SEPM EXPERIMENT 9

Aim: To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers

THEORY:

Docker is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container. Containers are light in weight which makes them portable and they are isolated from the underlying infrastructure and from each other container. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system.

Importance of Docker in System Design

Docker plays a crucial role in modern system design for several reasons:

- **Portability**: Docker containers can run consistently across different environments (development, testing, staging, production), ensuring that the application works the same way everywhere. This eliminates the "it works on my machine" problem.

- **Scalability**: Docker simplifies scaling applications. You can quickly spin up multiple containers to handle increased load and manage them using container orchestration tools like Kubernetes. This makes it easier to design systems that can scale horizontally.

- **Continuous Integration and Continuous Deployment (CI/CD)**: Docker integrates seamlessly with CI/CD pipelines, enabling automated testing, building, and deployment of applications. This accelerates development cycles and improves software quality.

- **Consistency**: Docker ensures that applications run consistently across different environments by encapsulating all dependencies and configurations within the container. This reduces environment-specific bugs and streamlines the development process.

- **Microservices Architecture**: Docker is well-suited for microservices architecture, where an application is composed of small, independent services that can be developed, deployed, and scaled independently. Docker containers provide the ideal encapsulation for each microservice.

Key Components of Docker

Docker consists of several key components that work together to facilitate containerized application development and deployment. Here are the main components:

- **Docker Engine**: The core part of Docker, it is responsible for creating and managing containers. It consists of:
  - **Docker Daemon (dockerd)**: Runs on the host machine and is responsible for building, running, and managing Docker containers.
  - **Docker CLI**: A command-line interface that allows users to interact with the Docker Daemon via commands.

  - **REST API**: Provides an interface that programs can use to communicate with the Docker Daemon.

- **Docker Images**: Read-only templates used to create containers. An image includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files.

- **Docker Containers**: Instances of Docker images that run applications. Containers are isolated from each other and the host system, ensuring that they have their own environment.

- **Dockerfile**: A text file that contains a set of instructions for building a Docker image. It specifies the base image, application code, dependencies, environment variables, and commands to run.

- **Docker Compose**: A tool for defining and running multi-container Docker applications. It uses a YAML file (docker-compose.yml) to configure the application's services, networks, and volumes.

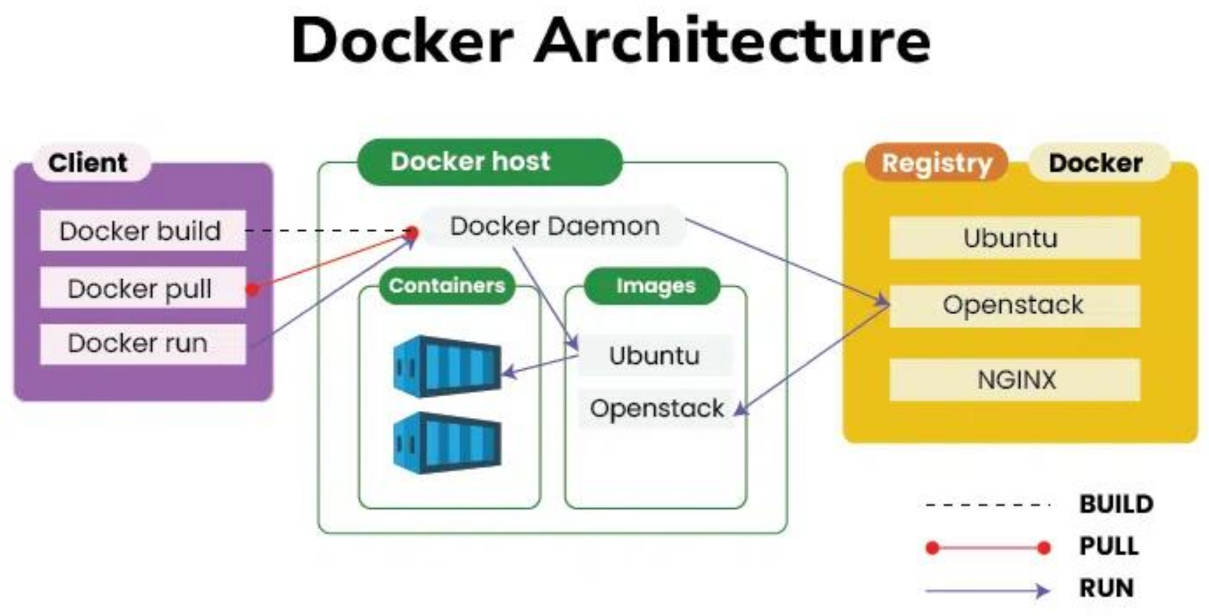
- **Docker Hub**: A cloud-based registry service for storing and distributing Docker images. Users can upload their own images and access publicly available images.

- **Docker Swarm**: A native clustering and orchestration tool for Docker. It allows you to manage a cluster of Docker nodes as a single virtual system, facilitating container orchestration, scaling, and load balancing.

- **Docker Network**: Provides networking capabilities for Docker containers. Docker creates default networks that containers can connect to, and users can define their own networks to control how containers communicate with each other and with external services.

Docker Architecture

Docker architecture is composed of several layers and components that work together to create, manage, and orchestrate containers. Below is an in-depth explanation of Docker's architecture:



Docker Architecture

1. Docker Engine

Docker Engine is the core component of Docker, responsible for building and running containers. It has three main parts:

- **Docker Daemon (dockerd)**: The background service running on the host machine. It listens for Docker API requests and manages Docker objects like images, containers, networks, and volumes.
- **Docker CLI**: The command-line interface used by users to interact with Docker. The CLI communicates with the Docker Daemon using the Docker API.
- **REST API**: A set of HTTP endpoints that provide an interface for interacting with the Docker Daemon programmatically. This API is used by both the Docker CLI and other tools to control Docker.

2. Docker Objects

Docker uses several key objects to build and run containerized applications:

- **Images**: Read-only templates used to create containers. Images are built from Dockerfiles and can be stored in Docker registries like Docker Hub.
- **Containers**: Instances of Docker images. Containers run applications and are isolated from the host system and other containers.
- **Volumes**: Persistent storage for Docker containers. Volumes can be shared between containers and are used to store data outside of the container's filesystem.
- **Networks**: Allow containers to communicate with each other and with external systems. Docker supports different types of networks, including bridge networks, overlay networks, and host networks.

3. Dockerfile

A **Dockerfile** is a text file containing a series of instructions on how to build a Docker image. Each instruction in a Dockerfile creates a new layer in the image. Typical instructions include specifying a base image, copying application files, installing dependencies, and defining environment variables.

4. Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file (docker-compose.yml) to configure the application's services, networks, and volumes. Docker Compose simplifies the management of complex applications by allowing you to define and run multiple containers as a single service.

5. Docker Swarm

Docker Swarm is Docker's native clustering and orchestration tool. It transforms a group of Docker engines into a single virtual Docker engine. Swarm enables the deployment, scaling, and management of multi-container applications across a cluster of machines. Key features of Docker Swarm include:

- **Node Management**: Nodes are individual Docker engines participating in the Swarm cluster. There are two types of nodes: managers and workers. Managers handle the cluster management tasks, while workers execute the containers.
- **Services**: A service is the definition of how a container should behave in production. It includes information on which image to use, network settings, and scaling policies.
- **Tasks**: A task is a single container running in the Swarm. Tasks are distributed across nodes by the Swarm manager.

6. Docker Networking

Docker provides several networking options to control how containers communicate:

- **Bridge Network**: The default network driver, allowing containers on the same host to communicate with each other.
- **Host Network**: Containers use the host's network stack directly, offering improved performance at the expense of isolation.
- **Overlay Network**: Enables containers running on different Docker hosts to communicate, typically used in Swarm clusters.
- **None Network**: Completely disables networking for a container, useful for isolated workloads.

7. Docker Storage

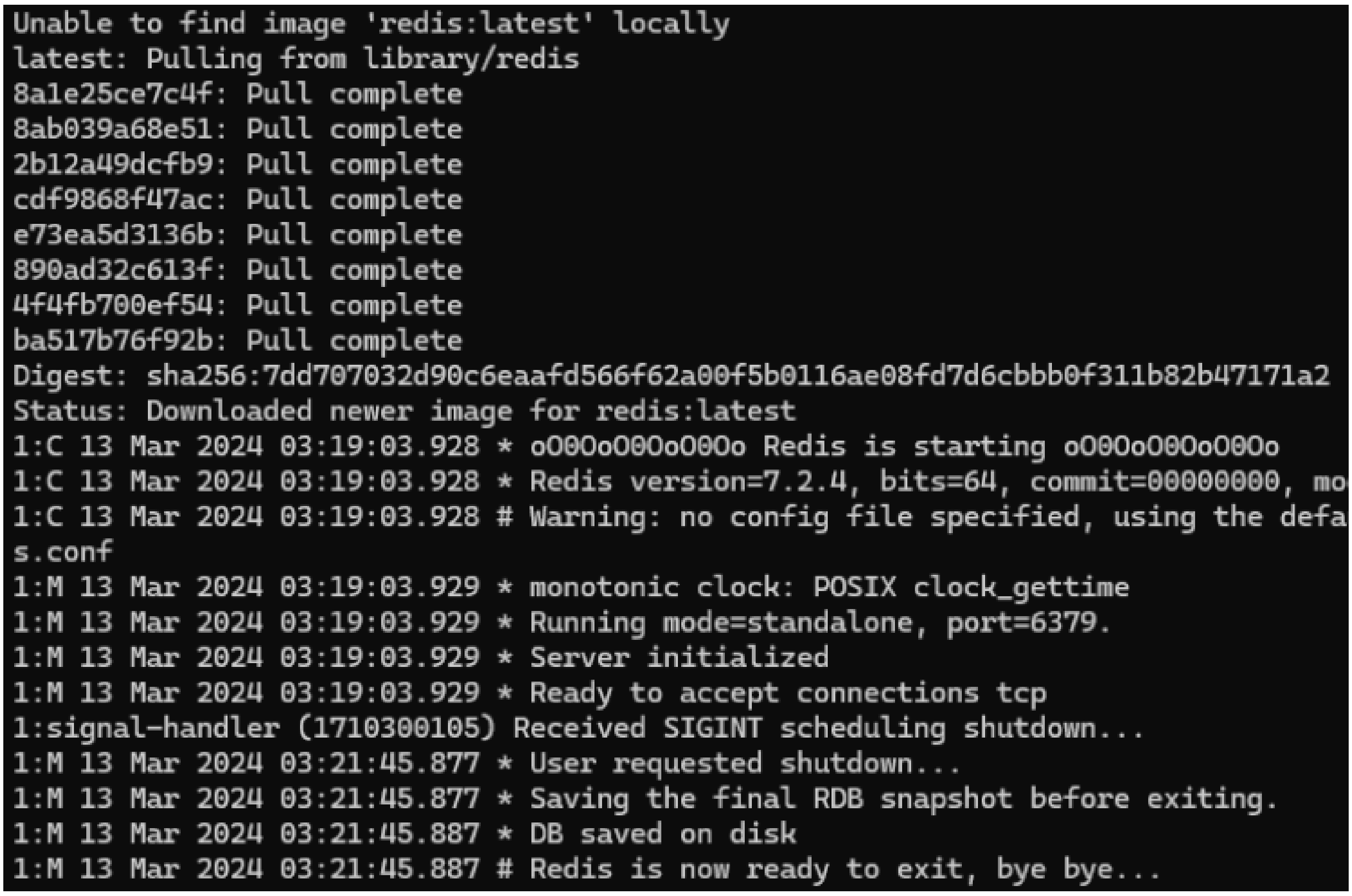
Docker provides different storage options to manage data:

- **Volumes**: Preferred method for persisting data, stored outside the container's filesystem.
- **Bind Mounts**: Mounts a directory or file from the host into the container. Useful for sharing data between the host and the container.
- **Tmpfs Mounts**: Creates temporary storage that is only stored in the host's memory and never written to the filesystem. Useful for sensitive data that does not need to persist.

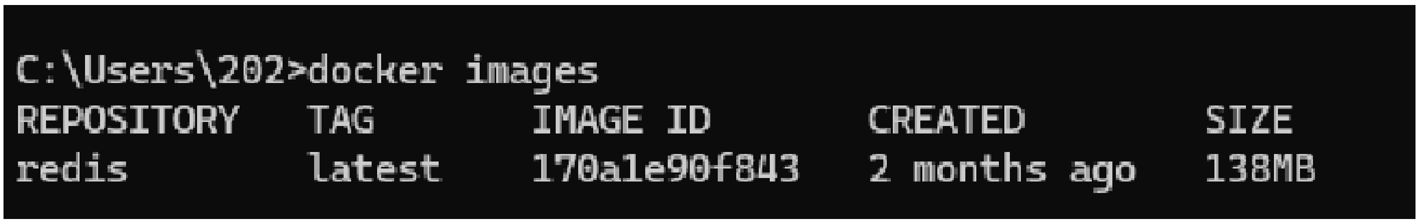
8. Docker Registry

A **Docker Registry** is a storage and distribution system for Docker images. Docker Hub is the default public registry, but you can also set up private registries. Registries store image repositories, which consist of multiple image versions (tags).

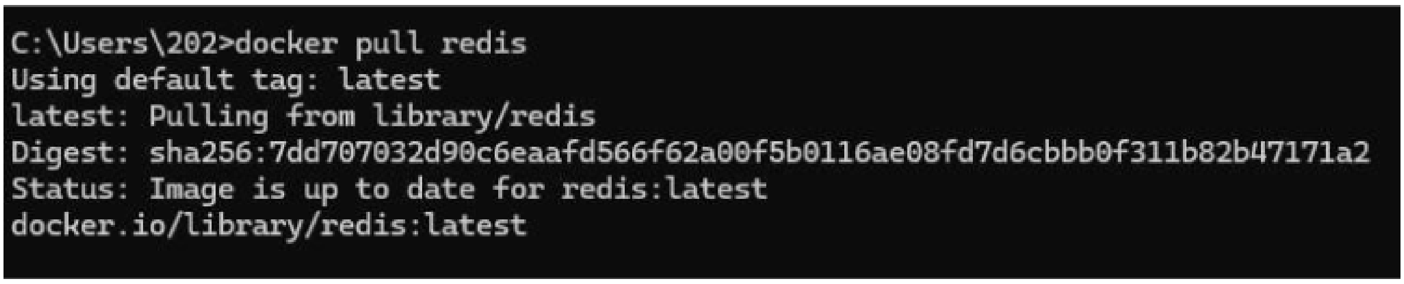
OUTPUT:



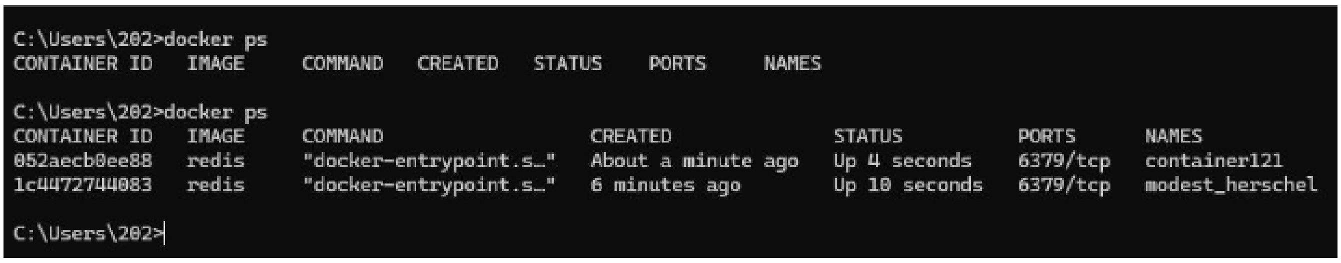
```
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
8a1e25ce7c4f: Pull complete
8ab039a68e51: Pull complete
2b12a49dcfb9: Pull complete
cdf9868f47ac: Pull complete
e73ea5d3136b: Pull complete
890ad32c613f: Pull complete
4f4fb700ef54: Pull complete
ba517b76f92b: Pull complete
Digest: sha256:7dd707032d90c6eaafd566f62a00f5b0116ae08fd7d6cbbb0f311b82b47171a2
Status: Downloaded newer image for redis:latest
1:C 13 Mar 2024 03:19:03.928 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 13 Mar 2024 03:19:03.928 * Redis version=7.2.4, bits=64, commit=00000000, mo
1:C 13 Mar 2024 03:19:03.928 # Warning: no config file specified, using the defa
s.conf
1:M 13 Mar 2024 03:19:03.929 * monotonic clock: POSIX clock_gettime
1:M 13 Mar 2024 03:19:03.929 * Running mode=standalone, port=6379.
1:M 13 Mar 2024 03:19:03.929 * Server initialized
1:M 13 Mar 2024 03:19:03.929 * Ready to accept connections tcp
1:signal-handler (1710300105) Received SIGINT scheduling shutdown...
1:M 13 Mar 2024 03:21:45.877 * User requested shutdown...
1:M 13 Mar 2024 03:21:45.877 * Saving the final RDB snapshot before exiting.
1:M 13 Mar 2024 03:21:45.887 * DB saved on disk
1:M 13 Mar 2024 03:21:45.887 # Redis is now ready to exit, bye bye...
```



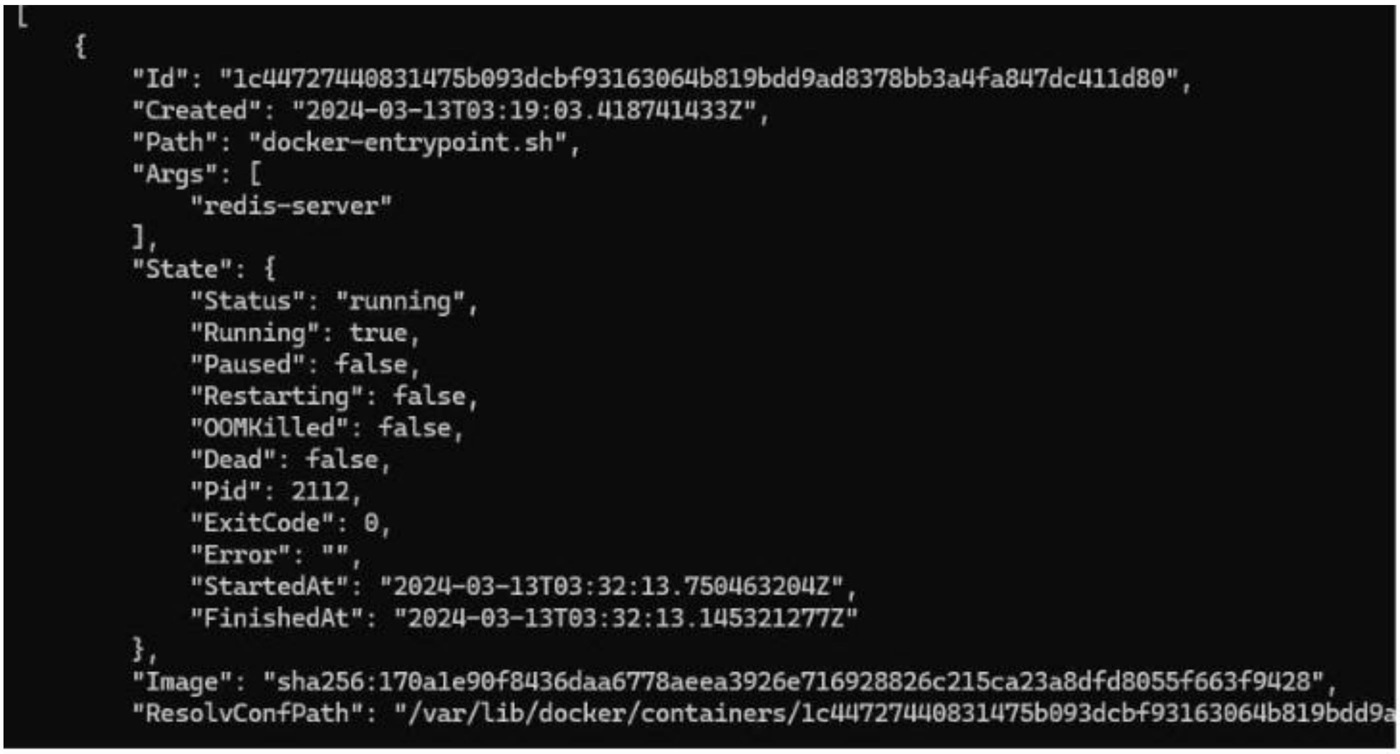
```
C:\Users\202>docker images
REPOSITORY   TAG      IMAGE ID       CREATED         SIZE
redis        latest   170a1e90f843   2 months ago    138MB
```



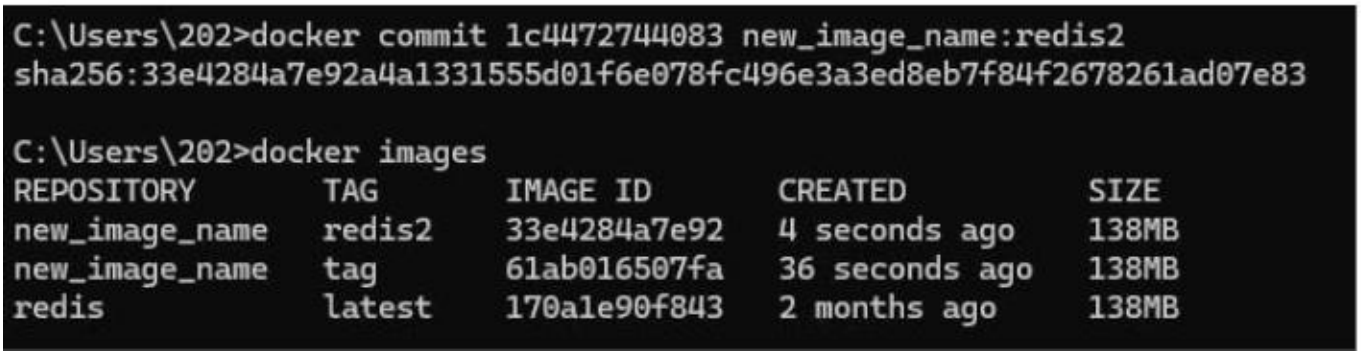
```
C:\Users\202>docker pull redis
Using default tag: latest
latest: Pulling from library/redis
Digest: sha256:7dd707032d90c6eaafd566f62a00f5b0116ae08fd7d6cbbb0f311b82b47171a2
Status: Image is up to date for redis:latest
docker.io/library/redis:latest
```



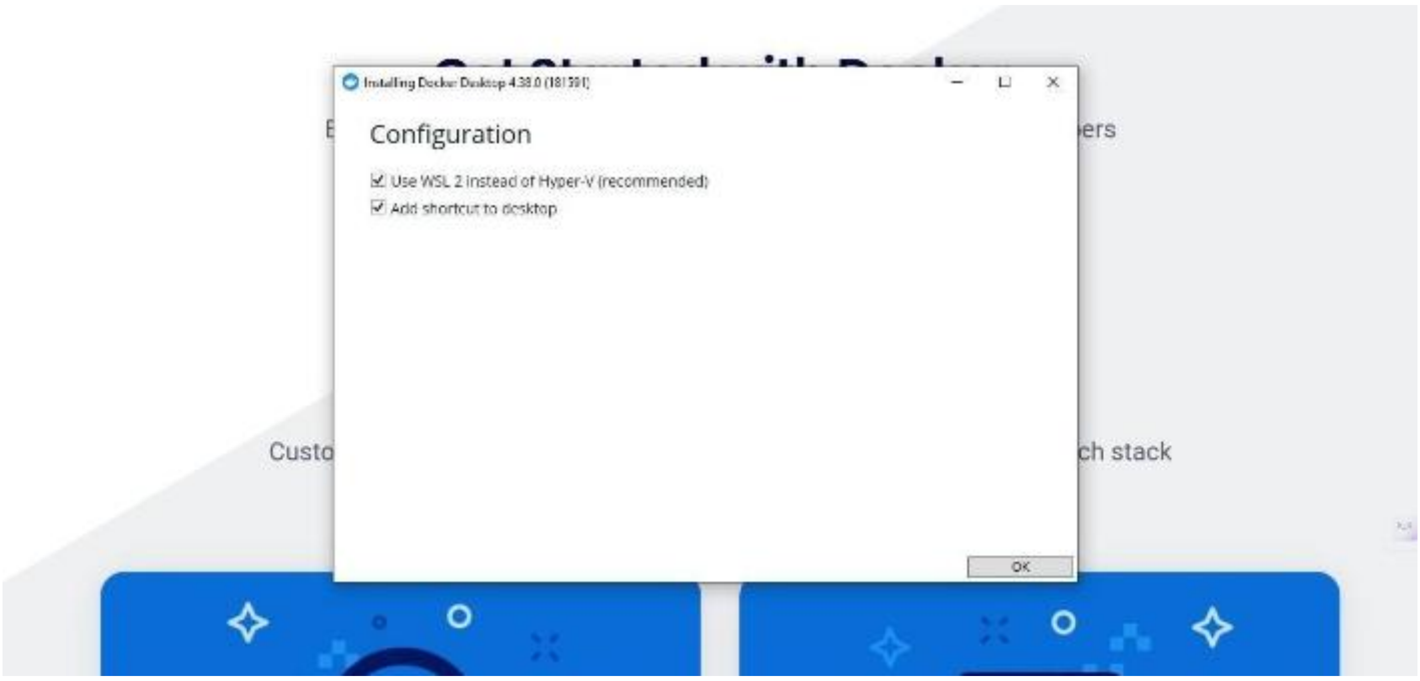
```
C:\Users\202>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\202>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS          PORTS      NAMES
052aecb0ee88   redis     "docker-entrypoint.s…"   About a minute ago   Up 4 seconds    6379/tcp   container121
1c4472744083   redis     "docker-entrypoint.s…"   6 minutes ago        Up 10 seconds   6379/tcp   modest_herschel

C:\Users\202>
```



```
[
    {
        "Id": "1c44727440831475b093dcbf93163064b819bdd9ad8378bb3a4fa847dc411d80",
        "Created": "2024-03-13T03:19:03.418741433Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 2112,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2024-03-13T03:32:13.750463204Z",
            "FinishedAt": "2024-03-13T03:32:13.145321277Z"
        },
        "Image": "sha256:170a1e90f8436daa6778aeea3926e716928826c215ca23a8dfd8055f663f9428",
        "ResolvConfPath": "/var/lib/docker/containers/1c44727440831475b093dcbf93163064b819bdd9a
```



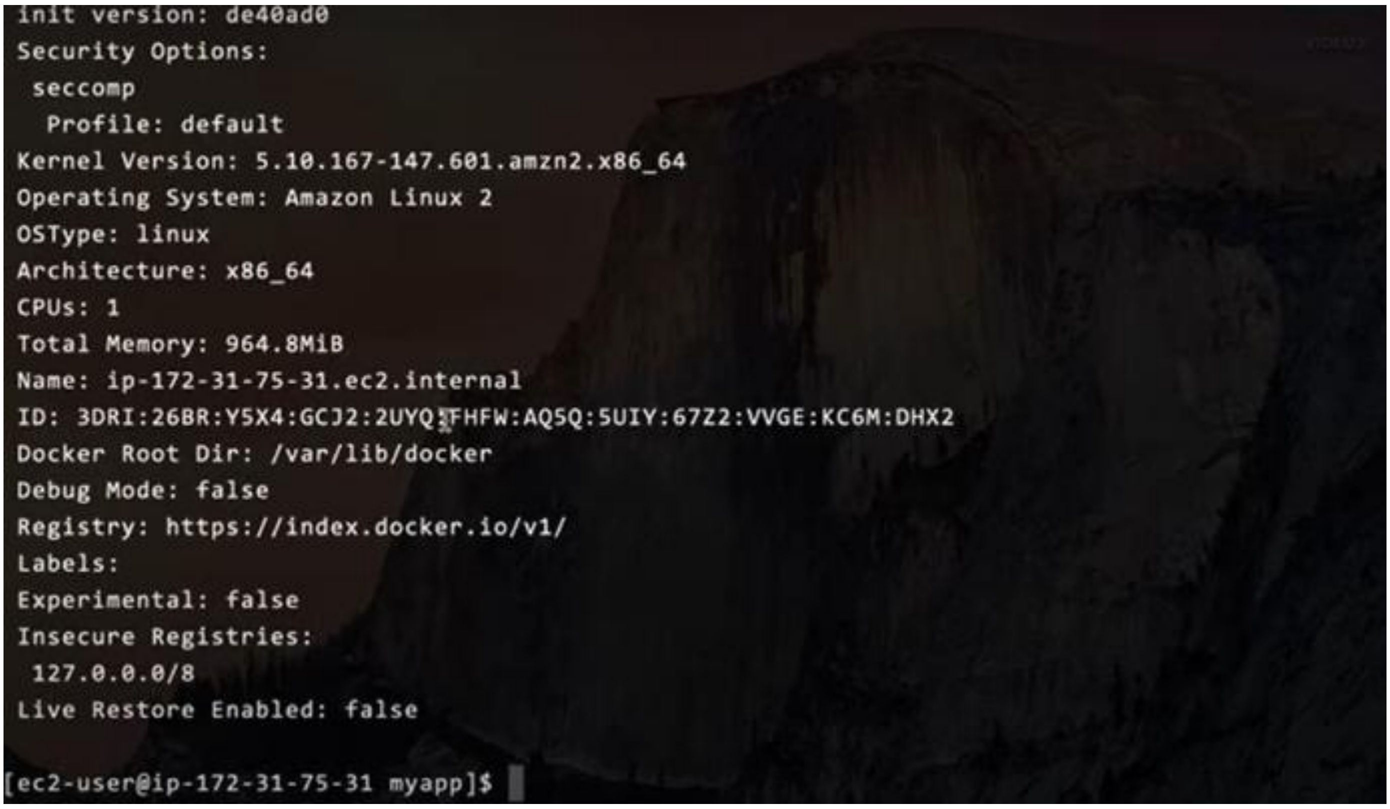
```
C:\Users\202>docker commit 1c44727440083 new_image_name:redis2
sha256:33e4284a7e92a4a1331555d01f6e078fc496e3a3ed8eb7f84f2678261ad07e83

C:\Users\202>docker images
REPOSITORY       TAG       IMAGE ID       CREATED          SIZE
new_image_name   redis2    33e4284a7e92   4 seconds ago    138MB
new_image_name   tag       61ab016507fa   36 seconds ago   138MB
redis            latest    170a1e90f843   2 months ago     138MB
```



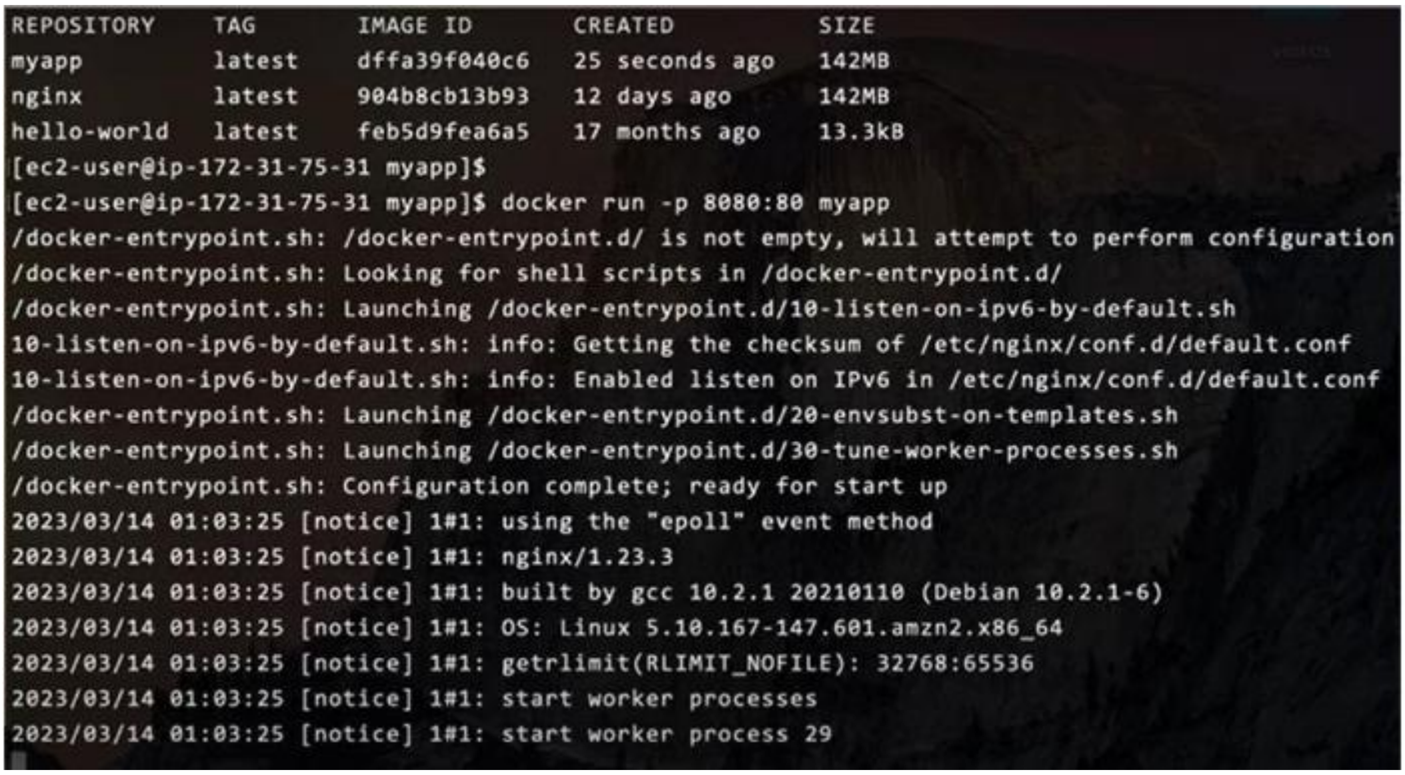
Installing Docker Desktop 4.38.0 (181591)

Configuration

☑ Use WSL 2 instead of Hyper-V (recommended)
☑ Add shortcut to desktop



```
init version: de40ad0
Security Options:
 seccomp
  Profile: default
Kernel Version: 5.10.167-147.601.amzn2.x86_64
Operating System: Amazon Linux 2
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 964.8MiB
Name: ip-172-31-75-31.ec2.internal
ID: 3DRI:26BR:Y5X4:GCJ2:2UYQ:FHFW:AQ5Q:5UIY:67Z2:VVGE:KC6M:DHX2
Docker Root Dir: /var/lib/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false

[ec2-user@ip-172-31-75-31 myapp]$
```



```
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
myapp         latest    dffa39f040c6   25 seconds ago   142MB
nginx         latest    904b8cb13b93   12 days ago      142MB
hello-world   latest    feb5d9fea6a5   17 months ago    13.3kB
[ec2-user@ip-172-31-75-31 myapp]$
[ec2-user@ip-172-31-75-31 myapp]$ docker run -p 8080:80 myapp
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/03/14 01:03:25 [notice] 1#1: using the "epoll" event method
2023/03/14 01:03:25 [notice] 1#1: nginx/1.23.3
2023/03/14 01:03:25 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/03/14 01:03:25 [notice] 1#1: OS: Linux 5.10.167-147.601.amzn2.x86_64
2023/03/14 01:03:25 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 32768:65536
2023/03/14 01:03:25 [notice] 1#1: start worker processes
2023/03/14 01:03:25 [notice] 1#1: start worker process 29
```