docker is installed and running.
   * QEMU binaries must be available in the system's PATH.

**Code Organization**

The code is divided into the following sections:

**1. Initialization**

* Initializes the Docker client using docker.from\_env().
* Creates the main Tkinter root window.
* Configures a logging module to capture application events.

**2. Functions**

**Virtual Machine Creation**

* create\_vm\_gui(): Handles the creation of VMs with options for using an existing image or creating a new configuration.
* **Key Functions**:
  + handle\_option\_selection(): Enables/disables fields based on the selected option.
  + browse\_image(): Opens a file dialog to select an existing image file.
  + browse\_config(): Opens a file dialog to select a configuration file and populates fields.
  + create\_vm(): Validates inputs and executes QEMU commands for creating VMs and logs actions and errors.

**Docker Management**

1. **Dockerfile Management**:
   * create\_dockerfile(): Allows the user to write and save a Dockerfile.
2. **Image Building**:
   * build\_docker\_image(): Builds a Docker image from a specified directory containing a Dockerfile.
3. **Listing Images and Containers**:
   * list\_docker\_images(): Lists all Docker images available locally.
   * list\_running\_containers(): Lists all currently running Docker containers.
4. **Container Management**:
   * stop\_container(): Stops a running Docker container selected by the user.
   * run\_new\_container()`: Allows the user to run a new container from a selected image with optional configurations like container name.
5. **Image Search and Download**:
   * search\_image(): Searches for a local Docker image by name.
   * search\_image\_dockerhub(): Searches for Docker images on DockerHub.
   * download\_image(): Downloads a Docker image from DockerHub.

**3. Main Application**

The main application window (root) provides a menu with buttons to access all features.

**Detailed Explanation of Functions**

**1. Virtual Machine Creation**

**create\_vm\_gui()**

Creates a GUI for VM creation with the following steps:

* **Option Selection**: Radio buttons to choose between using an existing image or creating a new VM.
* **Input Fields**:
  + For existing images: Path to the QCOW2 file, CPU count and memory size.
  + For new VMs: CPU count, memory size, disk size, and boot ISO file, and save location for the disk image.
* **Validation**:
  + Ensures all inputs are valid numbers and required files exist.
* **Execution**:
  + Uses os.system() to execute QEMU commands for VM creation.
* **Logging**:
  + Logs all significant actions, including VM creation and errors.

**Error Handling**

* Ensures the selected file exists.
* Validates numerical inputs for CPU, memory, and disk size.
* Displays errors using Tkinter's messagebox.

**2. Docker Management**

**Dockerfile Creation (create\_dockerfile)**

* Prompts the user to select a directory and write Dockerfile content.
* Saves the Dockerfile in the chosen directory.

**Build Docker Image (build\_docker\_image)**

* Allows the user to specify the image name and tag.
* Uses the Docker Python SDK to build the image from a Dockerfile.’
* Logs build progress and errors.

**Listing Docker Resources**

1. **Images**:
   * Retrieves a list of images using docker\_client.images.list().
   * Displays tags of available images.
   * Logs the listing action.
2. **Containers**:
   * Retrieves a list of running containers using docker\_client.containers.list().
   * Displays container IDs and names.
   * Logs the listing action.

**Stop Docker Container (stop\_container)**

* Lists running containers and allows the user to select one to stop.
* Stops the container using docker\_client.containers.get(container\_id).stop().
* Logs the action and any errors.

**Run New Docker Container (run\_new\_container)**

* Allows the user to run a new container from an existing image.
* Provides an option for container naming.
* Logs the action and captures errors.

**Search and Download Docker Images**

1. **Search Local Images**:
   * Searches for a partial match in the tags of local images.
   * Logs the search action.
2. **Search DockerHub**:
   * Executes docker search using subprocess.
3. **Download Image**:
   * Downloads an image from DockerHub using docker\_client.images.pull(image\_name).
   * Logs the download progress and errors.

**GUI Design**

**Main Menu**

Provides buttons for each feature:

* Create VM
* Create Dockerfile
* Build Docker Image
* List Docker Images
* List Running Containers
* Stop a Container
* Run New Container
* Search for Docker Image
* Search DockerHub
* Download Docker Image

**Submenus**

**Create VM Window**

* Fields and buttons dynamically enable or disable based on the selected option (existing image or new configuration).
* Includes a browse button to select the ISO path for new VM creation.
* Includes a browse button to choose the disk image path for an existing image VM.
* Includes a file dialog window for the user to choose the path to save the newly created disk image for new VM creation.

**Dockerfile Creation Window**

* A text area for writing the Dockerfile content.

**Docker Image Build Window**

* Fields for specifying the image name and tag.

**Usage**

**Running the Application**

1. Ensure Docker and QEMU are installed and configured.
2. Install required Python packages:

pip install docker

1. Run the script:

python cloud\_management\_system.py

1. Use the main menu to navigate and perform tasks.

**Example: Creating a VM**

1. Select **Create VM**.
2. Choose **Use Existing Image** or **Create New VM**.
3. Fill in the required fields.
4. Click **Create VM**.

**Error Handling and Debugging**

* **Common Errors**:
  + Missing dependencies: Ensure Docker and QEMU are installed.
  + Invalid inputs: Verify numerical inputs are positive integers.
* **Logs**: All significant actions are logged to a file for debugging and traceability

**Future Improvements**

1. Add support for cloud integration (e.g., AWS, Azure).
2. Expand logging capabilities to include user metrics and detailed action traces.
3. Enhance the GUI with additional customization options for VMs and Docker containers.

**Conclusion**

This application provides a unified interface for managing virtual machines and Docker resources, emphasizing usability and flexibility. By following the documentation, users can effectively utilize the CMS for various virtualization and containerization tasks.