

## Task 1

1 - Explain the difference between contextualization and orchestration processes.

Contextualization in Docker refers to the process of defining the environment of a Docker container. This is done using a Dockerfile, which contains a set of instructions to build an image. On the other hand, orchestration is the process of managing multiple containers, often across multiple machines. It involves tasks like scaling the application, ensuring fault tolerance, and managing networking between containers.

2 - Explain the followings commands and concepts:

i) Contents of the docker file used in this task.

```
FROM ubuntu:22.04
RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get install sl
ENV PATH="${PATH}:/usr/games/"
CMD ["echo", "Data Engineering-I."]
```

The Dockerfile used in this task is a text file that contains all the commands needed to build a Docker image. It starts from a base Ubuntu 22.04 image, updates the system, installs the `sl` package (which displays animations aimed to correct users who accidentally enter `sl` instead of `ls`), and sets an environment variable to include `/usr/games/` in the `PATH`. It runs the `echo` command to print the message "Data Engineering-I." to the console when container starts.

ii) Explain the command

```
# docker run -it mycontainer/first:v1 bash
```

- This command starts a new container from the `mycontainer/first:v1` image in interactive mode and runs the 'bash' command in it.

iii) Show the output and explain the following commands:

```
# docker ps
```

```
ubuntu@samir-mehdiyev-a4:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
```

It lists all running Docker containers.

```
# docker images
```

```
ubuntu@samir-mehdiyev-a4:~$ docker images
REPOSITORY      TAG          IMAGE ID            CREATED          SIZE
mycontainer/first v1           30fe335283a9       4 minutes ago   136MB
ubuntu           22.04       3db8720ecbf5       2 weeks ago     77.9MB
```

It lists all Docker images on the host.

```
# docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PI
DS							

It displays a live stream of containers resource usage statistics.

3—What is the difference between `docker run`, `docker exec` and `docker build` commands?

- `docker run` creates a new container and starts it.
- `docker exec` runs a command in a running container.
- `docker build` builds an image from a Dockerfile.

4 - Create an account on DockerHub and upload your newly built container to your DockerHub area. (Watch the UI and think: is it uploading the entire image, or just your changes? How is this advantageous?) Briefly explain how DockerHub is used. Make your container publicly available and report the name of your publicly available container.

DockerHub is a cloud-based registry service that allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. When you push an image to DockerHub, it uses layering to ensure that only the layers that have changed are pushed, saving bandwidth and time.

My container: samirmehd1yev/samir\_a4\_1:v1

5 - Explain the difference between `docker build` and `docker-compose` commands.

The `docker build` command is used to build Docker images from a Dockerfile. It allows you to specify the context (the directory containing the Dockerfile) and tag the resulting image with a name and optionally a version.

On the other hand, the `docker-compose` command is used to define and run multi-container Docker applications. It uses a YAML file (named **docker-compose.yml**) to specify the services, networks, and volumes required for your application. It can build, start, stop, and manage multiple containers as a single application stack.

## Task 2

1 – Explain your docker-compose configuration file.

```
version: '3.8'
services:
  spark-master:
    image: bitnami/spark:3.3.2
    environment:
      - SPARK_MODE=master
    ports:
      - '8080:8080'
  spark-worker-1:
    image: bitnami/spark:3.3.2
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
  spark-worker-2:
    image: bitnami/spark:3.3.2
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
```

The Docker Compose configuration file is a YAML file that defines services, networks, and volumes. Each service represents a container that is part of your application. In the context of a Spark cluster:

- The `version` field specifies the version of Docker Compose file format to be used.
- Under `services`, three services are defined: `spark-master`, `spark-worker-1`, and `spark-worker-2`. Each service uses the `bitnami/spark:3.3.2` Docker image which is the most popular spark image in the docker hub.
- The `spark-master` service runs in `master` mode, as specified by the `SPARK_MODE=master` environment variable.
- The `spark-worker-1` and `spark-worker-2` services run in `worker` mode, as specified by the `SPARK_MODE=worker` environment variable. They connect to the master node via the `SPARK_MASTER_URL=spark://spark-master:7077` environment variable.
- The master node's web UI is accessible on port 8080 of the host machine, as specified by the `ports` configuration.

Here is the web UI of final result, it is accessible via floating ip of vm on port 8080, or in local via localhost:8080.

 **Spark Master at spark://49055c416839:7077**

URL: spark://49055c416839:7077  
Alive Workers: 2  
Cores in use: 2 Total, 0 Used  
Memory in use: 2.0 GiB Total, 0.0 B Used  
Resources in use:  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

### Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240228133900-172.20.0.2-34075	172.20.0.2:34075	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20240228133900-172.20.0.3-38491	172.20.0.3:38491	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

### Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

### Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

2 – What is the format of the `docker-compose` compatible configuration file?

The Docker Compose configuration file is written in YAML (YAML Ain't Markup Language). It starts with a version number, followed by services, networks, and volumes. Each service represents a container and has its own configuration such as build context, environment variables, ports, volumes, etc.

3 - What are the limitations of `docker-compose`?

While Docker Compose is a powerful tool, it has some limitations:

- It's primarily designed for development and testing environments. For production environments, tools like Docker Swarm or Kubernetes are more suitable.
- Docker Compose does not support auto-scaling of services.
- It does not provide a built-in mechanism for rolling updates.
- Docker Compose runs on a single host. If you need to orchestrate containers over multiple hosts, you'll need a tool like Docker Swarm or Kubernetes.
- It does not have built-in service discovery. Services communicate with each other through links or shared networks, but there's no DNS-based service discovery like in Docker Swarm or Kubernetes.