

EXPERIMENT NO. 9

TO UNDERSTAND DOCKER ARCHITECTURE AND CONTAINER LIFE CYCLE, INSTALL DOCKER AND EXECUTE DOCKER COMMANDS TO MANAGE IMAGES AND INTERACT WITH CONTAINERS

Theory:

Introduction to Docker

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. Containers package applications with all dependencies, ensuring consistency across different computing environments.

Key Features of Docker:

- Lightweight and efficient
- Portability across different platforms
- Isolation of applications
- Fast deployment and scaling

Docker Architecture

Docker follows a client-server architecture consisting of the following key components:

1. Docker Client

- The command-line interface (CLI) that communicates with the Docker daemon.
- Commands like `docker run`, `docker pull`, and `docker stop` are executed from the client.

2. Docker Daemon (dockerd)

- The background service that manages containers and images.
- It listens to requests from the Docker client and manages container execution.

3. Docker Images

- Read-only templates used to create containers.
- They include application code, dependencies, libraries, and runtime environment.

4. Docker Containers

- Running instances of Docker images.
- Each container is isolated from the host system and other containers.

5. Docker Registry

- A repository to store and distribute Docker images.
- Examples: Docker Hub, AWS Elastic Container Registry (ECR).

Container Life Cycle

The Docker container life cycle includes the following stages:

1. **Create** - A container is created from an image but not yet running.
2. **Start** - The container begins execution.
3. **Pause/Unpause** - Temporarily suspends and resumes container processes.
4. **Stop** - Gracefully stops a running container.
5. **Restart** - Stops and then starts the container again.
6. **Kill** - Forcefully stops the container.
7. **Remove** - Deletes the container permanently.

Installing Docker

Follow these steps to install Docker on a Linux system:

1. Update the package repository:
2. `sudo apt update`
3. Install dependencies:
4. `sudo apt install apt-transport-https ca-certificates curl software-properties-common`
5. Add the official Docker repository:
6. `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg`
7. `echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu focal stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`
8. Install Docker:
9. `sudo apt update`
10. `sudo apt install docker-ce docker-ce-cli containerd.io`
11. Verify installation:
12. `docker --version`

Essential Docker Commands

1. Managing Docker Images

- **Pull an image from Docker Hub**
- `docker pull <image_name>`
- **List available images**
- `docker images`
- **Remove an image**
- `docker rmi <image_id>`

2. Working with Containers

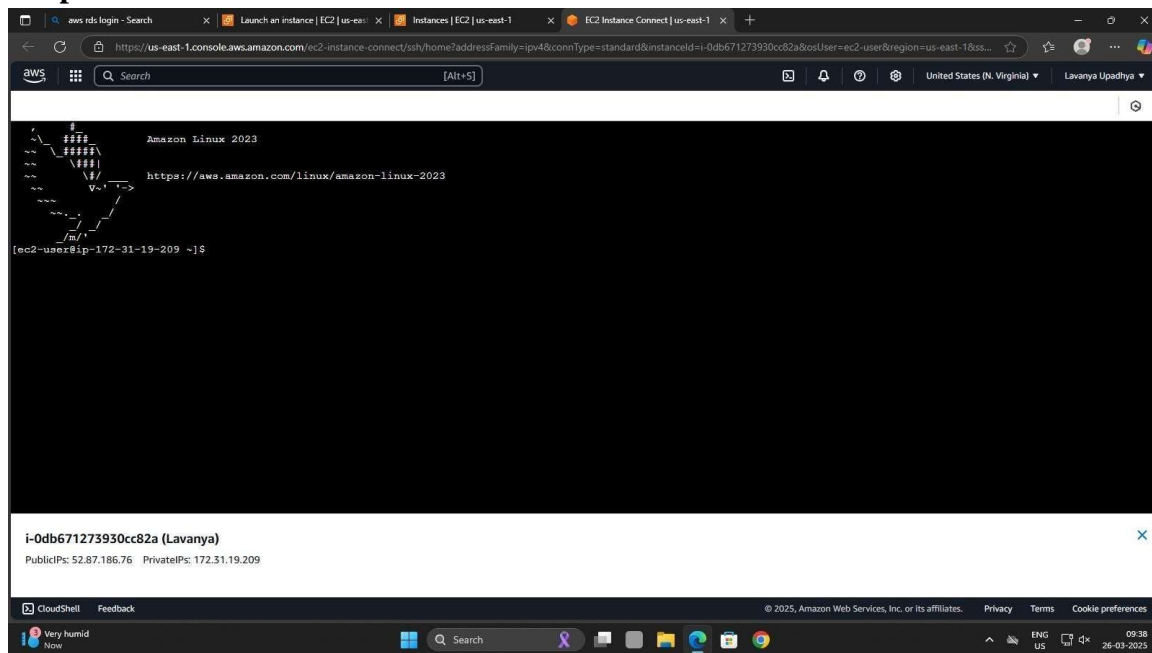
- **Run a container**

- `docker run -d -p 80:80 nginx`
- **List running containers**
- `docker ps`
- **List all containers (including stopped ones)**
- `docker ps -a`
- **Stop a running container**
- `docker stop <container_id>`
- **Remove a container**
- `docker rm <container_id>`

3. Interacting with Containers

- **Access a running container's shell**
- `docker exec -it <container_id> /bin/bash`
- **View logs of a container**
- `docker logs <container_id>`

Output:



Heramb Vengurlekar

T23 – 120

The screenshot displays the AWS CloudShell interface for an EC2 instance named 'i-0db671273930cc82a (Lavanya)'. The terminal shows the following commands and output:

```
[ec2-user@ip-172-31-19-209 ~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
[ec2-user@ip-172-31-19-209 ~]$ sudo usermod -aG docker ec2-user
[ec2-user@ip-172-31-19-209 ~]$ sudo docker pull philippaul/node-mysql-app:02
02: Pulling from philippaul/node-mysql-app
2ef1d741c74: Pull complete
b253aeafaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df589451: Pull complete
6f51ee005dea: Pull complete
5f32ed3c327: Pull complete
0c8cc2f24a4d: Pull complete
0d27a8e86132: Pull complete
b35ca9a95db0: Pull complete
46a182df3db1: Pull complete
f5b1a7ebae97: Pull complete
ff7978b844b1: Pull complete
Digest: sha256:f7c1cfff42a2f4a40b626b0d03f8b83bbcbef3f88d682cd43f395bf9e42966b
Status: Downloaded newer image for philippaul/node-mysql-app:02
docker.io/philippaul/node-mysql-app:02
[ec2-user@ip-172-31-19-209 ~]$ docker run --rm -p 80:3000 -e DBHOST=""C
[ec2-user@ip-172-31-19-209 ~]$ sudo docker run --rm -p 80:3000 -e DB_HOST="database-1.cqt2mok2aamt.us-east-1.rds.amazonaws.com" -e DB_USER="admin" -e DB_PASSWORD="lavanyaupadhyas" -d philippaul/node-mysql-app:02
>
[ec2-user@ip-172-31-19-209 ~]$ sudo docker run --rm -p 80:3000 -e DB_HOST="database-1.cqt2mok2aamt.us-east-1.rds.amazonaws.com" -e DB_USER="admin" -e DB_PASSWORD="laupadhyas" -d philippaul/node-mysql-app:02
ee7dbab7be9ce074d87e67dbel4570b87464f7930f3639b4b4eedcafc9734f
[ec2-user@ip-172-31-19-209 ~]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
ee7dbab7be9        philippaul/node-mysql-app:02   "docker-entrypoint.s..." 28 seconds ago      Up 27 seconds      0.0.0.0:80->3000/tcp, :::80->3000/tcp   strange_kalam
[ec2-user@ip-172-31-19-209 ~]$
```

Below the terminal output, the instance details for 'i-0db671273930cc82a (Lavanya)' are shown, including Public IP: 52.87.186.76 and Private IP: 172.31.19.209.

The bottom part of the screenshot shows the AWS CloudShell interface again, displaying the installed packages for the instance:

```
Installed:
containerd-1.7.25-1.amzn2023.0.1.x86_64      docker-25.0.8-1.amzn2023.0.1.x86_64      iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
iptables-nft-1.8.8-3.amzn2023.0.2.x86_64     libcgrouper-3.0-1.amzn2023.0.1.x86_64     libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
libnftnl-1.0.1-19.amzn2023.0.2.x86_64        libnftnl-1.2.2-2.amzn2023.0.2.x86_64     pigz-2.5-1.amzn2023.0.3.x86_64
runc-1.2.4-1.amzn2023.0.1.x86_64
```

The terminal output shows the following commands and output:

```
[ec2-user@ip-172-31-19-209 ~]$ sudo service dpcker start
Redirecting to /bin/systemctl start dpcker.service
Failed to start dpcker.service: Unit dpcker.service not found.
[ec2-user@ip-172-31-19-209 ~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
[ec2-user@ip-172-31-19-209 ~]$ sudo usermod -aG docker ec2-user
[ec2-user@ip-172-31-19-209 ~]$ sudo docker pull philippaul/node-mysql-app:02
02: Pulling from philippaul/node-mysql-app
2ef1d741c74: Pull complete
b253aeafaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df589451: Pull complete
6f51ee005dea: Pull complete
5f32ed3c327: Pull complete
0c8cc2f24a4d: Pull complete
0d27a8e86132: Pull complete
b35ca9a95db0: Pull complete
46a182df3db1: Pull complete
f5b1a7ebae97: Pull complete
ff7978b844b1: Pull complete
Digest: sha256:f7c1cfff42a2f4a40b626b0d03f8b83bbcbef3f88d682cd43f395bf9e42966b
Status: Downloaded newer image for philippaul/node-mysql-app:02
docker.io/philippaul/node-mysql-app:02
[ec2-user@ip-172-31-19-209 ~]$
```

Below the terminal output, the instance details for 'i-0db671273930cc82a (Lavanya)' are shown, including Public IP: 52.87.186.76 and Private IP: 172.31.19.209.

The screenshot displays two overlapping windows from the AWS Management Console. The top window, titled 'Launch an instance', shows a green success banner for instance `i-0db671273930cc82a`. Below this, the 'Next Steps' section offers several actions: 'Create billing and free tier usage alerts', 'Connect to your instance', 'Connect an RDS database', 'Create EBS snapshot policy', 'Manage detailed monitoring', 'Create Load Balancer', 'Create AWS budget', and 'Manage CloudWatch alarms'. The bottom window, titled 'Create database', shows the 'Choose a database creation method' section with 'Standard create' selected. The 'Engine options' section lists 'Aurora (MySQL Compatible)', 'MySQL' (selected), 'Aurora (PostgreSQL Compatible)', 'PostgreSQL', 'MariaDB', and 'Oracle'. A sidebar on the right provides details about MySQL on RDS, including its popularity and supported features like database size up to 64 TiB, General Purpose, Memory Optimized, and Burstable Performance instance classes, automated backup and point-in-time recovery, and up to 15 Read Replicas per instance.

Success
Successfully initiated launch of instance `i-0db671273930cc82a`

► Launch log

Next Steps

What would you like to do next with this instance, for example "create alarm" or "create backup"

Create billing and free tier usage alerts
To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.
[Create billing alerts](#)

Connect to your instance
Once your instance is running, log into it from your local computer.
[Connect to instance](#)
[Learn more](#)

Connect an RDS database
Configure the connection between an EC2 instance and a database to allow traffic flow between them.
[Connect an RDS database](#)
[Create a new RDS database](#)
[Learn more](#)

Create EBS snapshot policy
Create a policy that automates the creation, retention, and deletion of EBS snapshots.
[Create EBS snapshot policy](#)

Manage detailed monitoring
Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the instance is monitored every minute.

Create Load Balancer
Create an application, network gateway or classic Elastic Load Balancing load balancer.

Create AWS budget
AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage.

Manage CloudWatch alarms
Create or update Amazon CloudWatch alarms for the instance.

Create database

Choose a database creation method

☒ Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

☐ Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type

☐ Aurora (MySQL Compatible)

☒ MySQL

☐ MariaDB

☐ Aurora (PostgreSQL Compatible)

☐ PostgreSQL

☐ Oracle

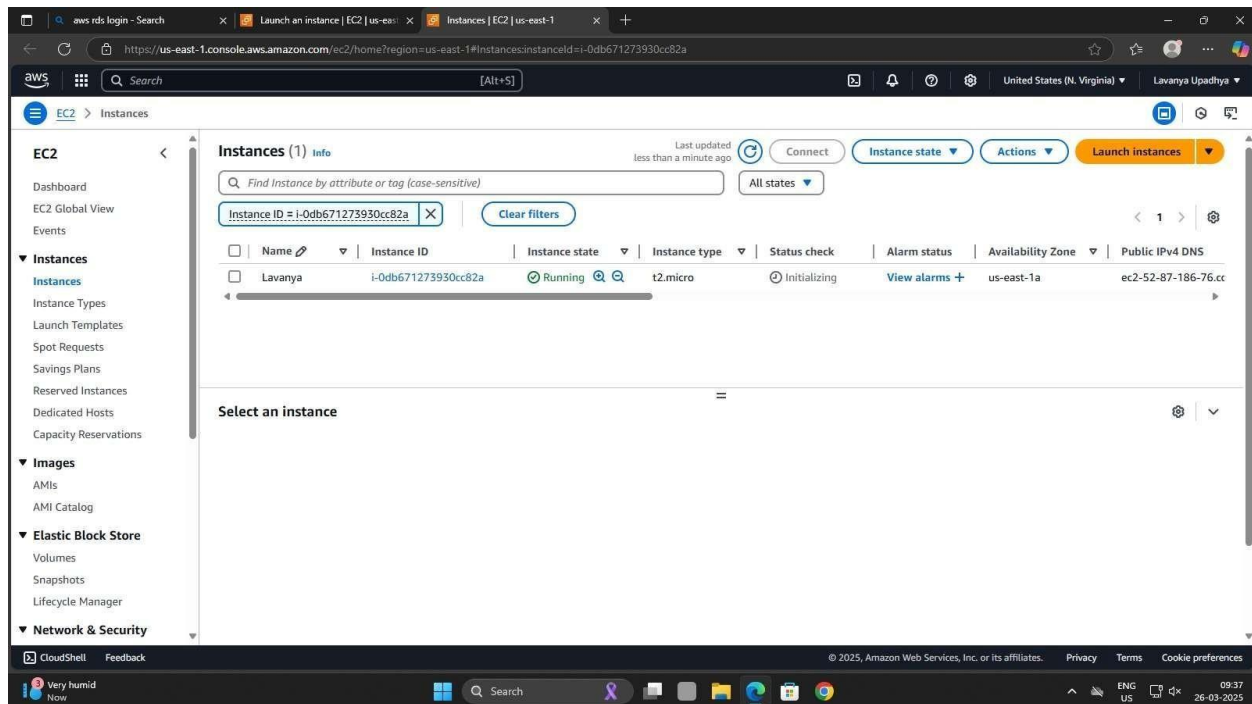
MySQL

MySQL is the most popular open source database in the world. MySQL on RDS offers the rich features of the MySQL community edition with the flexibility to easily scale compute resources or storage capacity for your database.

- Supports database size up to 64 TiB.
- Supports General Purpose, Memory Optimized, and Burstable Performance instance classes.
- Supports automated backup and point-in-time recovery.
- Supports up to 15 Read Replicas per instance, within a single Region or 5 read replicas cross-region.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

09:25 26-03-2025



Conclusion

In this lab, we explored Docker's architecture, the container life cycle, and performed practical operations such as installing Docker, pulling images, and managing containers. These fundamental concepts are essential for deploying and managing applications in a containerized environment.