## Overview

This documentation provides a comprehensive explanation of the Cloud Management System, emphasizing the design, functionality, and source code implementation. The system is designed to streamline virtualization and containerization tasks, offering features such as creating and managing virtual machines, generating Dockerfiles, building and managing Docker images, and handling running containers.

## Code Documentation

## Imports

import os

import tkinter as tk

from tkinter import filedialog, messagebox

Explanation of Imports

1. os: Used to interact with the operating system, including executing system commands for QEMU and Docker operations.  
2. tkinter: Employed for creating the GUI interface, enabling user-friendly interaction.  
3. filedialog and messagebox: File Dialog is used for file selection and saving, while Message Box is used for displaying success/error messages.

## GUI Design Colors

MAIN\_BG\_COLOR = "#F0F4FA"

MAIN\_BTN\_COLOR = "#4A90E2"

SUB\_BG\_COLOR = "#34405E"

SUB\_BTN\_COLOR = "#16A015"

These constants define the color scheme for the GUI:

The color of the background and button

def create\_vm(*cpu*, *memory*, *disk\_size*, *iso\_path*, *gui\_inputs*=False):

    try:

        if gui\_inputs:

            cpu, memory, disk\_size, iso\_path = cpu.get(), memory.get(), disk\_size.get(), iso\_path.get()

            if not all([cpu, memory, disk\_size, iso\_path]):

                return messagebox.showerror("Input Error", "All fields are required")

        os.system(f"qemu-img create -f qcow2 disk\_image.qcow2 {disk\_size}")

        os.system(f"qemu-system-x86\_64 -cpu {cpu} -m {memory} -hda disk\_image.qcow2 -cdrom {iso\_path} -boot d -no-fd-bootchk")

        if gui\_inputs:

            messagebox.showinfo("Success", "Virtual machine created successfully!")

    except Exception as e:

        if gui\_inputs:

            messagebox.showerror("Error", f"Failed to create virtual machine: {e}")

1. cpu: Specifies the CPU model to be emulated in the virtual machine.  
2. memory: Defines the amount of virtual RAM allocated to the VM (e.g., 512 MB, 1024 MB).  
3. disk\_size: Specifies the size of the virtual disk (e.g., 10G, 20G).  
4. iso\_path: Path to the ISO file used for booting the virtual machine.

## Workflow

1. Disk Creation:  
- Creates a virtual disk image named `disk\_image.qcow2` using the `qemu-img` command.  
- `-f qcow2`: Specifies the QCOW2 format for the disk image.  
- `<disk\_size>`: User-defined disk size.  
2. Virtual Machine Creation:  
- Executes the `qemu-system-x86\_64` command to start the VM.  
- `-cpu <cpu>`: Defines the CPU model.  
- `-m <memory>`: Sets the allocated RAM.  
- `-hda disk\_image.qcow2`: Attaches the created disk as the primary drive.  
- `-cdrom <iso\_path>`: Mounts the ISO file as the boot medium.  
- `-boot d`: Specifies boot from the CD-ROM.  
3. Error Handling:  
- Displays a success message if the VM is created successfully.  
- Captures and displays an error message in case of any issues.

## Additional Features

1. Create Dockerfile:  
   - Allows users to generate a Dockerfile based on their input.

def create\_dockerfile(*path*=None, *contents*=None, *gui\_inputs*=False, *path\_entry*=None, *contents\_text*=None):

    try:

        if gui\_inputs:

            path = path\_entry.get()

            contents = contents\_text.get("1.0", tk.END).strip()

            if not (path and contents):

                return messagebox.showerror("Input Error", "Both fields are required")

        with open(path, 'w') as file:

            file.write(contents)

        if gui\_inputs:

            messagebox.showinfo("Success", f"Dockerfile created at {path}")

    except Exception as e:

        if gui\_inputs:

            messagebox.showerror("Error", f"Failed to create Dockerfile: {e}")

2. Build Docker Image:  
- Builds a Docker image using a specified Dockerfile and assigns a user-defined name/tag to the image.

def build\_docker\_image(*dockerfile\_path*=None, *image\_name*=None, *gui\_inputs*=False, *dockerfile\_entry*=None, *image\_entry*=None):

    try:

        if gui\_inputs:

            dockerfile\_path, image\_name = dockerfile\_entry.get(), image\_entry.get()

            if not (dockerfile\_path and image\_name):

                return messagebox.showerror("Input Error", "Both fields are required")

        result = os.popen(f"docker build -t {image\_name} -f {dockerfile\_path} .").read()

        if gui\_inputs:

            messagebox.showinfo("Success", f"Docker image {image\_name} built successfully!\n\n{result}")

    except Exception as e:

        if gui\_inputs:

            messagebox.showerror("Error", f"Failed to build Docker image: {e}")

3. List Docker Images and Containers:  
- Displays all available Docker images and running containers.

def list\_docker\_images():

    try:

        result = os.popen("docker images").read()

        return result

    except Exception as e:

        return f"Error: {e}\nThis error may indicate that the Docker daemon is not running."

def list\_running\_containers():

    try:

        result = os.popen("docker ps").read()

        return result

    except Exception as e:

        return f"Error: {e}\nThis error may indicate that the Docker daemon is not running."

4. Stop Docker Containers:  
- Enables users to stop a running container by providing its ID or name.

def stop\_container(*container\_id*=None, *gui\_inputs*=False, *container\_id\_entry*=None):

    try:

        if gui\_inputs:

            container\_id = container\_id\_entry.get()

            if not container\_id:

                return messagebox.showerror("Input Error", "Container ID/Name is required")

        result = os.popen(f"docker stop {container\_id}").read()

        messagebox.showinfo("Success", f"Container {container\_id} stopped successfully!\n\n{result}" if result else f"Container {container\_id} stopped successfully!")

    except Exception as e:

        if gui\_inputs:

            messagebox.showerror("Error", f"Failed to stop container: {e}")

5. Search Docker Images:  
- Performs a local search or queries DockerHub for images.

def search\_image(*image\_name\_entry*, *result\_text*):

    image\_name = image\_name\_entry.get()

    if not image\_name:

        messagebox.showerror("Input Error", "Image name/tag is required")

        return

    try:

        result = os.popen(f"docker images --filter=reference={image\_name}").read()

        result\_text.delete(1.0, tk.END)

        result\_text.insert(tk.END, result)

    except Exception as e:

        messagebox.showerror("Error", f"Error: {e}\nThis error may indicate that the Docker daemon is not running.")

def search\_image\_dockerhub(*image\_name\_entry*, *result\_text*):

    image\_name = image\_name\_entry.get()

    if not image\_name:

        messagebox.showerror("Input Error", "Image name/tag is required")

        return

    try:

        result = os.popen(f"docker search {image\_name}").read()

        result\_text.delete(1.0, tk.END)

        result\_text.insert(tk.END, result)

    except Exception as e:

        messagebox.showerror("Error", f"Error: {e}\nThis error may indicate that the Docker daemon is not running.")

6. Pull Docker Images:  
- Allows users to download an image from DockerHub by providing its name/tag.

def pull\_image(*image\_name\_entry*):

    image\_name = image\_name\_entry.get()

    if not image\_name:

        messagebox.showerror("Input Error", "Image name/tag is required")

        return

    try:

        result = os.popen(f"docker pull {image\_name}").read()

        messagebox.showinfo("Result", result)

    except Exception as e:

        messagebox.showerror("Error", f"Error: {e}\nThis error may indicate that the Docker daemon is not running.")

## Error Handling

1. Input Validation:  
- Ensures valid inputs for CPU models, memory size, disk size, and paths.  
2. Error Messages:  
- Provides detailed error messages for missing files, incorrect inputs, or system failures.  
3. Success Notifications:  
- Displays success messages after completing operations like VM creation or Docker image building.

## Conclusion

The Cloud Management System is a comprehensive tool designed to manage virtualization and containerization efficiently. The GUI ensures ease of use, while the robust error-handling mechanisms enhance reliability. This documentation highlights the design choices and the functionality implemented in the system.