102. Configure applications

Configuring applications comprises of understanding the following concepts

* Configuring Command and Arguments on applications
* Configuring Environment Variables
* Configuring secrets

Seriya
See puriyala.

103. Commands "It is related to docker

This is related to "commands & arguments in a pod-definition-file"

Commands and Containers in docker (this will be converted to pods in next lecture)

~~Topics~~ Commands we're going to look at

Topics that we are going to look at

⇒ Commands ⇒ Arguments ⇒ Entry points in docker

Scenario:-

You were to run a Docker Container from an ubuntu image,

```
>docker run ubuntu
```

when you run the above command, it runs as instance of ubuntu image & exists immediately.

⇒ If you list the running Containers, you wouldn't see the Containers running.

```
docker ps
```

CONTAINERID IMAGE COMMAND CREATED STATUS PORTS

If you list all the containers including those that are stopped, you'll see that the new containers you ran is in an exited state.

```
docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
<some random #>	ubuntu	"/bin/bash"	43 seconds ago	Exited (0) 41 seconds ago	

What's the reason?

⇒ Unlike virtual machines, containers are not meant to host an operating system

⇒ Containers meant to run a specific task (ex) process such as to host an instance of a webserver, application server or database (or) simply to carryout some computations or analysis

⇒ Once the task is complete, the container exits

⇒ A container lives as long as the process inside it is alive.

⇒ If the webservice inside the container is stopped or crashes, the container exits,

So, who defines what process is run within the Container

If you look at the Docker file for popular Docker Images, like NGINX, you'll see an instruction called CMD, stands for Command, that defines the program that will be run within the container when it starts

For the NGINX image, it is the NGINX command

For the MYSQL image, it is the MYSQL D command

Install Nginx.

```
RUN \
    add-apt-repository -y ppa:nginx/stable && \
    apt-get update && \
    apt-get install -y nginx && \
    rm -rf /var/lib/apt/lists/* && \
    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
    chown -R www-data:www-data /var/lib/nginx
# Define mountable directories.
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/log/nginx", "/var/www/html"]

# Define working directory.
WORKDIR /etc/nginx

# Define default command.
CMD ["nginx"]
```

2:05

what we tried to do earlier was to run a container with a plain Ubuntu operating system

```
#Pull base image
FROM ubuntu:14.04

#Install
RUN \
  sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \
  apt-get update && \
  apt-get -y upgrade && \
  apt-get install -y build-essential && \
  apt-get install -y software-properties-common && \
  apt-get install -y byobu curl git htop man unzip vim \
                     wget && \

#Add files
ADD root/.bashrc /root/.bashrc
ADD root/.gitconfig /root/.gitconfig
ADD root/.scripts /root/.scripts

#Set environment variables
ENV HOME /root

#Define working directory
WORKDIR /root

#Define default command
CMD ["bash"]
```

Here, we can see that the default command used is

bash

=> bash is not really a process like a webserver (or) database server, bash is a shell that listens for inputs from a Terminal, if it cannot find a Terminal, it exits

when we run the ubuntu container earlier Docker created a container from the ubuntu image & launched the bash program

⇒ By default, docker doesn't attach a terminal to a container when it is run, so the bash program doesn't find the terminal & so it exits.

Since the process that was started when the container was created finished, the container exits as well

So have to specify a different command to start the container

option 1:-

To append a command to the Docker run command & that way, it overrides the default command specified within the image

Syntax:-

docker run ubuntu [COMMAND]

command:-

docker run ubuntu sleep 5

Here, we've run docker run command with "sleep 5" command as the added option.

⇒ This way when the container starts, it runs the sleep program, wait for 5 seconds & then exits

So how do you make that change permanent

⇒ You want the image to always run the sleep command when it starts

⇒ You would then create your own image from the base ubuntu image & specify a new command

```
FROM Ubuntu
CMD sleep 5
```

⇒ There are different ways of specifying the command

3:47 * Either the command simply as in a Shell form:

```
FROM Ubuntu
CMD sleep 5
```

(or)

JSON array format, like below

```
CMD sleep 5
CMD ["sleep", "5"]
```

Remember:-

⇒ When you specify in a JSON array format, the first array in the element, should be executable.

⇒ In this case, the sleep program

⇒ Should not specify the "command" & "parameters" together

CMD ["sleep 5"] ⇒ This is wrong

⇒ The Commands & parameters should be separate elements in the list

Now, I build my new image using the Docker build command & name it as "ubuntu sleeper"

> Docker build -t ubuntu-sleeper

4:22

Now, I could simply run the docker ubuntu sleeper command & get the same result,

It always sleeps for 5secs & exists

But what if I wish to change the no of seconds, it sleeps

⇒ Currently, it is hard coded to 5 seconds

docker run ubuntu-sleeper sleep 10

As learned before, one option is to run the Docker run command with the new command appended to it [here, it is "sleep 10"]

And so the command that will be run at startup will be "sleep 10"

command at Startup: sleep 10

⇒ The name of the image 'ubuntu sleeper' itself implies that the container will sleep. So we shouldn't have to specify the sleep command again

⇒ Instead, we would like it to be something like this

docker run ubuntu-sleeper 10

we only want to pass in the no. of seconds the container should sleep & the sleep command should be invoked automatically

This is where the "entry point" instruction comes into play

FROM Ubuntu

ENTRYPOINT ["sleep"]

⇒ Entry point instruction is like the command instruction as in, you can specify the program that will be run when the container starts, & whatever you specify on the

Command line, in this case 10, will get appended to the entry point

```
FROM        Ubuntu
ENTRYPOINT  ["sleep"]
```

```
docker run ubuntu-sleeper 10
```

```
Command at startup : sleep 10
```

So the command that will be run when the Container starts is sleep 10

This is the difference b/w the two

⇒Incase of CMD instructions, the command line parameters passed will get replaced entirely

⇒Incase of Entry points, the command line parameters will get appended

In the 2nd case, what if we run the ubuntu sleeper image command without appending the number of seconds,

```
docker run ubuntu-sleeper
sleep: missing operand
Try 'sleep --help' for more information
```

Command at Startup : sleep

Then the command at startup will be just sleep & you got the error that the operand is missing

How to configure a default value for the command if one was not specified in the command line

This is where you use both "entry point" as well as the "command" instruction

6:27

```
FROM Ubuntu
ENTRYPOINT ["sleep"]
CMD ["5"]
```

Command at Startup: sleep 5

```
docker run ubuntu-sleeper 10
```

Command at Startup: sleep 10

In this case, command instruction will be appended to the entry point instruction,

So at startup, the command would be sleep 5

⇒ If you didn't specify any parameters in command line, then it will override the command instruction

Remember:-

For this to happen, you should always specify the entry point & command instructions in a JSON format

Finally:-

What if you really want to modify the entrypoint during a runtime

For eg:- From sleep to imaginary sleep 2.0 command In this case, you can override it by using the entry point option in the Docker run command

The final command in the startup would then be

sleep 2.0 10

```
docker run --entrypoint sleep2.0 ubuntu-sleeper 10
```

```
Command at Startup: sleep2.0 10
```