syllabus

1. Introduction to DevOps,
2. Microservices,
3. Fragmentation of business requirement,
4. Containerisation,
5. docker,
6. Container life cycle,
7. YAML,
8. Docker Swarm and Docker Stack ,
9. Kubenetes,
10. Istio Service Mesh,
11. delivery pipeline,
12. Jenkins,
13. Selenium integration with Jenkins,Developing an application in a team,
14. code versioning system,

notes

1. deisadvantage of monolithic services

- 3 dis
- to avoid this use

2. Microservices

3. What is REST API and why it is used?

- A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.

day 4 : 29/11/2020

Containerization using Docker

docker command

0. check docker version

```
sudo docker version
```

1. command to start docker

```
sudo systemctl start docker
```

2. coomand to stop docker

```
sudo systemctl stop docker
```

3. command to restart docker

> sudo systemctl restart docker

4. change to root user

> su - password : Enter password

5. docker image

- 1. inspect docker image

> sudo docker image inspect

```
sudo docker image inspect jenkinsci/blueocean
```

- 2. check dpcker image list

> sudo docker image ls

- 3. check image history

> sudo docker history <repository/image>

```
sudo docker history hello-world
```

- 4. get OS image from docker library/hub

> sudo docker image pull <repo/image>

```
sudo docker image pull ubuntu
```

5. Container command

- 1. to create container out of docker image
  - it creates and runs the image ,
  - it get closed as no nothing to run on it

> sudo docker container run ubuntu

- 2. list of working and stopped container

> sudo docker container ls -a

- 3. list of working container

> sudo docker container ls

Note :

- docker creates Virtual machine / or sharing same OS on physical machine if same OS required for container
- if not , diff OS, a OS VM is created
- VM runs full OS
- for container it runs kernel :
    - its starts , if there are commands/process to run it does it , or container get exited.
- 4. to delete a container

```
sudo docker container rm ContainerID
```

- 5. to start the docker , with its Terminal in Container OS image

```
sudo docker container run -it ubuntu
```

- where -i : interactive, t : tele type(Terminal)

```
$ sudo docker container run -it  ubuntu
root@d5809cf98cc0:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@d5809cf98cc0:/# mkdir code
root@d5809cf98cc0:/# cd code/
root@d5809cf98cc0:/code# touch file1.txt
root@d5809cf98cc0:/code# touch file2.txt
root@d5809cf98cc0:/code# ls
file1.txt  file2.txt
root@d5809cf98cc0:/code# exit
exit
```

- 5. install docker image

```
sudo docker image pull mysql
```

6. use httpd image of apache instead of below commmad to install apache2 on container

```
sudo docker container run -it ubuntu
root@b33888469f38:/#
1. apt-get update

2. apt-get install apache2

3. service apache2 start

4. apt-get install curl
```

```
5. curl http://localhost
```

- 1. instead use

```
sudo docker image pull httpd
```

- 2. now to run httpd

```
sudo docker container run -it httpd
```

- 3. inspect httpd

```
sudo docker image inspect httpd
```

- 4. check network info

```
ifconfig
```

- 172.17.0.1 : ip address
- 5.
- 6. to run server in detached mode , in background using command -d (demion)

```
docker container run -d httpd
```

- 7. running with port forwarding

```
sudo docker container run -d -p 5678:80 httpd
```

- 8. to stop -d contianer

```
- sudo docker container ls

CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
2bd496dbce31        httpd               "httpd-foreground"   16 seconds ago
Up 13 seconds       80/tcp              upbeat_ardinghelli

- sunbeam@sunbeam-Inspiron-3583:~$ sudo docker
container rm 2bd

Error response from daemon: You cannot remove a running container
2bd496dbce31428c52bf5cc37f8fc32f3dbb435456cf421901e7c7d6525cb212. Stop the
container before attempting removal or force remove

- sunbeam@sunbeam-Inspiron-3583:~$ sudo docker container stop 2bd
2bd

- sunbeam@sunbeam-Inspiron-3583:~$ sudo docker container rm 2bd
2bd
```

7. connect/start apacheserver in container using httpd

- 1. check ip address/ network info

> ifconfig

- check ip

- 2. for port forwarding from pc to port 60 of container http5

> $sudo docker container run -d -p 5678:80 httpd

- 3. to execute any command inside container

> sudo docker container exec -it (containerID) bash

```
sudo docker container exec -it 1d712ffb3df6 bash

root@1d712ffb3df6:/usr/local/apache2# ls
bin    cgi-bin  error   icons    logs
build  conf htdocs  include  modules

root@1d712ffb3df6:/usr/local/apache2# cd htdocs

root@1d712ffb3df6:/usr/local/apache2/htdocs# ls
index.html

root@1d712ffb3df6:/usr/local/apache2/htdocs# cat index.html
<html><body><h1>It works!</h1></body></html>

root@1d712ffb3df6:/usr/local/apache2/htdocs# echo "hello world"
hello world

root@1d712ffb3df6:/usr/local/apache2/htdocs# apt-get install vim

root@1d712ffb3df6:/usr/local/apache2/htdocs# vim index.html
root@1d712ffb3df6:/usr/local/apache2/htdocs# vim index.html
root@1d712ffb3df6:/usr/local/apache2/htdocs# exit
exit
```

- 4. to see content of html file

> cat ____.html

- 5. vim commands
-
- 1. open html file

> vim ___.html

- 2. to delete a line

```
dd
```

- 3. to insert in file

```
Esc + i
```

- 4. to exit

```
Esc + :wq
```

- 5. now open browser to open your webpage on port 5678 coming from container port 80

```
http://localhost:5678/ http://172.17.0.1:5678
```

custom docker image creation 7/12/2020

docker hub log in

1. docker id: surajporje
2. password : 8668951369

-

how to create a Container/Instance of a standard image

1. to manually create DB

- 1. after downloading mysql image
  - create mysql container , with schema using
  - need environment variable
  - create port : mandetory

```
# code to create a container from standard image
sudo docker container run --name mydb -d -e MYSQL_ROOT_PASSWORD=root -e
MYSQL_DATABASE=mydb -p 9090:3306 mysql:5.7
```

for standard approach for image creation

0. mandetory to create Dockerfile

- docker file commands :
- 1. FROM : => indicates base image, from which we will create a custom image
- 2. ENV : => used to set environment variable , like mysql_root_password
- 3. COPY : => used to copy file from local machine to container specific folder
- 4. ADD : => used to copy file from local machine to container specific folder, and also download content from url and unzip file

- 5. EXPOSE => used to expose port 3306 from container for port forwarding , outside the container
- 6. to COPY everything from current directory on machine to home directory of container

> COPY . .

- 7. run a command when container starts

> CMD

- e.g:

> CMD ["node","server.js"] CMD node server.js

1. for our db image

- Dockerfile

```
# base image(mysql) used to create my image
FROM mysql:5.7

# ENV : used to set environment variable , like mysql_root_password
# set root password to root
ENV mysql_root_password "root"

# create a database named ecommercedb
ENV MYSQL_DATABASE "ecommercedb"

# copy db.sql file to container i.e folder /docker-entrypoint-initdb.d for
docker to run it
COPY ./db.sql /docker-entrypoint-initdb.d

# EXPOSE port 3306 from container for port forwarding
EXPOSE 3306
```

- sql file

```
create table user(id integer primary key auto_increment,firstname
varchar(40),lastname varchar(40),email varchar(50),password varchar(100));
create table product (id integer primary key auto_increment,title
varchar(100),price float);
```

- code on terminal

```
sudo docker image build . -t myappdb2
```

- where ,
- . => indicates current directory, where we can find both Docker file and sql file
- -t => t for tag : depicts the building image name
- so now to run our image i.e Container commands

```
$ : sudo docker container run -d -p 9090:3306 --name mydb myappdb
# here, myappdb : image name , mydb : container name , -d : detached , -p:
port mapping
# verify using ls command
$ : sudo docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                                NAMES
c121fb10f354        myappdb             "docker-entrypoint.s…"   49 seconds
ago      Up 46 seconds       33060/tcp, 0.0.0.0:9090->3306/tcp   mydb

# now open mysql
sudo mysql -u root --port 9090 --protocol tcp -p

# successfull, now can perform db operation here
```

2. create image for backend : node /express

- docker file

```
# create the backend image with node as base image
FROM node

# COPY everything from current directory on machine to home directory of
container
COPY . .

# expose port 3000
EXPOSE 3000

# run a command when container starts

CMD ["node","server.js"]
```

- code on terminal

```
# build iamge
sudo docker image build . -t mybackend
# check using ls
$ sudo docker image ls
# create a container
sudo docker container run -d -p 4000:4000 --name run mybackend
```

4. create index.html on container

- create Dockerfile

```
# use http as base image
FROM httpd

# copy index.html to the htdocs
COPY ./index.html /usr/local/apache2/htdocs/

# expose port 80
EXPOSE 80

# run apache in foreground(continuously )
CMD apachectl -D FOREGROUND
```

- create index.html
- commands on terminal

```
# build docker custom image
$ : sudo docker image build . -t myweb

# commands In Docker file

# Step 1/4 : FROM httpd

# Step 2/4 : COPY ./index.html /usr/local/apache2/htdocs/

# Step 3/4 : EXPOSE 80

#Step 4/4 : CMD apachectl -D FOREGROUND



$  sudo docker container run -d -p 3000:80 --name myweb myweb
```

5. to run angular application

- docker file commands

```
FROM httpd

COPY ./dist/mywebsite/ /usr/local/apache2/htdocs/

EXPOSE 80
```

```
CMD apachectl -D FOREGROUND
```

- terminal commands

```
$ : ng new mywebsite
# create application pages , then
$ :  ng serve --open

$ :~/dac/DevOps/docker-project/angular/mywebsite$ ng build --prod

# on ng build production level angualar copy is created , contains all
js,css files ,all ts files get compiled to js

$ :~/dac/DevOps/docker-project/angular/mywebsite/dist/mywebsite$ ls
3rdpartylicenses.txt  index.html
polyfills.35a5ca1855eb057f016a.js  styles.3ff695c00d717f2d2a11.css
favicon.ico           main.e225af7c98748907695e.js
runtime.acf0dec4155e77772545.js



# build custom image
$:~/dac/DevOps/docker-project/angular/mywebsite$ sudo docker image build .
-t angularwebsite

# check if image created
$: sudo docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED
SIZE
angularwebsite        latest              141617bee912        11 seconds
ago      138MB
# create container of custom image
dac/DevOps/docker-project/angular/mywebsite$ sudo docker container run -d -
p 8080:80 angularwebsite

# now open browser to check website on
> localhost:8080/
```

command to push/pull the images to/from docker hub

- 1. tag image name on machine as DockerId/image to push on docker hub

> sudo docker image tag dockerId/image-name

- 2. to push the image

> sudo docker image push dockerId/image-name

- 3. to pull the image stored on docker hub

- sudo docker image pull dockerId/image-name

```
# login to docker
$  sudo Docker login
# tag image name on machine as DockerId/image to push on docker hub
 sudo docker image tag myappdb surajporje/myappdb

 # to push the image
 sudo docker image push surajporje/myappdb

 # now we can delete images on our image ,
 # now we can be download/pull from docker hub
 sudo docker image pull surajporje/myappdb
```

Question

1. What is Docker?

- Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment be it development or test or production.
- Docker containers, wrap a piece of software in a complete filesystem that contains everything needed to run:
- code, runtime, system tools, system libraries etc. anything that can be installed on a server.
- This guarantees that the software will always run the same, regardless of its environment.

2. What is the need for DevOps?

- Nowadays instead of releasing big sets of features, companies are trying to see if small features can be transported to their customers through a series of release trains. This has many advantages like quick feedback from customers, better quality of software etc. which in turn leads to high customer satisfaction. To achieve this, companies are required to:
    - Increase deployment frequency
    - Lower failure rate of new releases
    - Shortened lead time between fixes
    - Faster mean time to recovery in the event of new release crashing
    - DevOps fulfills all these requirements and helps in achieving seamless software delivery.

3. How to build envrionment-agnostic systems with Docker?

- There are three main features helping to achieve that: + Volumes + Environment variable injection + Read-only file systems

4. Is there a way to identify the status of a Docker container?

- We can identify the status of a Docker container by running the command

```
docker ps –a
```

- which will in turn list down all the available docker containers with its corresponding statuses on the host.
- From there we can easily identify the container of interest to check its status correspondingly.

5. What are the advantages of DevOps?

- Technical benefits:
    - Continuous software delivery
    - Less complex problems to fix
    - Faster resolution of problems
- Business benefits:
    - Faster delivery of features
    - More stable operating environments
    - More time available to add value (rather than fix/maintain)

6. What are the most common instructions in Dockerfile?

- Some of the common instructions in Dockerfile are as follows:
- 1. FROM: We use FROM to set the base image for subsequent instructions. In every valid Dockerfile, FROM is the first instruction.
- 2. LABEL: We use LABEL to organize our images as per project, module, licensing etc. We can also use LABEL to help in automation. In LABEL we specify a key value pair that can be later used for programmatically handling the Dockerfile.
- 3. RUN: We use RUN command to execute any instructions in a new layer on top of the current image. With each RUN command we add something on top of the image and use it in subsequent steps in Dockerfile.
- 4. CMD: We use CMD command to provide default values of an executing container. In a Dockerfile, if we include multiple CMD commands, then only the last instruction is used.

7. What are the various states that a Docker container can be in at any given point in time?

- There are four states that a Docker container can be in, at any given point in time. Those states are as given as follows: + Running + Paused + Restarting + Exited

8. What is Docker container?

- Docker containers include the application and all of its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system.
- Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

9. What is Docker hub?

- Docker hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

10. What is Docker image?

- Docker image is the source of Docker container. In other words, Docker images are used to create containers. Images are created with the build command, and they'll produce a container when started with run.
- Images are stored in a Docker registry such as registry.hub.docker.com because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network.

11. What is the difference between the COPY and ADD commands in a Dockerfile?

- Although ADD and COPY are functionally similar, generally speaking, COPY is preferred.
- That's because it's more transparent than ADD. COPY only supports the basic copying of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious. Consequently, the best use for ADD is local tar file auto-extraction into the image, as in ADD rootfs.tar.xz /.

12. What is the function of CI (Continuous Integration) server?

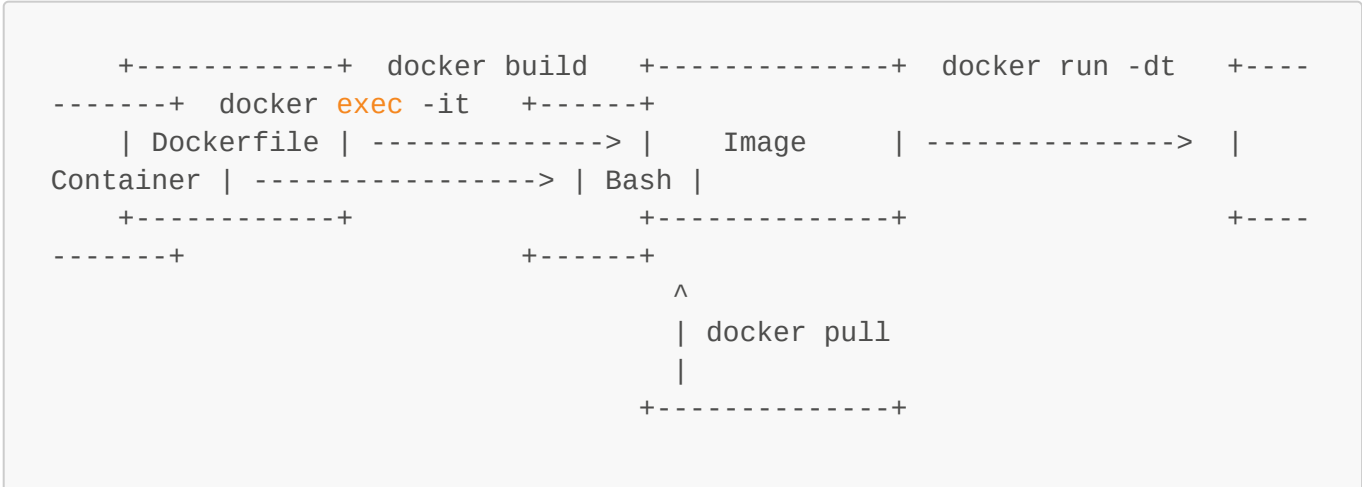
- CI server function is to continuously integrate all changes being made and committed to repository by different developers and check for compile errors. It needs to build code several times a day, preferably after every commit so it can detect which commit made the breakage if the breakage happens.

13. What type of applications - Stateless or Stateful are more suitable for Docker Container?

- It is preferable to create Stateless application for Docker Container. We can create a container out of our application and take out the configurable state parameters from application. Now we can run same container in Production as well as QA environments with different parameters. This helps in reusing the same Image in different scenarios. Also a stateless application is much easier to scale with Docker Containers than a stateful application.

14. Explain basic Docker usage workflow

- Everything starts with the Dockerfile. The Dockerfile is the source code of the Image. Once the Dockerfile is created, you build it to create the image of the container. The image is just the "compiled version" of the "source code" which is the Dockerfile. Once you have the image of the container, you should redistribute it using the registry. The registry is like a git repository -- you can push and pull images. Next, you can use the image to run containers. A running container is very similar, in many aspects, to a virtual machine (but without the hypervisor).



```
                                    |   Registry   |
                                    +--------------+
```

15. How will you monitor Docker in production?

- Docker provides tools like docker stats and docker events to monitor Docker in production. We can get reports on important statistics with these commands.

Docker stats: When we call docker stats with a container id, we get the CPU, memory usage etc of a container. It is similar to top command in Linux. Docker events: Docker events are a command to see the stream of activities that are going on in Docker daemon.

Some of the common Docker events are: attach, commit, die, detach, rename, destroy etc. We can also use various options to limit or filter the events that we are interested in.

16. What is Docker Swarm?

- Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

17. What is Hypervisor?

- The hypervisor handles creating the virtual environment on which the guest virtual machines operate. It supervises the guest systems and makes sure that resources are allocated to the guests as necessary. The hypervisor sits in between the physical machine and virtual machines and provides virtualization services to the virtual machines. To realize it, it intercepts the guest operating system operations on the virtual machines and emulates the operation on the host machine's operating system.

- The rapid development of virtualization technologies, primarily in cloud, has driven the use of virtualization further by allowing multiple virtual servers to be created on a single physical server with the help of hypervisors, such as Xen, VMware Player, KVM, etc., and incorporation of hardware support in commodity processors, such as Intel VT and AMD-V.

18. What is the difference between Docker Image and Layer?

- Image: A Docker image is built up from a series of read-only layers Layer: Each layer represents an instruction in the image’s Dockerfile. The below Dockerfile contains four commands, each of which creates a layer.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

- Importantly, each layer is only a set of differences from the layer before it.

19. What is virtualisation?

- In its conceived form, virtualisation was considered a method of logically dividing mainframes to allow multiple applications to run simultaneously. However, the scenario drastically changed when companies and open source communities were able to provide a method of handling the privileged instructions in one way or another and allow for multiple operating systems to be run simultaneously on a single x86 based system.
- The net effect is that virtualization allows you to run two completely different OS on same hardware. Each guest OS goes through all the process of bootstrapping, loading kernel etc. You can have very tight security, for example, guest OS can't get full access to host OS or other guests and mess things up.
- The virtualization method can be categorized based on how it mimics hardware to a guest operating system and emulates guest operating environment. Primarily, there are three types of virtualization:
    1. Emulation
    2. Paravirtualization
    3. Container-based virtualization