Docker Installation on AWS AMI

- >sudo yum update –y
- >sudo yum install docker –y
- >sudo service docker start
- >sudo usermod –a –G docker ec2-user
- Start a new session so that the user group addition will take effect
- >docker --version

Git & docker-compose installation on AWS AMI

- >sudo yum install git
- https://docs.docker.com/compose/install/

Docker – Run hello-world container

- Let us run our first container ‘hello-world’ just to check if everything is fine
- >docker run hello-world
- >docker image ls
- >docker container ls
- >docker ps (lists running containers)
- >docker ps –a (lists all containers)

Docker – Run alpine container

- 5MB Linux Image
- >docker pull alpine
- What are the flags –i and –t??
- >docker run -i –t alpine /bin/sh
- Duplicate your bash session in AWS AMI

Docker – Run alpine container

- Pause a container
- On the other terminal, find out the container ID of the running Alpine container & pause it
- >docker ps
- >docker pause CONTAINER-ID
- >docker unpause CONTAINER-ID
- >exit -> will terminate the container

Named Container

- You can specify a name to your container by using --name flag
- >docker run -it --name my-linux alpine /bin/sh
- This will name the container as ‘my-linux’
- Easy to identify your containers and manage it

Modifications in the Container

- Start the alpine container instance again
- >docker run –it –name my_alp alpine /bin/sh
- Create few files in the alpine container
- $touch /tmp/file1.txt
- $touch /tmp/file2.txt
- Where are these files located??
- >docker diff my_alp

Docker start & run

- What is the difference between starting a container and running a container??

Restart Named container

- >docker ps –a
- >docker start my_alp
- >docker attach CONTAINER-ID
- Check if /tmp/file1.txt & file2.txt exist?

Detach from a Container

- CTRL+P CTRL+Q -> will detach the shell from the container without stopping it
- >docker stop CONTAINER-ID

Remove Container(s)

- >docker rm CONTAINER-ID
- One shot to remove all containers
- >docker rm $(docker ps –aq)

Rename Containers

- You can name or rename containers as per your wish
- >docker rename CONTAINER NEW_NAME

Container in Loop

- Let us start an alpine container in a loop which just prints date stamp every 5 seconds
- >docker run –itd alpine /bin/sh –c “while true;do date;sleep 5;done”
- Where are the logs printed?

Docker logs command

- You can check the logs of any container by specifying the container Id or name
- >docker logs CONTAINER-ID
- Or attach to the container to check the logs
- >docker attach CONTAINER-ID

Non Interactive & no tty

- What happens if you run a container without tty and in non-interactive mode
- Run the same loop example (previous one) and check if you can attach to the container and come out of it without terminating it??

>docker run –d alpine bin/sh -c "while true;do date;sleep 5;done"

Automatic Clean up

- If you want to remove a container after it exits, use the --rm option

>docker run -it --rm alpine /bin/sh

Docker inspect

- Get all kinds of information regarding your container using inspect command

>docker inspect CONTAINER_ID

- Can get information about only what you need

>docker inspect --format='{{ .State.Running }}' CONTAINER_ID

Docker save

- In case you want to distribute images without pushing them to a registry, you can save it and transfer to other docker machines

>docker save --output my-alpine.tar alpine:latest

- This will create an image copy in your folder

>docker load –input my-alpine.tar

Docker exec

- Allows you to run any command on a running container
- Start an interactive alpine container

>docker run –itd --name my-alp alpine /bin/sh

- Let us execute some commands on my-alp

>docker exec –it my-alp date

>docker exec –it my-alp whoami

>docker exec –it my-alp ps

Dockerfile

- Build File for creating your container
- Create Repos/my-alpine folder
- Inside it, create a Dockerfile with contents:

```
FROM alpine
CMD [“echo”, “Hello Containers”]
```

Dockerfile

- Build this Dockerfile with a tag/name
- >docker build . –t hello_alpine
- >docker image ls
- >docker run –it hello_alpine

Dockerfile – Java App

- Mkdir Repos/Java-App
- Create a simple Hello.java

```
class Hello{
    public static void main(String[] args){
    System.out.println("This is java app \n
by using Docker");
    }
}
```

Dockerfile – Java App

- Create a Dockerfile to compile & run

```
FROM java:8
COPY . /var/www/java
WORKDIR /var/www/java
RUN javac Hello.java
CMD ["java", "Hello"]
```

Dockerfile – From scratch

- Is base image always required??
- Check out this example :

https://github.com/rchidana/Image-From-Scratch

Docker MongoDB

- Let us run a NoSQL DB – MongoDB as a container, this runs on port 27017

>docker pull mongo:4.0.4

>docker run -d -p 8080:27017 --name my-mongo-server mongo:4.0.4

>docker exec -it my-mongo-server bash

- You are now inside the MongoDB container

Docker MongoDB

- Let us try few MongoDB commands in Mongo Shell client

#mongo

>show dbs

- Create a new database

>use thepolyglotdeveloper

Docker MongoDB

- Insert some data in to the database

>db.people.save({ firstname: “Sourav", lastname: “Ganguly" })

>db.people.save({ firstname: “Sachin", lastname: “Tendulkar" })

>db.people.find({ firstname: “Sachin" })

Docker MySQL

- Pull & run MySQL latest server image

mysql/mysql-server:latest in detached mode

- Figure out its root password (default one which is autogenerated)
- Log in to the container using the password and run few SQL commands

Docker MySQL

- Run the same server container by passing a password of your choice (use ENV variable)
- Connect to this container and run a few sql commands

Python Container

- Write a simple Hello.py code and run it in one command

Build Image - Manual

- Let us build a Facebooc(k) web app manually
- https://github.com/rchidana/facebooc
- Clone this repo first
- Pre-Req is ubuntu image

>docker run –itd --name fb –p 8085:16000 ubuntu bash

Build Image - Manual

- We need to copy the source files required to build Facebooc into the container
- From outside the container, copy ‘facebooc’ folder contents into the container

>docker cp facebooc fb:/opt

- Get into the container & navigate to /opt

>docker exec –it fb bash

>cd /opt

Build Image - Manual

- Continue with Build instructions
- Once application is built & checked satisfactorily, let us save the image now

>docker container commit fb anandr72/facebooc:v1

- Push this image onto your DockerHub registry if you want

>docker image push anandr72/facebook:v1

Build Image - Dockerfile

- Switch to ‘docker’ branch of this source code :

https://github.com/rchidana/facebooc

- Build an image of this using Dockerfile and push this to docker hub as version2

Build Cent-OS with OpenSSH Server

- Write a Dockerfile to build the following image (lets call it centos-sshd)
  - Based on Centos latest
  - Install openssh-server
  - Add new user called ‘student’, set his password as ‘student’
  - Add his ssh keys if needed
  - Run openssh daemon on port 22

Build Cent-OS with OpenSSH Server

- Start a container powered by Centos-ssh

>docker run -d --name sshd -p 8085:22 centos-sshd

- Connect to ssh server

>ssh -p 8085 student@localhost

- Enter student as password
- Verify if you get into the CentOS container

Build Ubuntu + Apache2

- Write a Dockerfile to build Ubuntu based Apache2 server which runs on port 80
- Modify this file to inject an index.html of your choice (which needs to be copied from outside the container)

Docker Volumes – Ex1

- >docker volume create OutSideContainer
- Start a docker container with rm option
- >docker run –it --rm –v OutSideContainer:/InsideContainer alpine /bin/sh
- $cd /InsideContainer & create file1.txt & file2.txt
- Exit container and check if the volume still persists?

Docker Volumes – Ex1

- >docker inspect OutSideContainer
- Try to find out the actual content of the volume and see if you can list out the contents

Docker Volumes – Ex1

- Start another container and mount this volume inside it at a different mount point

>docker run -it -v OutsideContainer:/new_mount alpine /bin/sh

Docker Volumes – Ex2

- Create a new Data volume at run time

>docker run –it –- name cont2 –v datavolume2:/datavolume2 alpine /bin/sh

- Create some files in /datavolume2 inside the container
- Exit the container

Docker Volumes – Ex2

- Restart cont2

>docker start –ai cont2

- Check if the files that you created earlier are still present
- Exit the container and try to remove datavolume2

>docker volume ls

>docker volume rm datavolume2

Docker Volumes-from

- You can mount volumes from named containers using --volumes-from
- Start a container by name cont4 and create datavolume4 in it

>docker run –it –v /data --name cont4 alpine /bin/sh

- Create some files in /data and exit the container

Docker Volumes-from

- Start cont5 and mount all volumes that were mounted in cont4

>docker run –it –volumes-from cont4 –name cont5 alpine /bin/sh

- Check volumes & data
- Add some more data and exit
- Restart cont4 and check if modified data is present in the volumes

Docker Network - nginx

- Refer to Network Slides
- Start nginx server
- >docker run –name my-nginx –p 8085:80 –d nginx

Docker Network - Ghost

- Javascript Blogging platform
- >docker run –itd --name my-ghost –p 8085:2368 ghost:alpine
- Check if Ghost is up & running on port 8085
- http://localhost:8085/

Docker Compose

- Tool to define & run multiple containers
- Define services in YAML file
- Install Docker Compose :

https://docs.docker.com/compose/install/#install-compose

- >docker-compose --version

Docker Compose - WordPress

- https://github.com/rchidana/Cred-Suisse
- >cd Repos/Word-Press
- Create “docker-compose.yml”
- Please check the ports
- >docker-compose up –d
- http://MACHINE_VM_IP:8080

Docker Compose - WordPress

- Remove containers, network & volumes
- > docker-compose down
- >docker-compose down --volumes

Docker Compose - WebApp

- Example to build & run two containers, one powered by SQL and other built locally

>https://github.com/rchidana/Web-Compose

- Check the contents of docker-compose.yml
- Let us first build this & launch it

>docker-compose build

>docker-compose up –d

Docker Compose - WebApp

- Updates to any container contents can be handled by rebuilding the linked images
- Make some changes to index.html

>docker-compose build

- Stop previous containers & launch new one

>docker-compose down

>docker-compose up -d

Docker Health Checks

- Can include any periodic probes into your container to check its health

```
HEALTHCHECK --interval=1m --timeout=3s \
CMD curl -f http://localhost/ || exit 1
```

- Exit status = 0, success
- Exit Status = 1, unhealthy
- Health Check in Swarm node can be used to restart containers

Docker Debug & Events

- If needed, Docker daemon can be started or any container can be run in debug mode
- Additional information gets printed

>docker run –D –it hello-world

- Get real time events from server

>docker events --since '10m‘

>docker events --since '2017-01-05T00:35:30' --until '2017-01-05T00:36:05'

Docker Compose - Swarm

- Can use Docker Compose to submit job to a Docker Swarm

>docker swarm init

>docker stack deploy --compose-file XYZ NAME

>docker stack services NAME

>docker stack rm NAME

https://github.com/dockersamples/example-voting-app

Docker Swarm - Persistence

- By default, data & volume created in Swarm is tied to the node running the container
- Can use 3rd party plugins for persisting data across a cluster
- CloudStor – Volume plugin for AWS & Azure
- https://rexray.io/

Questions??