UPPSALA
UNIVERSITET

Assignment 4: Orchestration and contextualization using docker containers

Junjie Chu

Task 1. Introduction to Docker containers and DockerHub

Question:

1 - Explain the difference between contextualization and orchestration processes.

In cloud computing, contextualization means providing a customized computing environment or allows 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 and a process at the level of Platform as a Service (PaaS).

Contextualization usually focuses on user-defined-data and meta-data. But orchestration is the automated configuration, management, and coordination of computer systems, applications, and services.

We could say that, orchestration include contextualization. Contextualization is part of orchestration.

2 - Explain the followings commands and concepts:

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

FROM ubuntu:18.04: Specify the base image

RUN apt-get update

RUN apt-get -y upgrade

RUN apt-get install sl: Run 3 commands, update something(input y automatically), and install the software of 'a running train'.

ENV PATH="${PATH}:/usr/games/": This instruction sets environment variables, whether it is other instructions, such as RUN, or runtime applications, you can directly use the environment variables defined here.

CMD ["echo", "Data Engineering-I."]: When the docker image runs, show 'Data Engineering-I' in the command line.

UPPSALA
UNIVERSITET

Both CMD and RUN are used to execute commands. The difference lies in the timing of the execution of the commands. The RUN command applies to the commands executed when docker build builds the docker image, while the CMD command is used when docker run executes the docker image to build the container.

ii) Explain the command

docker run -it mycontainer/first:v1 bash

docker run: Create a new container and run a command

-t: reassign a pseudo input terminal for the container, usually used together with -i;

-i: Run the container in interactive mode, usually used together with -t;

'bash' is to run bash after loading the container, and the container will not be killed.

With this command, we can enter the container terminal for debugging.

iii) Show the output and explain the following commands:

Docker ps: List all running container information.

Docker images: View the list of local mirrors.

```
root@junjie-chu-a4:~# docker ps
CONTAINER ID   IMAGE                  COMMAND   CREATED         STATUS         PORTS     NAMES
85044324b1f8   mycontainer/first:v1   "bash"    2 minutes ago   Up 2 minutes             mystifyi
ng_snyder
root@junjie-chu-a4:~# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
mycontainer/first   v1        8b34eb563476   15 minutes ago   101MB
ubuntu              18.04     329ed837d508   12 days ago      63.3MB
root@junjie-chu-a4:~#
```

Docker stats: The docker stats command is used to display the system resources used by the container. By default, the stats command will refresh the output every 1 second.

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O
  PIDS
85044324b1f8   mystifying_snyder   0.00%     992KiB / 3.852GiB   0.02%     976B / 0B   0B / 0B
  1
```

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

docker build: Use Dockerfile to create an image.

docker run: Create a new container and run a command

UPPSALA
UNIVERSITET

docker exec: execute commands in the running container

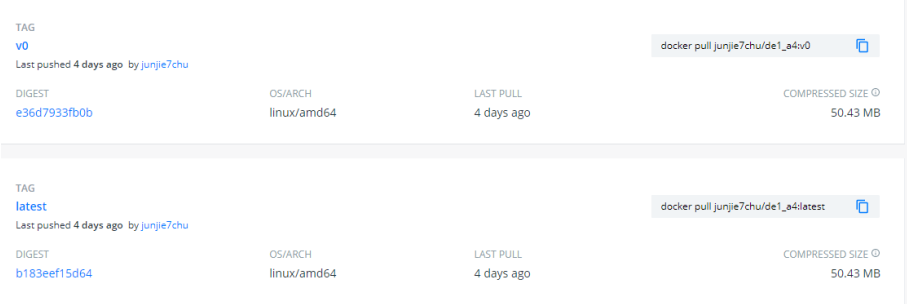
‘docker exec’ is used in a running container while ‘docker run’ is used to create a new container and run a command in the new container. We should add a command after the 2 commands. When using ‘docker build’, we create a new image(not a container) and we cannot add a command after it.

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.

Name: junjie7chu/de1_a4

Command: docker pull junjie7chu/de1_a4:v0

Picture:



DockerHub is a cloud-based repository run and managed by the Docker company. It is an online repository, and Docker images can be published and used by other users. There are two types of repositories: public repositories and private repositories. If you are a company, you can have a private repository in your own organization, and the public mirror can be used by anyone.

Once Docker is running on your system, you can open a terminal and run:

$ docker images

This command will display all docker images on the current system. Assuming you want to deploy Ubuntu on your local machine, you might:

$ docker pull ubuntu

If an Ubuntu image already exists on your system, this command will automatically update the system to the latest

version. Therefore, if you want to update an existing image, just run the docker pull command.

Now suppose you want to install a certain mirror on the system, you can run the search command to find the desired mirror:

$ docker search

As you can see, there are many different versions of images on DockerHub. Because anyone can publish a mirror, various mirrors are optimized for different projects, so you can choose the appropriate mirror. You just need to install the appropriate image for your needs.

After creating a Docker image, we can easily publish the image to DockerHub. First, you need to log in to DockerHub. If you don't have an account, please create an account. Then, you can open the terminal application and log in with your own Docker Hub user name (mine is junjie7chu).

$ dockerlogin--username=junjie7chu

Enter the password and you are logged in. Now run the docker images command to get the ID of the image you created last time.

$ docker images

Now, suppose you want to push the image ng to DockerHub. First, we need to mark the image:

$ docker tag

Then push the image:

$ docker push

The push points to the image repository you specify:

Once the push is complete, open DockerHub, log in to your account, and you will be able to see your first Docker image. Now anyone can deploy your image. This is the easiest and fastest way to develop and release software. Whenever you update the image, users can simply run:

Docker containers solve many of the problems faced by traditional workloads and allow you to develop, test, and deploy applications at any time.

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

’docker-compose’ is to orchestrate containers. By using docker-

compose, you can write a series of operations such as the startup parameters of each container, environment variables, container naming, specifying the link parameters of different containers, etc. in the docker-compose.yml file at one time. When starting this whole environment (for example, with 3 containers), you only need to type a docker-compose up command to complete it.

The role of ‘docker file’ is to build an image. It contains the environment, program code, etc. required for installation and operation. This creation process is done using ‘docker file’. The ‘docker file’ is prepared for the ‘docker build’ command and is used to build an independent image. It can also be used to build the image in real time in docker-compose.

So docker-compose create several containers based on some images while docker build create a image.

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

Questions

1 - Explain your docker-compose configuration file.

```yaml
version: "3"
services:
  master:
    image: sparkaio/first:v0
    hostname: master
    restart: always
    ports:
      - "4040:4040"
      - "7077:7077"
      - "8080:8080"
    command: bash /usr/local/spark/sbin/start-master.sh
  worker1:
    image: sparkaio/first:v0
    restart: always
    depends_on:
      - master
    ports:
      - "8081:8081"
    command: bash /usr/local/spark/sbin/start-slave.sh spark://master:7077
```

version: '3'

Indicates that the Docker-Compose file uses Version 3 file.

services:

master:

image: sparkaio/first:v0

UPPSALA
UNIVERSITET

The second level label under the services label is ‘master’, this name is customized by the user, and it is the service name. Another service name is ‘worker1’.

‘image’ is the image name or image ID of the specified service. If the mirror does not exist locally, Compose will try to pull the image.

hostname: master

restart: always

ports:

- "4040:4040"

- "7077:7077"

- "8080:8080"

‘hostname’ specifies the hostname of the service, which is used to specify the spark master name.

‘restart’ specifies that when the service meet a problem, it automatically restarts.

‘ports’ defines the mapping between host computer’s port and container’s port. So that the container’s ports could be accessed using host IP+host port. I specify the ports which will be used in Spark here for the future debugging but actually they are not very important in the case.

depends_on:

- master

‘depend_on’ specifies the order of service loading, for example, the worker service needs to be performed after the master service.

command: bash /usr/local/spark/sbin/start-slave.sh spark://master:7077

‘command’: The command executed by default after the container is started. In the case, I want the spark’s master process and spark’s worker process to run by default. So I use 2 commands in the .yml file.

Check if the services really work (running pi example):

UPPSALA
UNIVERSITET

```
root@junjie-chu-a4:~/dockercompose# docker-compose ps
         Name                       Command               State                                                Ports
-------------------------------------------------------------------------------------------------------------------------------------------------
dockercompose_master_1    bash /usr/local/spark/sbin ...   Up      0.0.0.0:4040->4040/tcp, 0.0.0.0:7077->7077/tcp, 0.0.0.0:8042->8042/tcp,
                                                                   0.0.0.0:8080->8080/tcp, 0.0.0.0:8088->8088/tcp
dockercompose_worker1_1   bash /usr/local/spark/sbin ...   Up      8081/tcp
root@junjie-chu-a4:~/dockercompose# docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED         STATUS         PORTS
                                                    NAMES
d9369095c990   sparkaio/first:v0   "bash /usr/local/spa…"   5 minutes ago   Up 5 minutes   8081/tcp
                                                    dockercompose_worker1_1
fb3c8966bcd9   sparkaio/first:v0   "bash /usr/local/spa…"   5 minutes ago   Up 5 minutes   0.0.0.0:4040->4040/tcp, 0.0.0.0:7077->7077/tcp, 0.0.0.0:8042->8042/
tcp, 0.0.0.0:8080->8080/tcp, 0.0.0.0:8088->8088/tcp   dockercompose_master_1
```

```
root@junjie-chu-a4:~/dockercompose# docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED         STATUS         PORTS
      NAMES
29285fb70f7f   sparkaio/first:v0   "bash /usr/local/spa…"   3 minutes ago   Up 3 minutes   0.0.0.0:8081->8081/tcp
      dockercompose_worker1_1
37a506e5cb6e   sparkaio/first:v0   "bash /usr/local/spa…"   3 minutes ago   Up 3 minutes   0.0.0.0:4040->4040/tcp, 0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/
tcp   dockercompose_master_1
```

```
root@junjie-chu-a4: ~/dockercompose                                    root@master: /
CONTAINER ID   NAME                        CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
29285fb70f7f   dockercompose_worker1_1     48.40%    204.3MiB / 3.852GiB   5.18%     6.87MB / 6.24MB   0B / 8.55MB   42
37a506e5cb6e   dockercompose_master_1      50.24%    273.3MiB / 3.852GiB   6.93%     6.24MB / 6.87MB   0B / 414kB    75
```

```
21/03/17 16:51:51 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
21/03/17 16:51:51 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 5.309832 s
Pi is roughly 3.146955734778674
21/03/17 16:51:51 INFO SparkUI: Stopped Spark web UI at http://master:4040
21/03/17 16:51:51 INFO StandaloneSchedulerBackend: Shutting down all executors
21/03/17 16:51:51 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
21/03/17 16:51:51 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

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

The Compose file is a YAML file defining services, networks and volumes. The default path for a Compose file is ./docker-compose.yml. Either a .yml or .yaml extension could be used for this file.

A service definition contains configuration that is applied to each container started for that service, much like passing command-line parameters to docker run. Likewise, network and volume definitions are analogous to docker network create and docker volume create. We could also specify a docker file to build an image.

3 - What are the limitations of docker-compose?

Using docker-compose is fine if we are working on a single machine or don't need to distribute containers across multiple inter-connected machines. Because there is no load balance, tasks are unevenly distributed. At the same time, the fault tolerance is not high, and when there is a problem or every deployment, we will experience a few seconds of downtime.

And when the "production" environment needs to be distributed across multiple machines, and that containers should be distributed across those machines dynamically, docker-compose is not suitable. If we only use docker-compose, we have to run it many times on many hosts. We should look into using Docker Swarm or Kubernetes for deploying and scaling to multiple hosts.

UPPSALA
UNIVERSITET

Task 3. Introduction to different orchestration and contextualization frameworks (1 point)

Orchestration is the automatic configuration, management and coordination of systems, applications and services. We need it to help us combine multiple tasks and configurations among multiple hosts.

Part 1: Orchestration frameworks

Ansible is an automatic orchestration tool. Based on Paramiko, it realizes the functions of batch system configuration, batch program deployment, batch operation command, etc. It can be roughly divided into three layers: user, master control end and controlled terminal. Ansible is based on modular work and provides a framework. The module that ansible runs has batch deployment instead of Ansible.

Features of Ansible: It does not need to install any client on the controlled host; it has no server and can run commands directly; it can develop modules in any language based on module work; it can customize playbook with yaml language; it can work based on SSH; it can realize multi-level command(strong multi-tier solution).

Terraform is an open source multi cloud resource orchestration tool. Users use a specific configuration language to describe infrastructure. Terraform tool uniformly analyzes it, constructs the relationship between resources, generates the execution plan, and calls the specific implementation of each cloud vendor to complete the management of the whole infrastructure life cycle.

Features of Terraform: The infrastructure can be described in a domain specific language based on the design of infrastructure as code, which eliminates the ambiguity in the description semantics of infrastructure automation and reduces the impact of human factors. Terraform will generate a readable implementation plan, and the changes of key infrastructure can be fully reviewed before the orchestration. The relationship between resources is described based on directed acyclic graph. When the relationship between resource attributes and resources changes, the change action will be fully executed in parallel.

Kubernetes is an open source container orchestration engine of Google. When an application is deployed in a production environment, multiple instances of the application are usually deployed to balance the load of application requests. We can create multiple containers and run an application instance in each container. Then, through the built-in load balancing strategy, we can manage and access this group of application instances easily.

Features of Kubernetes: Kubernetes has an automation mechanism. We can realize automatic expansion, automatic update, automatic deployment, automatic management of resources and so on through

it. Kubernetes is service-oriented, which allows us to put aside the system environment and operation details, and focus on logic business. The system built on it can run independently in physical machine, virtual machine, private cloud and public cloud. Kubernetes will check the application instances regularly. If a new application instance started, the load balancing will automatically start. If Kubernetes finds an instance's state is wrong, it will automatically kill the problem instance and reschedule a new one. Kubernetes can make the whole cluster rolling update smoothly so that we can update the application without stopping external services.

Swarm is a cluster management tool officially provided by docker. Its main function is to abstract several docker hosts as a whole, and manage all kinds of docker resources on these docker hosts through a single entrance. Swarm is similar to Kubernetes, but it is lighter and has fewer functions than Kubernetes.

Features of Swarm: It is highly integrated with docker engine and does not need additional plug-ins. There are two kinds of nodes in swarm cluster: manager and worker. Manager manages the cluster and can also provide external services, while worker provides external services. There can be multiple managers in a cluster, any worker can be promoted to manager, and any manager can be demoted to worker. It also has the characteristics of multi host network, load balancing, certificate security, rolling upgrade and so on.

Part 2: Kubernetes and Docker Swarm vs Docker Compose

Docker compose can only manage dockers on the current host, that is, it cannot start docker containers on other hosts. If we want to run spark clusters on several hosts, we need to run docker-compose many times. But docker swarm (or Kubernetes) is a tool for managing docker containers on multiple hosts. They could help us manage the cluster running on many hosts easily.

If you use docker-compose and your app is only running on a single server, the chances of downtime due to an affected DC, broken containers or network link gets significantly increased. If you docker-compose and your app is running across multiple servers, it is very hard to balance the work load between the servers.

Kubernetes and Docker Swarm could help us solve the problems. Docker Swarm (or Kubernetes) is a tool used to manage docker containers on multiple hosts. It can help you start the container and monitor the state of the container. If the state of the container is abnormal, it will help you restart a new container to provide services. This improve the fault tolerance rate. At the same time, it also provides load balancing between services. If your app is running across multiple servers and multiple data centers then chances of downtime due to an affected DC or network link gets significantly reduced by using these 2 tools.