PCET's
Pimpri
Chinchwad
University, Pune

Learn | Grow | Achieve

Checked By	Dr. Avinash Golande
Sign	
Date	
Grade	

EXPERIMENT NO . 09

- **Title:-** Use **Python and docker-py** or **subprocess** module to deploy multiple container instances and demonstrate scalability**.**

- **Objective :-**

- To understand the concept of container scalability in cloud environments.
- To deploy multiple container instances using Python automation.
- To demonstrate dynamic scaling and resource utilization.

- **Resources used :-** PC / Laptop with Docker installed, Python 3.x, docker SDK for Python (pip install docker), subprocess module, time module.

- **Theory :-**

Modern cloud and DevOps systems often require running multiple containers of the same service (e.g., Nginx or Flask) to handle heavy workloads or distribute traffic.
This concept is known as **scalability** — increasing or decreasing the number of container instances according to demand.
Python can automate container management using two common methods:

1. **docker-py** (official Docker SDK for Python) — interacts directly with the Docker Engine API.
2. **subprocess module** — executes Docker CLI commands from Python.

By deploying multiple instances of a Docker image on different ports, we simulate **horizontal scaling**, where identical services run concurrently to share load.

This forms the foundation of scalable microservice architecture used in cloud platforms like AWS and Kubernetes.

- **Code**

Method 1: Using docker-py

```
import docker
import time

# Initialize Docker client
client = docker.from_env()
image_name = "nginx:latest"
num_containers = 3

# Pull Docker image
print(f"Pulling Docker image '{image_name}'...")
client.images.pull(image_name)

containers = []
print(f"\nLaunching {num_containers} container instances...\n")

for i in range(num_containers):
    container = client.containers.run(
        image_name,
        name=f"nginx_instance_{i+1}",
        detach=True,
        ports={'80/tcp': 8000 + i},
        tty=True
    )
    containers.append(container)
    print(f"Started container: nginx_instance_{i+1} on port {8000 + i}")
    time.sleep(1)

# Scale up
print("\nScaling up: launching 2 more containers...\n")
for i in range(num_containers, num_containers + 2):
    container = client.containers.run(
        image_name,
        name=f"nginx_instance_{i+1}",
        detach=True,
        ports={'80/tcp': 8000 + i},
        tty=True
    )
    containers.append(container)
    print(f"Scaled up container: nginx_instance_{i+1} on port {8000 + i}")

# Cleanup
time.sleep(5)
print("\nCleaning up containers...")
for container in containers:
    container.stop()
    container.remove()
    print(f"Removed {container.name}")

print("\nAll containers stopped and removed successfully!")
```

Method 2: Using subprocess

```
import subprocess
import time

image = "nginx:latest"
num_containers = 3

# Pull Docker image
subprocess.run(["docker", "pull", image], check=True)

# Launch multiple containers
for i in range(num_containers):
    name = f"nginx_instance_{i+1}"
    port = 8000 + i
    subprocess.run(["docker", "run", "-d", "--name", name, "-p", f"{port}:80", image], check=True)
    print(f"Started container: {name} on port {port}")
    time.sleep(1)

# Scale up
print("\nScaling up...")
for i in range(num_containers, num_containers + 2):
    name = f"nginx_instance_{i+1}"
    port = 8000 + i
    subprocess.run(["docker", "run", "-d", "--name", name, "-p", f"{port}:80", image], check=True)
    print(f"Scaled container: {name} on port {port}")

# Cleanup
time.sleep(5)
print("\nCleaning up containers...")
for i in range(num_containers + 2):
    name = f"nginx_instance_{i+1}"
    subprocess.run(["docker", "rm", "-f", name], check=True)
    print(f"Removed container: {name}"print("\nAll containers removed.")
```

- **Output**

```
Pulling Docker image 'nginx:latest'...
Launching 3 container instances...
Started container: nginx_instance_1 on port 8000
Started container: nginx_instance_2 on port 8001
Started container: nginx_instance_3 on port 8002

Scaling up...
Scaled up container: nginx_instance_4 on port 8003
Scaled up container: nginx_instance_5 on port 8004

Cleaning up containers...
Removed nginx_instance_1
Removed nginx_instance_2
Removed nginx_instance_3
Removed nginx_instance_4
Removed nginx_instance_5

All containers stopped and removed successfully!
```

• Conclusion:

Successfully deployed and scaled multiple Docker containers using Python. Demonstrated scalability by dynamically launching and removing container instances. Learned two automation methods — docker SDK and subprocess module. Understood the concept of horizontal scaling in containerized environments.