

Task 1. Introduction to Docker containers and DockerHub

1.Explain the difference between contextualization and orchestration processes.

Contextualization allows a virtual machine instance to learn about its cloud environment and user requirement (the 'context') and configure itself to run correctly. Thus, it can be viewed as the autonomous configuration of individual components. Orchestration is the process of managing a set of automated tasks to create an entire workflow. In Orchestration, a sysadmin sets up a system to perform a series of jobs based on a specific set of rules and parameters. So the person designing the process dictates the desired result. However, the computer can make decisions based on changing circumstances.

2.Explain the followings commands and concepts:

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

Dockerfile is a text file used to build an image, and it contains the instructions to build the image.

FROM: The customized images are all FROM-based images, and the ubuntu:20.04 here is the basic image required for customization. Subsequent operations are based on ubuntu:20.04.

RUN: It contains all the commands that need to be run, here are getting index sources for updated packages, updating the packages, and installing sl.

ENV: It's to set the environment variable.

CMD: It's similar to the RUN instruction, used to run commands (CMD runs in docker run, RUN runs in docker build). Here, the command is to output the string "Data Engineering-I".

ii) Explain the command

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

docker run is to create a new container and run a command on it, the full command should be docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

-it: run the container in interactive mode

mycontainer/first:v1: it's the image name

So this command is to use the image mycontainer/first:v1 to start a container in interactive mode and execute the bash command inside the container.

iii) Show the output and explain the following commands:

```
# docker ps
```

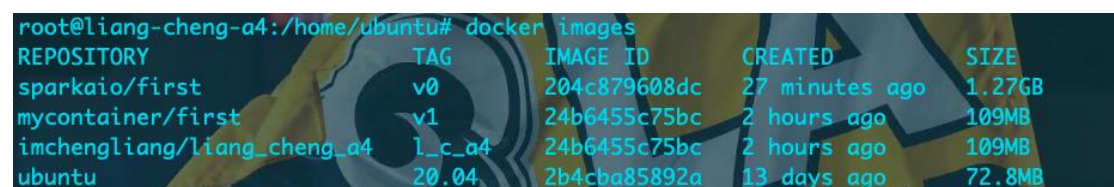
This command can show the list of containers



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f2f7de54aba8	sparkaio/first:v0	"/bin/sh -c 'screen ...'"	34 minutes ago	Up 34 minutes		gallan

```
# docker images
```

This command can show the list of images



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sparkaio/first	v0	204c879608dc	27 minutes ago	1.27GB
mycontainer/first	v1	24b6455c75bc	2 hours ago	109MB
imchengliang/liang_cheng_a4	l_c_a4	24b6455c75bc	2 hours ago	109MB
ubuntu	20.04	2b4cba85892a	13 days ago	72.8MB

docker stats

This command is to display a live stream of containers resource usage statistics

```
root@liang-cheng-a4:/home/ubuntu# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
```

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

docker run is normally used on newly created container. When we want to create a container, start it, and run a process on it, then this command is used. The object of this operation is image because we need to assign an image when we create the container.

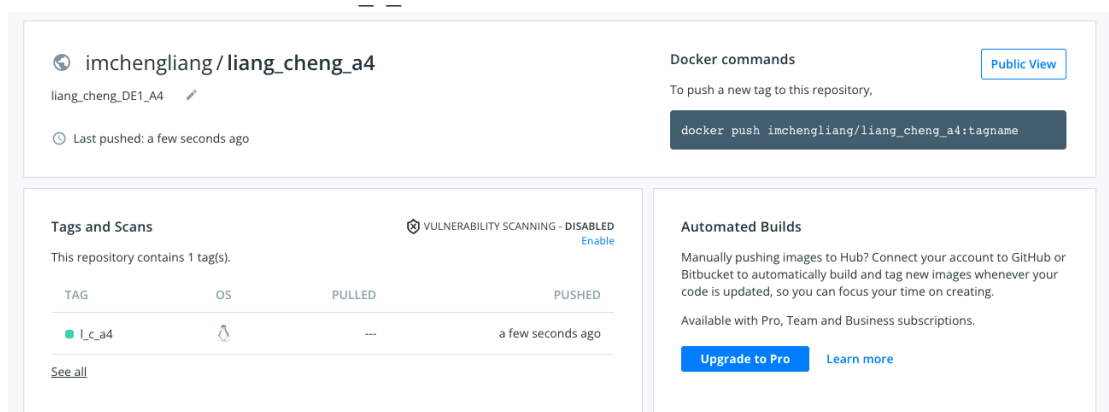
docker exec is used on the currently existed container. If we want to obtain or change things in the specific container, this command is suitable to be used.

docker build is used for creating new image based on the Dockerfile.

4.Create an account on DockerHub and upload your newly built container to your DockerHub area. Explain the usability of DockerHub. Make your container publicly available and report the name of your publicly available container.

DockerHub is a cloud-based repository where Docker users can create, test, store, and distribute container images. Users can access to all public open-source images in this repository, and they are also able to store images on private repository which would only be used by the repository owners.

The name of the container is l_c_a4



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

docker-compose is a tool that can define multi-container applications through a yml file, and can create or manage multiple containers according to the definition of the yml file with a single command. The commands of docker-compose are used to manage the container cluster.

docker build is to create an image using a Dockerfile, so this command is not used for container.

Task 2. Build a multi-container Apache Spark cluster using docker-compose



```
root@liang-cheng-a4:/home/ubuntu/docker-spark# sudo docker-compose ps

```

Name	Command	State	Ports
dockerspark_master_1	/usr/local/spark/sbin/star ...	Up	0.0.0.0:50070->50070/tcp, :::50070->50070/tcp, 0.0.0.0:6066->6066/tcp, :::6066->6066/tcp, 0.0.0.0:7070->7070/tcp, :::7070->7070/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
dockerspark_worker_1	/bin/sh -c screen -d -m \$S ...	Up	

1.Explain your docker-compose configuration file.

version: “2”: It specifies the version of compose file, here is version 2 file

service: define all service information, all the first level keys under service are the name of service, there are 2 services: master and worker

image: it means the docker image used on docker-compose

command: it lists the commands that are needed to be run after the containers start, here are starting spark master and spark worker

ports: it lists the ports that the spark should use

hostname: it's the hostname of spark masker

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

The format of docker-compose configuration file should be YAML or YML. In these kinds of file, data structures are represented by indentation, consecutive items are represented by minus signs, key-value pairs are separated by colons, arrays are enclosed in square brackets [], and hashes are enclosed in curly braces {}.

3.What are the limitations of docker-compose?

For docker-compose, users must manually install/update it because neither apt nor yum can finish step, which might cause the version conflict in actual production. There is no rolling update in docker-compose, which means that all containers can't be replaced during the runtime. It's convenient that we can run multiple containers by docker-compose, but it just wraps the docker API so all the running is not a separate process, and all the docker memory would be gone once it restarts.

Task 3. Introduction to different orchestration and contextualization frameworks

Docker-Compose is used to manage your containers, kind of like a container steward. We write a file, declare the container to be started in this file, configure some parameters, execute this file, Docker will start all containers according to the declared configuration. But Docker-Compose can only manage the current host, which means it can't start Docker containers on other hosts. And this creates network communication challenges when application workloads are distributed across multiple hosts or cloud providers. What's more, when using a Docker Compose-based application, the server running the application must remain running for the application to continue to work. This means that the server running Compose becomes a single point of failure. Because of these limitations, orchestration and contextualization frameworks

like Kubernetes and Docker Swarm become necessary for developers.

Kubernetes is an open-source platform created by Google for container deployment operations, scaling up and down, and automation across clusters of hosts. This production-ready, enterprise-grade, self-healing (auto-scaling, auto-replicating, auto-restart, auto-placement) platform is modular, so it can be deployed for any architecture. Kubernetes also distributes the load among the containers. It aims to alleviate the problems faced by tools and components for running applications in private and public clouds by grouping containers and naming them as logical units. Their power lies in easy scalability, environment-independent portability, and flexible growth. Compared with distributed systems, Kubernetes is more of an all-in-one framework. This is a complex system because it provides strong guarantees for cluster state and a unified set of APIs. This slows down container scaling and deployment. All pods in Kubernetes are distributed among nodes, which provides high availability by tolerant of application failures. The load balancing service in Kubernetes detects unhealthy pods and deletes them. Therefore, this supports high availability. Kubernetes uses its own YAML, API, and client definitions, each of which differs from the standard docker equivalents. That is, we cannot use Docker Compose or the Docker CLI to define containers. When switching platforms, the YAML definitions and commands need to be rewritten.

Docker Swarm is a tool for managing Docker containers on multiple hosts. It can be responsible for helping us start containers and monitor container status. If the container status is abnormal, it will help us restart a new container to provide services, and provides load balancing between services, which Docker-Compose cannot do. Docker Swarm can deploy containers faster than Kubernetes, which allows for faster reaction times to scale on demand. Docker Swarm also provides high availability as services can be replicated across Swarm nodes. The Swarm manager node in Docker Swarm is responsible for the entire cluster and handles the resources of the worker nodes. The Docker Swarm API does not fully contain all of Docker's commands but provides many of the familiar features of Docker. It supports most tools that run with Docker.

So basically speaking, both Kubernetes and Docker Swarm can run many of the same services but may require slightly different approaches in certain details. Their features can greatly make up for the shortcomings of Docker Compose and bring convenience to developers.