Convert a simple Bash script into a Python script for enhanced functionality

Table of Contents

Description

This guide covers how to convert a simple **Bash script** used for automating Docker container management into a **Python script**. Python provides more flexibility, better error handling, and can integrate with other tools or libraries, making it ideal for more complex automation tasks.

Problem Statement

While Bash scripts are simple and effective for automating tasks, they lack advanced error handling, data manipulation, and scalability that can be easily achieved with Python. This document converts a simple Bash script that manages Docker containers into a Python script with enhanced functionality.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-05) is required before proceeding with Lab Guide-06.

Software Required

- **Python 3.x**: Installed on your machine.
- **Docker**: Docker Desktop installed and running.
- **pip**: Python package manager to install necessary libraries.
- **TodoAPP_MYSQl**: To download the source folder **click here**

Hardware Requirement

- Minimum of 4 GB RAM

- At least 2 cores in the processor

Implementation Steps

Step-1: Create a Custom Docker Network and MySQL Container

1. **Create the Docker Network**:

   First, we'll create a custom network named **todoapp_network**.

   ```
   docker network create todoapp_network
   ```

   CreateNetwork

2. **Run the MySQL Container**: Use the following command to create a MySQL container connected to the custom network:

   ```
   docker run -d -p3306:3306 --network=todoapp_network -e
   MYSQL_ROOT_PASSWORD=P@ssw0rd -e MYSQL_DATABASE=tododb --name=mysqldb mysql
   ```

   mysqlnet

Step-2: Original Bash Script (Container Management)

Consider a simple **Bash script** to build and run a Docker container for the **TodoApp**:

`run_todoapp.sh` **- Original Bash Script**

```
#!/bin/bash

# Set variables
IMAGE_NAME="todoapp_image"
CONTAINER_NAME="todoapp_container"

# Build the Docker image
docker build -t $IMAGE_NAME .

# Run the Docker container
# docker run -d --name $CONTAINER_NAME -p 8080:8080 $IMAGE_NAME
docker run -d -p8081:8081 --name todoapp_container --network=todoapp_network -e
MYSQL_HOST=mysqldb todoapp_image
```

Step-3: Convert the Bash Script to Python

Now, let's convert the above Bash script into a **Python script**. This script will perform the same tasks: build and run a Docker container.

2.1 `run_todoapp.py` - Python Script

```python
import subprocess
import sys

# Set variables
image_name = "todoapp_image"
container_name = "todoapp_container"
port = "8081:8081"

def run_command(command):
    """Run shell commands and handle errors."""
    try:
        result = subprocess.run(command, shell=True, check=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        print(result.stdout.decode())
    except subprocess.CalledProcessError as e:
        print(f"Error running command: {command}")
        print(e.stderr.decode())
        sys.exit(1)

# Step 1: Build Docker image
print("Building Docker image...")
run_command(f"docker build -t {image_name} .")

# Step 2: Run Docker container
print("Running Docker container...")
run_command(f"docker run -d --name {container_name} -p {port} --
network=todoapp_network -e MYSQL_HOST=mysqldb {image_name}")

print(f"TodoApp is running on http://localhost:{port.split(':')[0]}")
```

Key differences in the Python script:

- **subprocess module**: Executes shell commands in Python with better error handling.
- **run_command function**: Centralizes command execution and error handling.

Usage:

1. Run the Python script:

```
python run_todoapp.py
```

pythonscript

Step-4: Adding Enhanced Functionality in Python

Now that we have the basic conversion, let's enhance the Python script by adding the following functionalities:

- **Check if the container is already running** before trying to start it.
- **Stop and remove the container** if it’s running to ensure a fresh start.
- **Log output** to a file for debugging purposes.

3.1 Enhanced `run_todoapp.py` with Additional Functionality

```
import subprocess
import sys
import logging

# Set up logging
logging.basicConfig(filename='todoapp.log', level=logging.INFO, format='%
(asctime)s - %(message)s')

# Set variables
import subprocess
import sys
import logging

# Set up logging
logging.basicConfig(filename='todoapp.log', level=logging.INFO, format='%
(asctime)s - %(message)s')

image_name = "todoapp_image"
container_name = "todoapp_container"
port = "8081:8081"

def run_command(command):
    """Run shell commands and handle errors."""
    try:
        result = subprocess.run(command, shell=True, check=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output = result.stdout.decode()
        logging.info(f"Command succeeded: {command}")
        logging.info(f"Output: {output}")
        print(output)
    except subprocess.CalledProcessError as e:
        logging.error(f"Error running command: {command}")
        logging.error(e.stderr.decode())
        print(f"Error running command: {command}")
        print(e.stderr.decode())
        sys.exit(1)

def check_container_running():
    """Check if the container is already running."""
    command = f"docker ps --filter 'name={container_name}' --format
'{{{{.Names}}}}'"
```

```
    result = subprocess.run(command, shell=True, stdout=subprocess.PIPE)
    return container_name in result.stdout.decode()

def stop_and_remove_container():
    """Stop and remove the container if it's running."""
    if check_container_running():
        print(f"Stopping and removing container: {container_name}...")
        run_command(f"docker stop {container_name}")
        run_command(f"docker rm {container_name}")

# Step 1: Stop and remove any running container
stop_and_remove_container()

# Step 2: Build Docker image
print("Building Docker image...")
run_command(f"docker build -t {image_name} .")

# Step 3: Run Docker container
print("Running Docker container...")
run_command(f"docker run -d --name {container_name} -p {port} --network=todoapp_network -e MYSQL_HOST=mysqldb {image_name}")

print(f"TodoApp is running on http://localhost:{port.split(':')[0]}")
logging.info("TodoApp is running successfully")
```

Enhancements:

- **Check if the container is already running**: Prevents errors by stopping and removing an existing container.
- **Logging**: All output and errors are logged into `todoapp.log` for future reference.
- **Error handling**: Improved error messages and process termination if something fails.

Usage:

1. Make sure Python 3.x is installed.
2. Run the enhanced Python script:

```
python run_todoapp.py
```

pythonscript2 3. Check `todoapp.log` for logs of the container management tasks.

References

- Python `subprocess` Module Documentation: https://docs.python.org/3/library/subprocess.html
- Automating Docker Workflows: https://docs.docker.com/engine/admin/