Hands-On Lab (HOL)

Lab Name

HOL-04: Docker CPU and Memory Resource Management on Ubuntu 22.04

Author : **Sandeep Kumar Sharma**

Learning Objectives

By the end of this lab, participants will be able to:

Understand why CPU and memory limits are required for Docker containers. Assign memory (RAM) limits to containers. Assign CPU limits to containers using different Docker options. Verify and monitor container resource usage. Understand what happens when a container exceeds its allocated memory or CPU. Relate Docker resource limits to real-world production and Kubernetes environments.

Lab Outcome

After completing this lab, you will be able to control and optimize resource usage of Docker containers. You will clearly understand how Docker uses Linux cgroups internally to enforce CPU and memory limits and how these limits help in maintaining system stability in production environments.

Pre-requisites

Docker must be installed and running on Ubuntu 22.04. Basic knowledge of Docker containers (HOL-01, HOL-02, HOL-03). Root or sudo access.

Lab Explanation and Hands-On Steps

Section 1: Why Resource Management is Important

By default, a Docker container can consume all available CPU and memory of the host system. In real production environments, this can cause performance degradation or system crashes. To avoid this, Docker allows us to define CPU and memory limits for each container.

Docker internally uses **Linux cgroups (control groups)** to enforce these limits.

Section 2: Assigning Memory (RAM) to a Container

Create a container with a fixed memory limit.

```
docker run -it --name container1 --memory=256m ubuntu
```

Explanation: - `--memory=256m` limits the container to 256 MB RAM - If the container tries to use more than this limit, it will be killed by the kernel (OOM Killer)

Exit the container:

```
exit
```

Verify the container:

```
docker ps -a
```

Section 3: Memory with Swap (Conceptual Understanding)

```
docker run -it --name container2 --memory=256m --memory-swap=512m ubuntu
```

Explanation: - `--memory` sets physical RAM limit - `--memory-swap` sets RAM + swap limit - If swap is disabled on the host, the container will still be restricted to RAM only

Exit the container:

```
exit
```

Section 4: Assigning CPU Limits to a Container

Option 1: Limiting CPU using --cpus (Recommended)

```
docker run -it --name container3 --cpus="1" ubuntu
```

Explanation: - The container can use only 1 CPU core - Docker throttles CPU usage beyond this limit - This is the most commonly used CPU control method

Exit the container:

```
exit
```

Option 2: CPU Core Pinning using --cpuset-cpus

```
docker run -it --name container4 --cpuset-cpus="0" ubuntu
```

Explanation: - The container can run only on CPU core 0 - Useful for performance-sensitive workloads

Exit the container:

```
exit
```

Section 5: Assigning CPU and Memory Together

```
docker run -it
--name container5
--cpus="1"
--memory="512m"
ubuntu
```

Explanation: - CPU limited to 1 core - Memory limited to 512 MB - This is a typical real-world production configuration

Exit the container:

```
exit
```

Section 6: Verifying Resource Configuration

Inspect container configuration:

```
docker inspect container5
```

Look for the following fields: - `Memory` - `NanoCpus` - `CpusetCpus`

Section 7: Monitoring Live Resource Usage

```
docker stats
```

This command displays real-time CPU and memory usage of all running containers. This is one of the most important monitoring commands used by DevOps engineers.

Press `CTRL + C` to exit.

Section 8: Understanding OOM (Out Of Memory) Behavior

If a container exceeds its memory limit: - The Linux kernel triggers the OOM Killer - The container is terminated - Container status will show:

```
Exited (137)
```

This is a very common production and interview scenario.

Section 9: Cleanup

Remove all containers created in this lab:

```
docker rm -f container1 container2 container3 container4 container5
```

Verify cleanup:

```
docker ps -a
```

Summary

In this lab, you learned how to assign CPU and memory resources to Docker containers and how Docker enforces these limits using Linux cgroups. You also learned how to monitor container resource usage and understand OOM behavior. This lab forms a strong foundation for Kubernetes resource management concepts such as requests and limits.