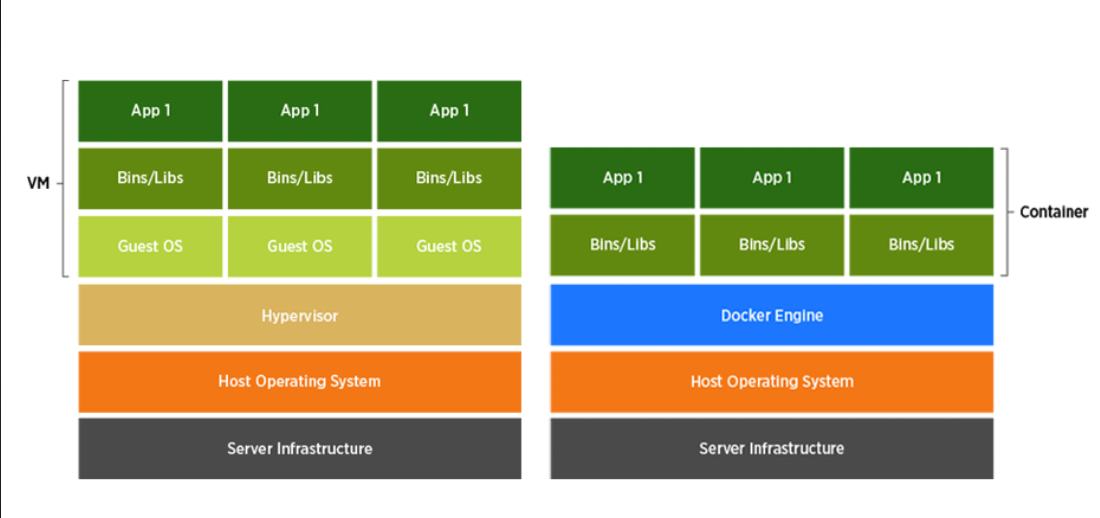
Using container approach, we effectively use resources

We can scale

Deployment is easy and fast, c0z container itself contains app + all its dependencies

Comparison b/w Vm & docker



**Why Docker ?**

* Faster time to market
* Developer productivity
* Deployment velocity
* IT infrastructure reduction
* IT operational efficiency -
* Faster issue resolution

**IN IT Market most of the work loads are getting migrated to Docker so docker is a must skill for devops guy.**

**Steps to create docker containers :**

Lets dockerize sample python project :

<https://github.com/javahometech/python-standalone-docker>

Install pythin

Git clone

Cd in to particular path

Pip install flask – I am installing my dependencies on my host

Pip install redis –

We want to write docker file

Alpine image – it is light weight docker image which is around 5 mb

From alpine:3.11

# Install python3

When you build docker images always prefer light weight base images

Prefer official images

From alpine:3.11

# Install python3

RUN apk add python3

# Install flask

RUN pip install flask

# Install redis

RUN install redis

# copy code to ypur docker image

WORKDIR /app

COPY app.py .

# open 5000 port

Expose 5000

# start python app at docker run time

CMD python app.py

**Why containers?**

* Docker containers are light weight they consume less compute resources.
* There is no guest OS in case of containers.
* Docker Containers are portable. (i.e.) – container will have application and all its dependencies such that take a container put it on any server it works without any dependency issues.
* Container creation time is fast. (i.e) containers are created & started in seconds (i.e creation and startup time is less than 2 seconds.)
* **Deployment is easy & fast.**

**Process of creating containers:**

* Step1: Write a Docker file

In the same file docker file contains instructions which tells what should be part of our Docker image

* Step2: Using Docker file build docker image
* Step3: Using Docker image create Docker containers. (we can create 1000’s of docker containers)

**What is docker?**

Docker is containerization tool for running applications, Today docker is widely used for running apps, and lots of migrations from VM to containers is happening in the industry

**Why we shoud use docker?**

* Docker containers are light weight i.e. they consume fewer resources, docker containers stats in 1-2 seconds
* Docker is higly portable
* Docker apps will scale better than VM based apps
* Deploying docker apps is easy and fast (most important for micro services based applications)

**The process of dockerizing application**

Write Dockerfile --> Build Docker Image --> Create Docker Container

**Run Docker Container on VM**

* take virtual machine (EC2 from AWS)
* Install docker daemon on EC2
* sudo yum install docker -y
* sudo chkconfig docker on
* sudo service docker start

**Run tomcat in docker container**

sudo docker run -d --rm -p 8080:8080 tomcat:8.0

-d stands for detach mode, you container runs in detach mode -it stands for interactive terminal i.e. you container runs in fore ground -p port mapping, this maps a port on a vm with port on a docker containr

**Access tomcat**

http://docker-host-ip:8080

**Addiing custom user to docker group**

Such that we do not have to put sudo in front of every docker command.

sudo usermod -a -G docker ec2-user

sudo service docker restart

Relogin to VM

30-12-19

**Run docker:**

To learn docker files instructions We need a temporary environment to execute and check our instructions.

Lets create a/c in docker hub.

Go to docker.hub.com – to store our docker images

Why we store there ? – to easily shift to server

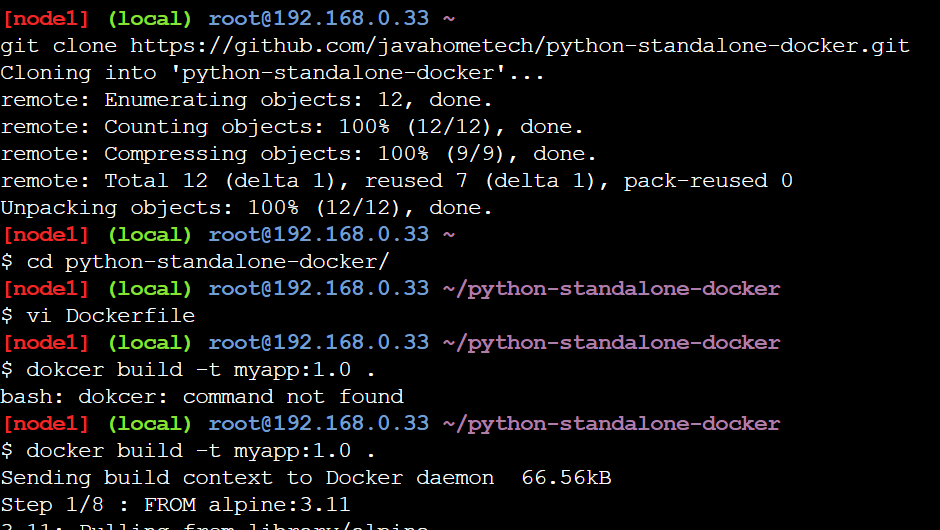
* Docker hub Is a docker public registry which contains docker images over internet.
* Play with docker :it’s a website which provides temporary docker environment which is useful for learning docker.
* Login – addnew instance -it gives new host with docker installed. Alt enter to max screen
* Add to git
* In play docker
* Git clone of that docker file
* ls – displays where you are
* Cd in to where you are
* Docker build -t (stands for tagging) myapp:1.0 . (. Is current directory)
* Docker images
* Vi docker file
* Our image is ready
* To run our application we have to create container from the Image.
* Docker run -d -p 8080:5000 myapp:1.0 (-d = run container in detached mode , that runs in background)

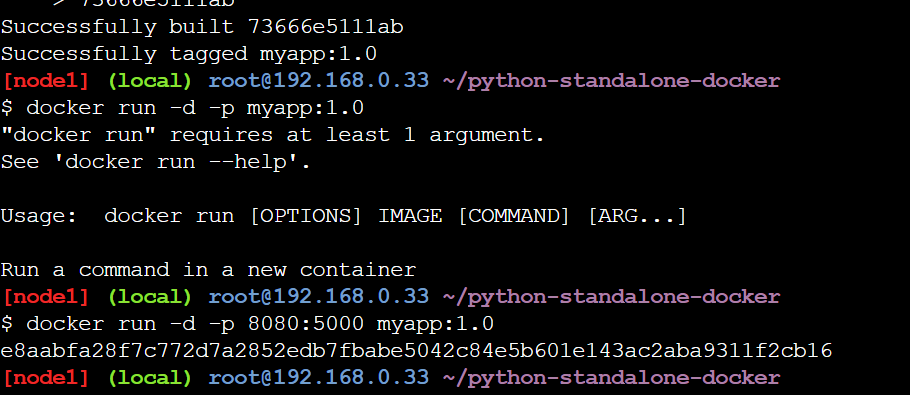
(-p is port mapping, 8080 is the host port, 5000 is container port, 🡪port used by the application inside the container)

You want t0 run, multiple container on single host , keep image below, if we hit the host , there will be conflicts it don’t know wherer to route bcoz all container ports are 5000 so we can change host ports 8080, 8081,8082

If we want to run another container

* Docker run -d -p 8081:5000 myapp:1.0
* It starts a container , our app is up and running.
* Docker start containerid
* Docker ps – displays container





**Writing Docker file:**

Docker file starts with FROM which contains our base image.

Always use base image – alpine is official image it is light weight and choose the one which is light weight, I can point to image which is having python no need to install python

Always use a specific version instead of pointing to latest tag

**FROM alpine:3.11**

**LABEL AUTHOR= “Hari Kammanna”**

**LABEL EMAIL=**[**lokesh@gmail.cm**](mailto:lokesh@gmail.cm)

It is just meta data , nothing to do with logic.

**LABEL LAST\_UPDATED=”30-Dec-19”**

**RUN –** used to run commands on the image ex:RUN

Ex: installing packages

**WORKDIR :** to set a specified directory as a present working directory – if that folder doesn’t exists it creates , if exists it uses the same.

WORKDIR /app -🡪 here present working directory is app , why this means I want to have separate folder for source code.

**COPY : copies files or folders from the host in to docker image**

**Ex:COPY app.py**

**EXPOSE :**  define the network port that this container will listen on run time

Ex:Expose 5050

All above are build time instructions to build image

**CMD:** cmd is executed at docker run time , it is useful for starting applicatiions when we run containers.

Ex:CMD python app.py

Cmd is run time instruction to run docker container

When we run docker run -d

**IQ) What is difference b/w RUN & CMD:**

* Run is build time instruction
* CMD is run time instruction

**IQ) can we use more than 1 cmd in one docker file ?**

Yes we can use ,but only one CMD is considered,it executes only last one.

**CMD**

**CMD:**

* Shell form

CMD command parap1 parap2 parap3

* Executable form

CMD [“command” , parap1 parap2 ]

CMD ping google.com

**Doker file**

From alpine:latest

CMD ping google.com

From alpine:latest

HOST\_NAME=google.com

CMD ping $HOST\_NAME

From alpine:latest

ENV HOST\_NAME=google.com

CMD[“ping”, “$HOST\_NAME”] -- this wont work

Docker run -it app ( it means interactuve terminal) - foreground

Docker run -d app ( it is in background)

Docker logs – to see background tasks.

[Difference b/w shell form and executable form:

* Shell form can substitute variables with its values
* Command form can not substitute variables

A CMD declared In a docker image can be over written at run time over command line, We can overwrite CMD at run time

From alpine:latest

ENV HOST\_NAME=google.com

CMD ping $HOST\_NAME

Docker run -it app echo “Java Home”

With above CMD python is replaced by java home

**ENTRY POINT:**

Instead of CMD we can use entry point like CMD it is run time instruction

It also has 2 forms

* Shell form
* CMD form

From alpine:latest

ENV HOST\_NAME=google.com

ENTRYPOINT ping $HOST\_NAME

* Docker build -t app .
* Docker run -it app ping facebook.com
* Inside docker image it is google.com we can’t overwrite with fb
* If we have entry file in docker file in shell form we can’t overwrite that at CMD line in the run time.

From alpine:latest

ENV HOST\_NAME=google.com

ENTRYPOINT [“ping”, “google.com”]

* Docker run -it app ping facebook.com
* It is appended

From alpine:latest

ENV HOST\_NAME=google.com

ENTRYPOINT [“echo”, “java home”] -- executable form

* If entry point is declared in docker file in executable form and at run time if we pass any commands, command line value gets appended with entry point declared in docker file
* Docker build -t app .
* Docker run -it app bangalore
* o/p : javahome banglore

**Using CMD & Entry point together:**

Yes we can use together

From alpine:latest

ENV HOST\_NAME=google.com

ENTRYPOINT [“PIng”] now o/p is ping google.com these gets appended

CMD [“google.com”] -- I can replace google at run time to faceboook.com docker run -it app faceb.com

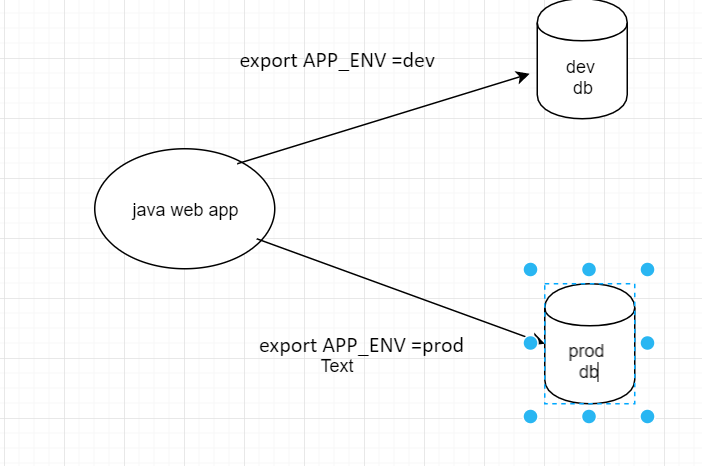
ENV instructions:

* It is to declare environment variables in the container.

Environment variables are helpful inorder to pass arguments to our application at run time.

(Lets say we have java web application , this connects to database i.e Dev it has ip,user name ,pass,port numbers will be different , we have one more db server in produ tion ,I have a docker image ,I want to run this app in dev.)

export APP\_ENV =dev or export APP\_ENV =prod for prod



Docker run -it -e HOST\_NAME=facebook.com app ( it runs foreground)

ARG:

Define a variable that users can pass a value to at build time

FROM alpine:latest

ARG HOST\_NAME=google.com

CMD ping “$HOST\_NAME”

**Docker commands:**

* Docker images
* Docker pull alpine:latest – I am pulling alpine image , it Is coming from docker hub

(when we say docker run it actullay pulls which is not present in the local and it creates container \*d

* Docker images

**If I want to delete image**

Docker rmi alpine:latest -- If it is in use we can’t delete , we have to remove container be4 this.

**If we want to delete multiple images**

Docker rmi alpine:latest image1:v2 image2:v3

**If I want to remove 100 of images**

docker pull alpine:latest

docker pull tomcat8:latest

docker images -q 🡪 I get only image Id’s

docker rmi $(docker images -q) -- that becomes argument , it deletes every images but not 1 image i.e alpine bcoz that is in use

docker run -itd alpine:latest – now this is In use so you cant delete

**Dangling images :**

The images which doesn’t have any tags

vi docker file

From alpine:latest

LABEL: NAME=HARI

Docker build -t hari:1 . - I build hari

If I go to docker file, do some change and build a new image

From alpine:latest

LABEL: NAME=HARI

LABEL: NAME=HARI2

Docker build -t hari:2 .

* Images which don’t have any tags is called dangling images

Docker images --“filter dangling=true”

Docker rmi $(Docker images --“filter dangling=true” -q) – if we put filter it gives un taged images.

**How do we run container**

Docker run -it d 8080:5050 kammana/nodeapp:v1 ( d stands for daemon , it – interactive terminal)

Ctrl C

Docker run -d -p 8080:5050 kammana/nodeapp:v1

Docker ps -- list running containers

Docker ps -a 🡪 running + exited containers display

We can

Docker stop containerid

Docker sart containerid

I want to get into container for trouble shooting

* Docker exec -it containerid bash

We are inside container now - I want to execute command – I want to get interactive terminal on this container, bash gives me terminal of container

* Exit
* Docker exec -it containerid ls -- it display container but we are still outside of container.
* I want to see **pwd**
* Docker exec -it containerid pwd
* Docker exec -it containerid ls /usr
* See environment variables
* Docker exec -it containerid printenv
* Check container logs :

Docker logs containerid

Containers can be integrated with Splunk /ELK for centralizing app logs

* Ls /etc/docker/
* $ cat /etc/docker/daemon.json
* **Docker volumes**

We have Docker host on top of it we have docker container on it we have container file system– any data on container if we delete it is lost , but it should n’t loose

For ex: we are ruuning statefull app like DB which stores data on container , if we loose container , we loose data on container

Our objective is we shouldn’t loose data even if we loose the container it is acheieved using docker volumes .

**IQ) what is volume:**

It is folder on the host machine data on the container can be mounted to volume.(now container is state less even though stateful app (my sql) )

Docker rm -f $(docker ps -qa) -- returns all container id which are running and exited , rm forcefully remove my containersthough running

Docker run -d alpine:latest

Docker ps

Docker ps -a --if no process is running it stops

Docker run -itd alpine:latest

Docker ps

Docker exec -it containerid ash

/ # ls

/ # mkdir hari

/ # cd hari

/hari # vi welcome.sh

/hari # ls

Welcome.sh

/hari # exit

Hari rm -f container id

$ docker run -itd alpine:latest

$docker exec -it containerid ash

/ # ls

/ # exit

Id we delete container also , data shouldn’t lost , create a new container and volume should point to that new one.

**Create a volume:**

Docker volume create mydata -- a folder is created on the host in following location

Docker volune inspect mydata ---/var/lib/docker/volumes/mydata/\_data ---(thisis path)

We can mount the above volume on to the container such that data on the container in the specific path is stored on the host in a volume

Docker run -itd -v mydata:/hari alpine:latest -- any data under /hari on container is actually stored on the volume i.e mydata

Docker exec -itcontainerid ash

/ # ls

/ # cd hari/

/hari # vi info.txt

Give some data

/hari # ls

/hari # exit

$ cat /var/lib/docker/volumes/mydata/\_data/

Cd /var/lib/docker/volumes/mydata/\_data/

$ ls /var/lib/docker/volumes/mydata/\_data/

Touch one.sh

Touch two.sh

Can I see these two files in container

Docker exec -it containerid ash

/ # cd hari/

Displyas one and two .sh

/haari # exit

Docker rm -f containerid

Docker run -itd -v mydata:/hari alpine:latest - we get one container id

Docker exec -it newcontainerid ash -- base image is alpine ash

/ # cd hari/

Displyas one and two .sh

/haari # exit

**Bind Mount:**

It stores data permenantly.

**Tmpfs Mount:**

It is stored in a RAM.

Can we use diff path ?

Yes, ( docker run - -name some-mongo -v /tmp/hari:/data db -d mang0:3.4 ) --ex:

Docker volume list

Docker volume rm mongodb - to remove volume – it is deleting entire volume

**Docker networking:**

**The current**  scope of discusiion is networking on a single host

How containers communicate on same host

There are diff types N/w drivers

**Bridge N/w ( default bridge)** – it is default network driver.

* By default containers join bridge network.
* Bridge allows comm b/w containers using IP’s , we can’t use host or container name.
* Docker run -itd –name=one alpine:latest -- it creates container and names as one
* Docker run -itd –name=two alpine:latest
* Docker inspect bridge
* Docker exec -it one ash
* / # ping 172.17.0.3
* (Docker network create hari
* Docker network list )

**Custom bridge :**

Custom bridge supports IP based, & container name based communication

i.e container names are resolved in to ip address

custom bridge has default built-in DNS

docker network create – driver=bridge javahome

* Docker run -itd –name=one --network=javahome alpine:latest -- it creates container and names as one
* Docker run -itd –name=two --network=javahome alpine:latest
* Docker inspect javahome
* Docker exec -it one ash
* / # ping 172.20.0.3
* / # ping two