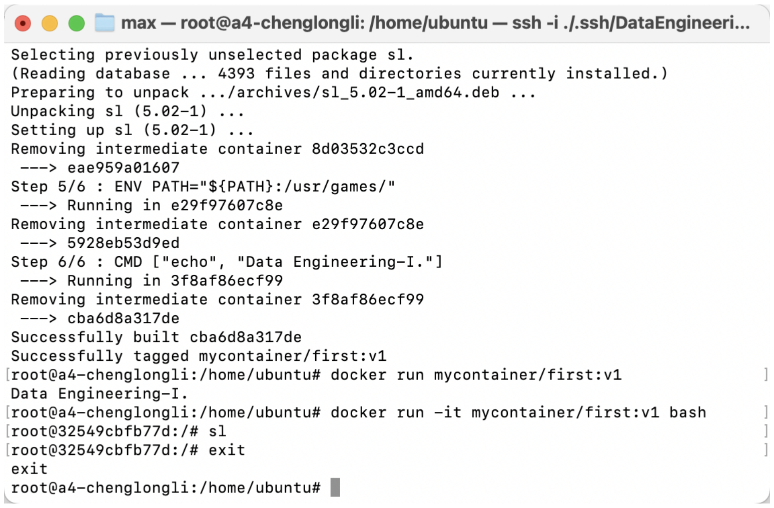
DataEngineering I A4

Chenglong Li

Task 1


max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineeri...

```
Selecting previously unselected package sl.
(Reading database ... 4393 files and directories currently installed.)
Preparing to unpack .../archives/sl_5.02-1_amd64.deb ...
Unpacking sl (5.02-1) ...
Setting up sl (5.02-1) ...
Removing intermediate container 8d03532c3ccd
 ---> eae959a01607
Step 5/6 : ENV PATH="${PATH}:/usr/games/"
 ---> Running in e29f97607c8e
Removing intermediate container e29f97607c8e
 ---> 5928eb53d9ed
Step 6/6 : CMD ["echo", "Data Engineering-I."]
 ---> Running in 3f8af86ecf99
Removing intermediate container 3f8af86ecf99
 ---> cba6d8a317de
Successfully built cba6d8a317de
Successfully tagged mycontainer/first:v1
root@a4-chenglongli:/home/ubuntu# docker run mycontainer/first:v1
Data Engineering-I.
root@a4-chenglongli:/home/ubuntu# docker run -it mycontainer/first:v1 bash
root@32549cbfb77d:/# sl
root@32549cbfb77d:/# exit
exit
root@a4-chenglongli:/home/ubuntu#
```

Question 1

- **Contextualization**
  - In cloud computing contextualization means providing customized computing environment.
  - Allows a virtual machine instance to learn about its cloud environment and user requirement (the ‘context’) and configure itself to run correctly.
- **Orchestration**
  - Orchestration is a process of resource contextualization based on the automation available in the cloud systems.

  - Essential for large complex applications.
  - A process at the level of Platform as a Service (PaaS).

Contextualisation focuses on making information or activities more relevant or easier to understand by providing a framework or context, while orchestration focuses on coordinating components or resources efficiently and effectively to achieve a specific goal.

Question 2

i)

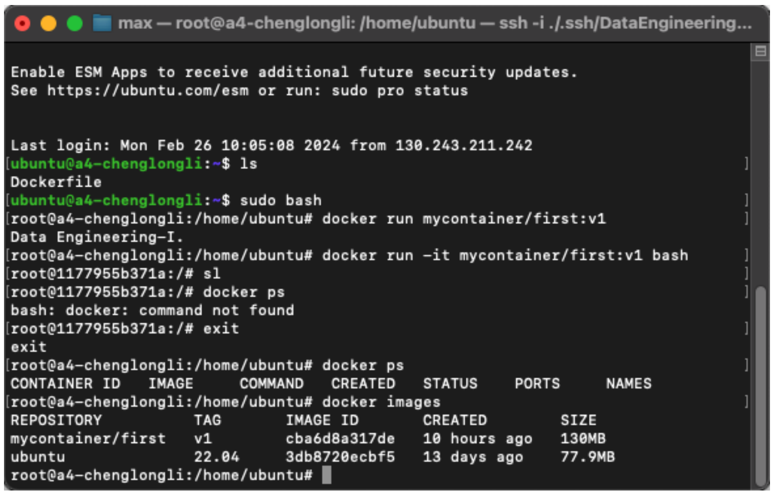
A Docker file defines how to build a Docker image and contains a set of instructions and parameters.
The meaning of each line of instruction:

- FROM ubuntu:22.04: Specify the base image as Ubuntu 22.04, which all build steps will be based on.
- RUN apt-get update: Update the package list of Ubuntu.
- RUN apt-get -y upgrade: Upgrade all installed packages to the latest version.
- RUN apt-get install sl: Installs the sl command.
- ENV PATH="$PATH:/usr/games/": Set the environment variable PATH to add the /usr/games/ directory, making it possible to run programs under /usr/games/ directly.
- CMD ["echo", "Data Engineering-I."]: Specify the default command to be executed when the container is started, in this case it is to print "Data Engineering-I.".

ii)

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

- docker run: Start a new docker container.
- -it: Run the container in "interactive mode" and assign a pseudo-terminal.

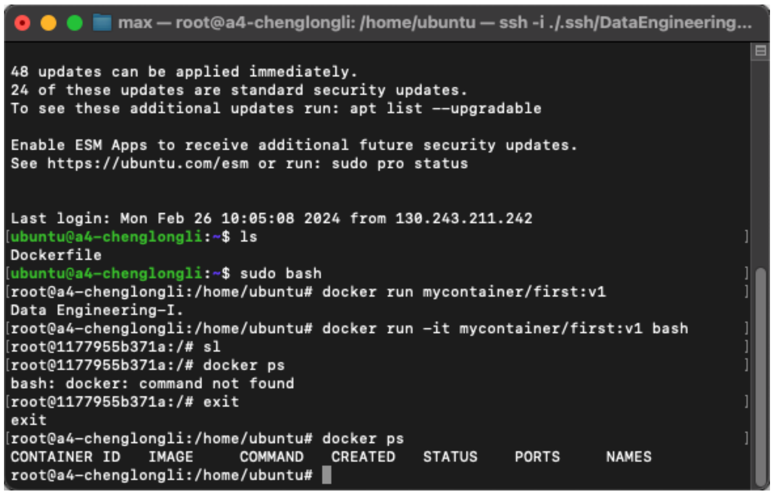

max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineering...

```
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Mon Feb 26 10:05:08 2024 from 130.243.211.242
ubuntu@a4-chenglongli:~$ ls
Dockerfile
ubuntu@a4-chenglongli:~$ sudo bash
root@a4-chenglongli:/home/ubuntu# docker run mycontainer/first:v1
Data Engineering-I.
root@a4-chenglongli:/home/ubuntu# docker run -it mycontainer/first:v1 bash
root@1177955b371a:/# sl
root@1177955b371a:/# docker ps
bash: docker: command not found
root@1177955b371a:/# exit
exit
root@a4-chenglongli:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@a4-chenglongli:/home/ubuntu# docker images
REPOSITORY           TAG       IMAGE ID       CREATED        SIZE
mycontainer/first    v1        cba6d8a317de   10 hours ago   130MB
ubuntu               22.04     3db8720ecbf5   13 days ago    77.9MB
root@a4-chenglongli:/home/ubuntu#
```

- mycontainer/first:v1: Specify the name and label of the image to run.
- bash: Commands to be executed after container startup, start bash terminal.

iii)

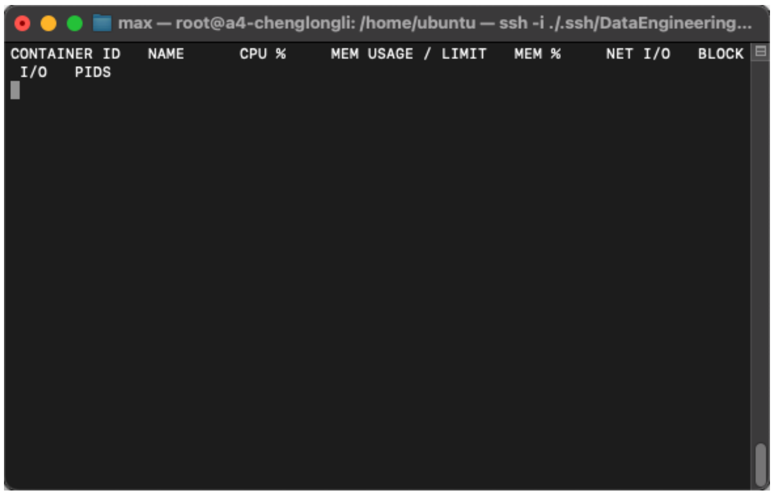

max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineering...

```
48 updates can be applied immediately.
24 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Mon Feb 26 10:05:08 2024 from 130.243.211.242
ubuntu@a4-chenglongli:~$ ls
Dockerfile
ubuntu@a4-chenglongli:~$ sudo bash
root@a4-chenglongli:/home/ubuntu# docker run mycontainer/first:v1
Data Engineering-I.
root@a4-chenglongli:/home/ubuntu# docker run -it mycontainer/first:v1 bash
root@1177955b371a:/# sl
root@1177955b371a:/# docker ps
bash: docker: command not found
root@1177955b371a:/# exit
exit
root@a4-chenglongli:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@a4-chenglongli:/home/ubuntu#
```

- # docker ps: Displays information about the currently running container.


max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineering...

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK
 I/O   PIDS
```

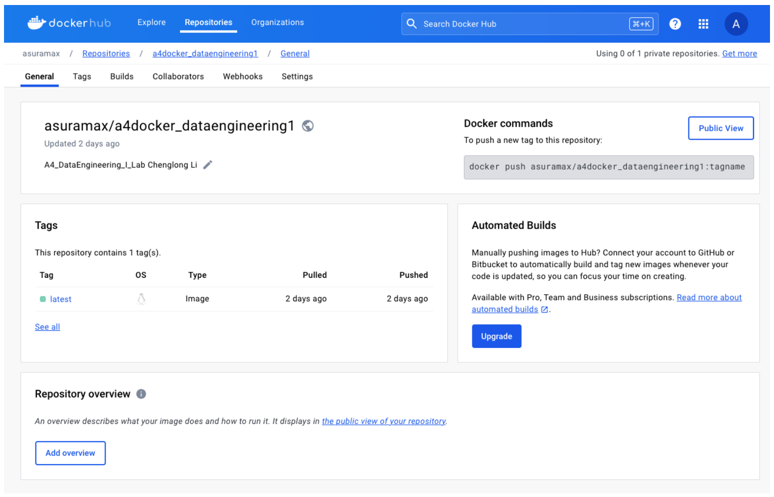
- # docker images: List all Docker images on local storage.
- # docker stats: Displays real-time resource usage for all containers, including CPU, memory usage, network I/O, and more.

Question 3

- docker run: will start up a new container and run a process within that new container.
- docker exec: will execute a command in a container is already running.
- docker build: will build the image defined by Dockerfile.

Question 4

asuramax/a4docker_dataengineering1
When I push an image to DockerHub, Docker checks if the layers of my image already exist on the server. If certain layers already exist, Docker only uploads the missing layers. This layer-based upload has advantages because it saves bandwidth and time. Instead of uploading the entire image, I'm only uploading changes that don't yet exist on DockerHub.



asuramax / Repositories / a4docker_dataengineering1 / General

Using 0 of 1 private repositories. Get more

asuramax/a4docker_dataengineering1

Updated 2 days ago

A4_DataEngineering_I_Lab Chenglong Li

Docker commands

To push a new tag to this repository:

```
docker push asuramax/a4docker_dataengineering1:tagname
```

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	2 days ago	2 days ago

See all

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Repository overview

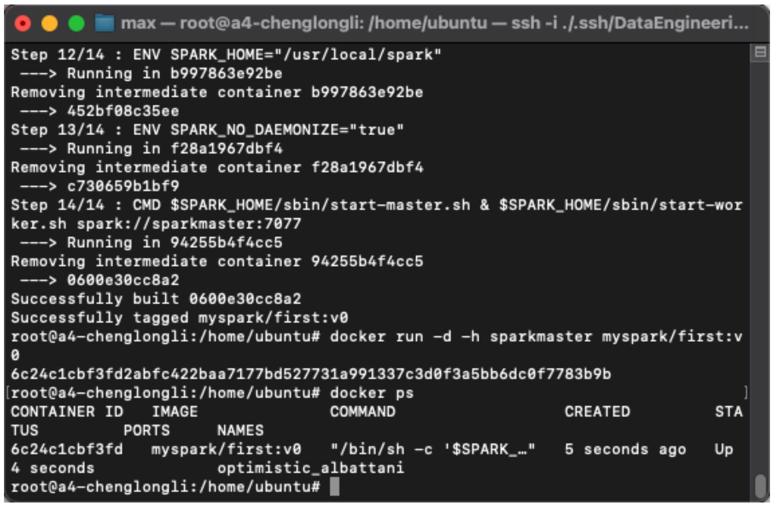
An overview describes what your image does and how to run it. It displays in the public view of your repository.

Question 5

”docker build” focuses on building images and is a single operation related to image creation. Whereas ”docker-compose” is used to define and run multi-container applications, covering the entire workflow from building images to launching containers.

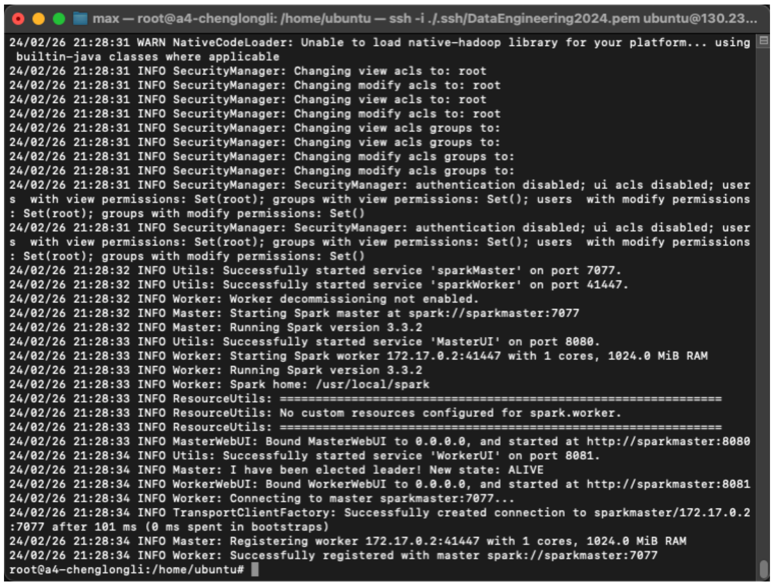
- docker build: docker build will build the image defined by Dockerfile.
- docker-compose: docker-compose can use the compose subcommand, docker compose [-f <arg> ···] [options] [COMMAND] [ARGS ···], to build and manage multiple services in Docker containers.

Task 2



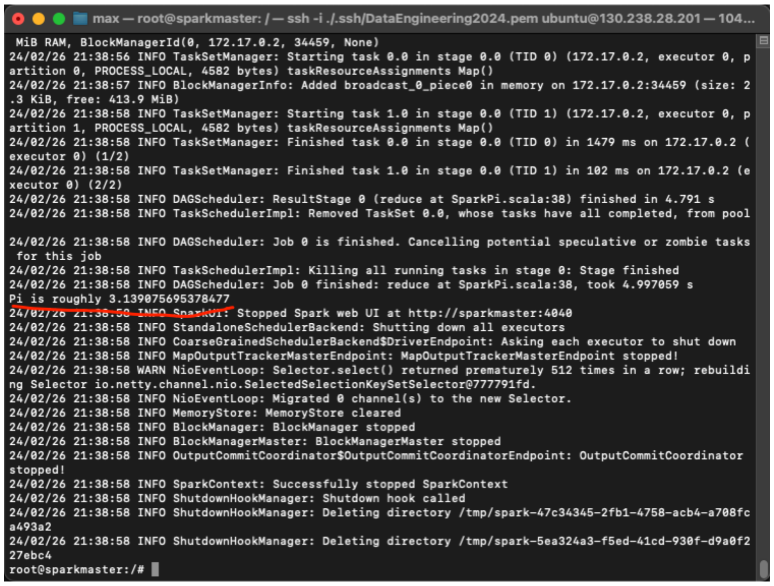
max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineeri...

```
Step 12/14 : ENV SPARK_HOME="/usr/local/spark"
 ---> Running in b997863e92be
Removing intermediate container b997863e92be
 ---> 452bf08c35ee
Step 13/14 : ENV SPARK_NO_DAEMONIZE="true"
 ---> Running in f28a1967dbf4
Removing intermediate container f28a1967dbf4
 ---> c730659b1bf9
Step 14/14 : CMD $SPARK_HOME/sbin/start-master.sh & $SPARK_HOME/sbin/start-wor
ker.sh spark://sparkmaster:7077
 ---> Running in 94255b4f4cc5
Removing intermediate container 94255b4f4cc5
 ---> 0600e30cc8a2
Successfully built 0600e30cc8a2
Successfully tagged myspark/first:v0
root@a4-chenglongli:/home/ubuntu# docker run -d -h sparkmaster myspark/first:v
0
6c24c1cbf3fd2abfc422baa7177bd527731a991337c3d0f3a5bb6dc0f7783b9b
root@a4-chenglongli:/home/ubuntu# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STA
TUS         PORTS     NAMES
6c24c1cbf3fd   myspark/first:v0   "/bin/sh -c '$SPARK_…"   5 seconds ago   Up
4 seconds             optimistic_albattani
root@a4-chenglongli:/home/ubuntu# 
```



max — root@a4-chenglongli: /home/ubuntu — ssh -i ./.ssh/DataEngineering2024.pem ubuntu@130.23...

```
24/02/26 21:28:31 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
 builtin-java classes where applicable
24/02/26 21:28:31 INFO SecurityManager: Changing view acls to: root
24/02/26 21:28:31 INFO SecurityManager: Changing modify acls to: root
24/02/26 21:28:31 INFO SecurityManager: Changing view acls to: root
24/02/26 21:28:31 INFO SecurityManager: Changing modify acls to: root
24/02/26 21:28:31 INFO SecurityManager: Changing view acls groups to:
24/02/26 21:28:31 INFO SecurityManager: Changing view acls groups to:
24/02/26 21:28:31 INFO SecurityManager: Changing modify acls groups to:
24/02/26 21:28:31 INFO SecurityManager: Changing modify acls groups to:
24/02/26 21:28:31 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; user
s  with view permissions: Set(root); groups with view permissions: Set(); users  with modify permissions
: Set(root); groups with modify permissions: Set()
24/02/26 21:28:31 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; user
s  with view permissions: Set(root); groups with view permissions: Set(); users  with modify permissions
: Set(root); groups with modify permissions: Set()
24/02/26 21:28:32 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
24/02/26 21:28:32 INFO Utils: Successfully started service 'sparkWorker' on port 41447.
24/02/26 21:28:32 INFO Worker: Worker decommissioning not enabled.
24/02/26 21:28:32 INFO Master: Starting Spark master at spark://sparkmaster:7077
24/02/26 21:28:32 INFO Master: Running Spark version 3.3.2
24/02/26 21:28:33 INFO Utils: Successfully started service 'MasterUI' on port 8080.
24/02/26 21:28:33 INFO Worker: Starting Spark worker 172.17.0.2:41447 with 1 cores, 1024.0 MiB RAM
24/02/26 21:28:33 INFO Worker: Running Spark version 3.3.2
24/02/26 21:28:33 INFO Worker: Spark home: /usr/local/spark
24/02/26 21:28:33 INFO ResourceUtils: ==============================================================
24/02/26 21:28:33 INFO ResourceUtils: No custom resources configured for spark.worker.
24/02/26 21:28:33 INFO ResourceUtils: ==============================================================
24/02/26 21:28:33 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://sparkmaster:8080
24/02/26 21:28:34 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
24/02/26 21:28:34 INFO Master: I have been elected leader! New state: ALIVE
24/02/26 21:28:34 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://sparkmaster:8081
24/02/26 21:28:34 INFO Worker: Connecting to master sparkmaster:7077...
24/02/26 21:28:34 INFO TransportClientFactory: Successfully created connection to sparkmaster/172.17.0.2
:7077 after 101 ms (0 ms spent in bootstraps)
24/02/26 21:28:34 INFO Master: Registering worker 172.17.0.2:41447 with 1 cores, 1024.0 MiB RAM
24/02/26 21:28:34 INFO Worker: Successfully registered with master spark://sparkmaster:7077
root@a4-chenglongli:/home/ubuntu# 
```


max — root@sparkmaster: / — ssh -i ./.ssh/DataEngineering2024.pem ubuntu@130.238.28.201 — 104...

```
 MiB RAM, BlockManagerId(0, 172.17.0.2, 34459, None)
24/02/26 21:38:56 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0) (172.17.0.2, executor 0, p
artition 0, PROCESS_LOCAL, 4582 bytes) taskResourceAssignments Map()
24/02/26 21:38:57 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 172.17.0.2:34459 (size: 2
.3 KiB, free: 413.9 MiB)
24/02/26 21:38:58 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1) (172.17.0.2, executor 0, p
artition 1, PROCESS_LOCAL, 4582 bytes) taskResourceAssignments Map()
24/02/26 21:38:58 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1479 ms on 172.17.0.2 (
executor 0) (1/2)
24/02/26 21:38:58 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 102 ms on 172.17.0.2 (e
xecutor 0) (2/2)
24/02/26 21:38:58 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 4.791 s
24/02/26 21:38:58 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool

24/02/26 21:38:58 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks
 for this job
24/02/26 21:38:58 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
24/02/26 21:38:58 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 4.997059 s
Pi is roughly 3.139075695378477
24/02/26 21:38:58 INFO SparkUI: Stopped Spark web UI at http://sparkmaster:4040
24/02/26 21:38:58 INFO StandaloneSchedulerBackend: Shutting down all executors
24/02/26 21:38:58 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
24/02/26 21:38:58 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/02/26 21:38:58 WARN NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuildi
ng Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@777791fd.
24/02/26 21:38:58 INFO NioEventLoop: Migrated 0 channel(s) to the new Selector.
24/02/26 21:38:58 INFO MemoryStore: MemoryStore cleared
24/02/26 21:38:58 INFO BlockManager: BlockManager stopped
24/02/26 21:38:58 INFO BlockManagerMaster: BlockManagerMaster stopped
24/02/26 21:38:58 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator
stopped!
24/02/26 21:38:58 INFO SparkContext: Successfully stopped SparkContext
24/02/26 21:38:58 INFO ShutdownHookManager: Shutdown hook called
24/02/26 21:38:58 INFO ShutdownHookManager: Deleting directory /tmp/spark-47c34345-2fb1-4758-acb4-a708fc
a493a2
24/02/26 21:38:58 INFO ShutdownHookManager: Deleting directory /tmp/spark-5ea324a3-f5ed-41cd-930f-d9a0f2
27ebc4
root@sparkmaster:/#
```

Question 1

```
version: "2"
services:
  master:
    image: mycontainer/first:v0
    command: /usr/local/spark/sbin/start-master.sh
    hostname: master
    environment:
      - SPARK_MODE=master
    extra_hosts:
      - "master:127.2.0.101"
    ports:
      - "6066:6066"
      - "7070:7070"
      - "8080:8080"
      - "50070:50070"
  worker:
    image: mycontainer/first:v0
    command: /usr/local/spark/sbin/start-worker.sh
    environment:
      - SPARK_MODE=worker
    links:
      - master
```

This ”docker-compose.yml” configuration file defines a basic Spark cluster with a master and a worker.

- Version: 2
- Services: This file defines two services: master and worker.
    - master:
        - image: The Docker image used for this service is mycontainer/first, labelled v0.
        - command: A command that is executed when the container is started and is used to start the Spark master node.
        - hostname: Set the container's hostname to ”master”.
        - environment: The environment variable SPARK_MODE is defined with a value of master, indicating that this service is the master node for Spark.

    - extra_hosts: Added additional hostname mapping to the container's "/etc/hosts" file to map master to IP: 127.2.0.101.
    - ports: Mapping ports within the container to ports on the host, including ports on the Spark master node and Web UI.
  - worker:
    - links: Defines a link between this service and the master service, allowing worker node containers to access master node containers via the hostname master. This is used in Docker Compose file version 2 for network communication between containers.

Question 2

Compose file version 2 reference.
YAML
The Docker Compose configuration file is the core of Docker Compose and is used to define services, networks and data volumes. The format is YAML and the default path is ./docker-compose.yml, we can use .yml or .yaml extension, currently the latest version of the Compose configuration file format is V3.
YAML is a recursive acronym for "YAML Ain't a Markup Language". When the language was developed, YAML actually meant: "Yet Another Markup Language".
The syntax of YAML is similar to that of other high-level languages, and allows for simple representation of lists, hash tables, scalars, and other data forms. Its use of whitespace indentation and extensive dependency on appearance make it particularly suitable for expressing or editing data structures, various configuration files, dumping debugging content, and document outlines (e.g., many email headers are formatted in a way that is very close to YAML).
Configuration files in YAML have a .yml suffix, e.g. runoob.yml.

Question 3

- We have to waste time and effort by manually installing/updating docker-compose on our server. The Linux package managers (apt, yum) can not help us with this.
- We have to start from scratch when it comes to high availability.

- We don't have health checks available in production with Docker Compose. Unfortunately, docker run and docker-compose won't re-create containers that failed a built-in health check.
- We can't replace a container without downtime. No rolling updates are available.
- We don't have access to the yum/apt packages. Docker Compose requires pip install or manual updates.
- Docker Compose fails to prove itself on reboots. In fact, it is only a wrapper around the Docker API. This means that we can not count on it running as a separate process. This way, all the YAML changes (including containers) are erased from Docker's memory once it restarts.

Task 3

Frameworks Overview

1. Kubernetes[1]: Kubernetes is an open source container orchestration system. It manages containerized applications across multiple hosts for deploying, monitoring, and scaling containers. Originally created by Google, in March of 2016 it was donated to the Cloud Native Computing Foundation (CNCF). Kubernetes, or “k8s” or “kube” for short, allows the user t declare the desired state of an application using concepts such as “deployments” and “services.” Kubernetes can help us deliver and manage containerized, legacy, and cloud-native apps, as well as those being refactored into microservices.

2. Docker Swarm[2]: A Docker Swarm is a container orchestration tool running the Docker application. It has been configured to join together in a cluster. The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster are referred to as nodes.

3. Apache Mesos: Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Kafka, Elasticsearch) with API’s for resource management and scheduling across entire datacenter and cloud environments.Mesos works by introducing a layer of abstraction that provides a means to use entire datacenters as if they were a single, large server. Instead of focusing on one application running on a specific server, Mesos’s resource isolation allows for multitenancy—the ability to run multiple applications on a single machine—leading to more efficient use of computing resources[3].

4. HashiCorp Nomad: HashiCorp Nomad is an easy-to-use workload manager that enables users to schedule tasks and deploy applications in a containerized or noncontainerized infrastructure. It allows you to write code and build software using declarative infrastructure as code[4].

Kubernetes improves Spark cluster by providing advanced scheduling, automatic workload-based scaling, and seamless handling of node failures to ensure high availability and resiliency. Kubernetes’ ecosystem simplifies deployment and management, while its service discovery and load balancing capabilities enhance inter-container communication and improve efficiency.

Docker Swarm provides easier scaling and management of Spark clusters than the original docker-compose through direct Docker native integration. Its built-in load balancing and simple declarative model will help make it easier to deploy and scale Spark nodes, improving resource utilisation and application responsiveness.

Runtime orchestration and contextualisation through frameworks such as Kubernetes or Docker Swarm not only address the limitations of docker-compose in managing complex distributed applications, but also leverage the power of containerisation to achieve scalable, elastic and efficient application deployment.

References

[1] Kubernetes T. Kubernetes. Kubernetes Retrieved May. 2019;24:2019.

[2] Soppelsa F, Kaewkasi C. Native Docker Clustering with Swarm. Packt Publishing; 2017.

[3] Ignazio R. Mesos in action. Simon and Schuster; 2016.

[4] Sabharwal N, Pandey S, Pandey P. In: Getting Started with Nomad. Berkeley, CA: Apress; 2021. p. 201-36. Available from: https://doi.org/10.1007/978-1-4842-7129-2_8.