Uppsala University

Transformmetoder Projekt

Author:
Your name here

Personnummer: XXX

Filternummer: X

March 6, 2023

1 Task 1. Introduction to Docker containers and DockerHub

1.a Question 1: Explain the difference between contextualization and orchestration processes.

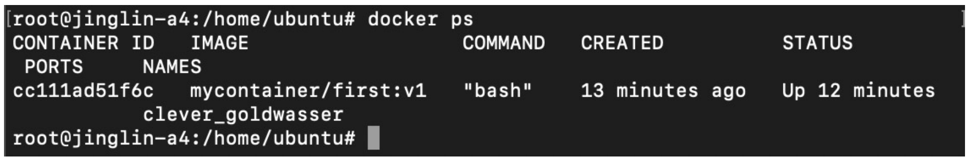
Contextualization means providing a customized computing environment or allowing a virtual machine instance to learn about its cloud environment and user requirement (the ’context’) and configure itself to run correctly.
Orchestration is a process of resource contextualization based on the automation available in the cloud systems. It refers to the process of managing and coordinating different components or services to work together. The aim of orchestration is to automate the management and operation of complex systems, enabling them to scale and adapt to changing demands.

1.b Question 2: Explain the followings commands and concepts:

I) Contents of the docker file used in this task.
FROM ubuntu:22.04: FROM creates a layer from the ubuntu:22.04 Docker image.
RUN apt-get update: Run ’docker build apt-get update’ to update apt.
RUN apt-get -y upgrade: Run ’apt-get -y upgrade’ to update the package information.
RUN apt-get install sl: Run ’apt-get install sl’ to install sl.
ENV PATH=”$ PATH:/usr/games/”: To update the PATH environment variable for the software your container installs.
CMD [”echo”, ”Data Engineering-I.”]: CMD specifies what command to run within the container. Here we want to run ”echo Data-Engineering-I.”.

II) Explain the command # docker run -it mycontainer/first:v1 bash
This command allows the user to run containers interactively. One can achieve this by adding the interactive flag ’-it’ to the ’docker run’ command and by providing a shell (here is bash). ’mycontainer/first:v1’ is the name of the container.

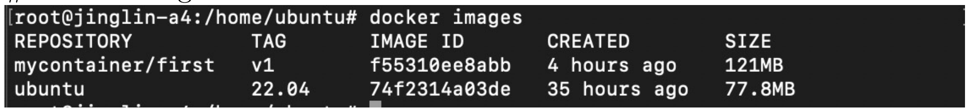
III) Show the output and explain the following commands:
# docker ps



```
[root@jinglin-a4:/home/ubuntu# docker ps
CONTAINER ID   IMAGE                   COMMAND   CREATED          STATUS
 PORTS     NAMES
cc111ad51f6c   mycontainer/first:v1   "bash"    13 minutes ago   Up 12 minutes
           clever_goldwasser
root@jinglin-a4:/home/ubuntu# 
```

This command shows the running containers.

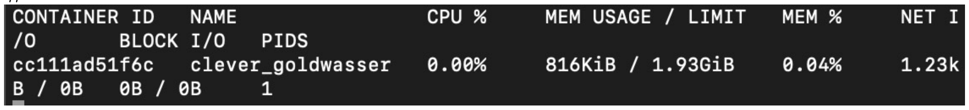
# docker images



```
[root@jinglin-a4:/home/ubuntu# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
mycontainer/first   v1        f55310ee8abb   4 hours ago    121MB
ubuntu              22.04     74f2314a03de   35 hours ago   77.8MB
```

This command will show all top level images, their repository and tags, and their size.

# docker stats



```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT   MEM %     NET I
/O        BLOCK I/O   PIDS
cc111ad51f6c   clever_goldwasser   0.00%     816KiB / 1.93GiB    0.04%     1.23k
B / 0B    0B / 0B     1
```

This command returns a live data stream for running containers. To limit data to one or more specific containers, specify a list of container names or ids separated by a space.

1.c Question 3: What is the difference between docker run, docker exec and docker build commands?

The 'docker run' command first creates a writeable container layer over the specified image, and then starts it using the specified command.
The 'docker build' command builds Docker images from a Dockerfile and a "context". A build's context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the context.
The 'docker exec' command runs a new command in a running container.
In summary, 'docker run' is used to start a new container from an existing image, 'docker exec' is used to run a command inside a running container, and 'docker build' is used to create a new Docker image from a Dockerfile.

1.d Question 4: Briefly explain how DockerHub is used.

My container: jinglin409/my-first-repo:latest
When uploading the container to DockerHub, it uploaded the entire image, not just the changes. This is advantageous because it ensures that the im-

age is complete and can be used to create containers without any missing dependencies.
DockerHub is a cloud-based repository where you can store, manage, and share Docker images. It is used to host and distribute Docker images so that other users can download and use them. You can use DockerHub to share your Docker images with others, collaborate with other developers, and deploy your Docker images to production environments.

1.e Question 5: Explain the difference between docker build and docker-compose commands.

One can use 'docker compose' subcommand to build and manage multiple services in Docker containers. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.
In summary, 'docker build' is used to create a Docker image from a Dockerfile. In contrast 'docker-compose' is used to define and run multi-container Docker applications. 'docker build' is typically used for building custom images for individual containers. In contrast 'docker-compose' is used for running and managing multi-container applications.

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

2.a Question 1: Explain your docker-compose configuration file

The whole docker-compose configuration file is attached beside with command for each line.
Basically, the configuration file has a fixed format, we need to specify the version of the Compose file format, at least one service, and optionally volumes and networks. In my configuration file, I set the docker-compose version to 3.9 which has two services and one network. One of the services is 'spark-master' and the other one is 'spark-worker'. Services refer to the containers' configuration so in each service, I give the different settings for them.

```
version: '3.9'                       # docker-compose version 3.9

services:                            # services to be run
  spark-master:                      # service name
    image: myspark/first:v0         # image name
    container_name: spark-master     # container name
    hostname: sparkmaster            # hostname
    ports:                           # ports to be exposed
      - "8080:8080"                   # port 8080 of the container
                                      # to be exposed to port 8080
                                      # of the host
      - "7077:7077"                   # port 7077 of the container
                                      # to be exposed to port 7077
                                      # of the host
    environment:                     # environment variables
      - SPARK_MODE=master             # SPARK_MODE is set to master
      - "SPARK_MASTER_PORT=7077"      # SPARK_MASTER_PORT is set to
                                      # 7077
      - "SPARK_MASTER_WEBUI_PORT=8080"
      # SPARK_MASTER_WEBUI_PORT is set to 8080

  spark-worker:                      # service name
    image: myspark/first:v0         # image name
    container_name: spark-worker     # container name
    hostname: sparkworker            # hostname
    depends_on:                      # dependency on spark-master
      - spark-master
    ports:                           # ports to be exposed
      - 8080
    environment:                     # environment variables
      - SPARK_MODE=worker             # SPARK_MODE is set to worker
      - "SPARK_MASTER_URL=spark://spark-master:7077"
      # SPARK_MASTER_URL is set to spark://spark-master:7077
      - "SPARK_WORKER_WEBUI_PORT=8080"
      # SPARK_WORKER_WEBUI_PORT is set to 8080

networks:                            # networks to be created
  spark-net:                         # network name
```

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

The compose file is a YAML file defining services, networks, and volumes for a Docker application. YAML is a human-readable data serialization language that is commonly used for configuration files. These YAML rules, both human-readable and machine-optimized, provide an effective way to snapshot the whole project from ten-thousand feet in a few lines. The default path for a Compose file is 'compose.yaml' or 'compose.yml' in working directory. Compose implementations should also support 'docker-compose.yaml' and 'docker-compose.yml' for backwaard compatibility. If both files exist, Compose implementations must prefer canonical 'compose.yaml' one.

2.c Question 3: What are the limitations of docker-compose?

Docker-compose is a great option for small-scale applications that don't require a lot of infrastructure. It's easy to use and can be deployed quickly. However, docker-compose is not very scalable and is not suitable for large-scale applications.
Another limitation is docker run and docker-compose won't re-create containers that failed a built-in health check.
What's more, Docker compose supports only a limited set of networking options, which may be insufficient. For example it dose not support more advanced networking configurations like multi-host networking or software-defined networking.

3 Task 3. Introduction to different orchestration and contextualization frameworks

Distributed computing has become mainstream and arises today in a wide variety of domains for a wide range of applications. Especially large-scale distributed applications are becoming increasingly popular due to their ability to handle complex tasks across multiple machines, resulting in faster and more efficient processing. So here comes the question that how can we manage multiple nodes and services efficiently and conveniently. Runtime orchestration which is also known as container orchestration is used to solve this problem.
Container orchestration can be used in any environment where you use containers. It can help to deploy the same application across different environments without needing to redesign it. This microservice-based architecture

enables the application lifecycle's automation by providing a single interface for creating and orchestrating containers. With containers, applications development, testing, and production are accelerated. Container orchestration is the process of managing multiple containers in a way that ensures they all run at their best.
Contextualization involves tailoring the environment in which the application runs to suit its specific needs. This includes customizing the container environment, network configuration, and other parameters to optimize performance and ensure compatibility with the application's requirements.
Container orchestration tools provide a framework for managing containers and microservices architecture at scale. Some popular options are Kubernetes, Docker Swarm, Apache Mesos and Hasicorp Nomad. Here in this essay I will introduce these four frameworks.
Kubernetes: Kubernetes is an open source, out-of-the-box container orchestration tool. It comes with an excellent scheduler and resource manager for deploying highly available containers more efficiently. Kubernetes eliminates many of the manual processes involved in deploying and scaling containerized applications. It also assists with workload portability and load balancing by letting you move applications without redesigning them.
Docker Swarm: Docker Swarm is maturing in terms of functionalities compared to other open-source container cluster management tools. Docker Swarm is a part of Docker ecosystem, so it can integrated with Docker Engine, when using it you don't need additional orchestration software to create or manage a swarm.
Apache Mesos: Mesos can manage container orchestration very efficiently. Mesos is not a dedicated tool for containers, instead, you can use it for VM or Physical machine clustering for running workloads other than containers. It has an efficient Marathon framework for deploying and managing containers on a Mesos cluster.
Hasicorp Nomad: Nomad is an orchestration platform from Haschicorp that supports containers. It shares a similar philosophy of kubernetes in managing applications at scale. But Normad supports container and non-container workloads.
Frameworks like Kubernetes and Docker Swarm can provide some features to improve the current solution which is using docker-compose. First, Kubernetes and Docker Swarm provide built-in scaling features that make it easy to define the number of containers running in the cluster. Second, Kubernetes and Docker Swarm can detect and recover from container failures automatically. This is important for Spark clusters because the failure of a single container can affect the entire application. Third, Kubernetes and

Docker Swarm offer built-in load balancing that distributes traffic evenly across the containers in the cluster. This ensures that Spark jobs are evenly distributed across the nodes in the cluster, preventing any single node from becoming a bottleneck. Fourth, Kubernetes and Docker Swarm provide a centralized way to manage configuration data across the cluster. So it is more easier to update the Spark configuration across all the nodes in the cluster simultaneously. Last, Kubernetes and Docker Swarm allow to specify resource limits and requests for each container in the cluster. And ensures that containers have access to the resources they need to run efficiently, preventing resource contention and improving performance.