Docker - The Container Virtualization Tool

==========

Day-1:

###############

#

Diff between..

- Physical server
- Virtual machine
- Docker container

VM, Docker, usage in DevOps.

#

what is docker? why docker?

#

Supported Platforms -

- Docker is supported on
  - Linux platforms
    - Ubuntu, RHEL, CentOs..etc.
  - Windows
  - OS X
- Cloud Platforms
  - Amazon EC2
  - Rackspace Cloud
  - Google compute Engine..etc.
  - Azure

Note:

Linux containers can be created on Windows and OS-X.

HOW?- Windows & Mac Docker installers contain a tiny Linux virtual machine.

So, Docker creates linux container on top of this tiny Linux VM.

Requirements:

- 64-bit architecture
- Linux 3.8 or later Kernel versions

#

Requirements Check:

- Check Kernel version
    $ uname -a
    $ uname -r

- Check OS name:
    $ lsb_release -a / -cs
    $ cat /etc/os-release

Installation Steps:
==========
Update apt-get cache
    $ sudo apt-get update

Install docker dependencies
    $ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

Add GPG key to apt repository
    $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Setup Stable repository (Add docker download URL manually to apt cache)
    $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Update apt package index:
    $ sudo apt-get update

Install Latest version of Docker
    $ sudo apt-get install docker-ce

#
Installation Check
sudo docker --version

#
uninstall docker:
$ sudo apt-get remove docker-ce
$ sudo rm -rf /var/lib/docker (Removes all containers and images)

DAY-2
#
Managing docker containers
==========
Create a new container using below command

$ sudo docker run -it ubuntu /bin/bash

- "docker run" command provides all lanuching capabilities for docker to create a container.
- we use `docker run` to create new containers.

-i : opens STDIN from the container
-t : tells docker to assign a pseudo-tty to the container.
-it : provides interactive shell

ubuntu : Is an image and also called as "stock image" or "base image".
This image will be downloaded from Docker hub when we run 'docker run' command first time.
/bin/bash: 'shell program' that will be installed in the terminal.

Inspect the new container.. Let's beleive that it's separate machine
1.
hostname
2.
cat /etc/hosts

3. hostname -i

5. ps -ef

6. cd / && pwd && ls

How to setup SSH for containers
Setup SSH in the container so that other machies can communicate with it

- Create a container using below command
    $ docker run -it ubuntu /bin/bash
- Find it's IP address
    $ hostname -i (ex: 172.17.0.2)

- Try to connect to 172.17.0.2 from the host machine.
  You will get below error as the container doesn't have ssh server running.
    $ ssh gamut@172.17.0.2

- Now, Install ssh server using below commands
$ apt-get update
$ apt-get install openssh-server

- Start the server
$ service ssh start (other ssh commands: status/stop/restart)

- Create an user and set up password
$ useradd -m -d /home/gamut -s /bin/bash gamut
$ passwd gamut

- Now try to connect to the container using ssh from the host machine
  The connection should be succussful as the SSH is installed and the server is running.
$ ssh gamut@172.17.0.2

To Enable root user over ssh for ubuntu containers, we need to change below file.
  Once this change is done, you can connect to the container using 'root' user.
Add below line under "# Authentication:" in "/etc/ssh/sshd_config"
    PermitRootLogin yes

#
Manage/Run Docker as a non-root user
------OR------
Run docker command without 'sudo'
==========

1. Create the "docker" group
    $ sudo groupadd docker

2. Add user to the group

$ sudo gpasswd -a gamut docker (add)

3. Logout and Login back so that your group membership is re-evaluated.

-- verification and cleanup

4. Check group existance:
$ grep docker /etc/group

5. Delete the group
$ sudo groupdel docker

6. Remove user from the group
$ sudo gpasswd -d gamut docker (remove)

#
Shotdown a container
"exit" to stop the container

#
Login to a stopped container
$ docker start
$ docker attach
$ docker start -ai mystifying_tu

List all containers(stopped and running)
$ docker cotainer ls -a
$ docker ps -a

List given no. of containers
$ docker ps -a -n2

List running containers only
$ sudo docker cotainer ls
$ docker ps

List Stopped containers only
$ docker container ls -f status=exited (Where Status can be exited/running)

"docker container ls" command output shows
- Image name from which container is created
- ID - container can be identified using short UUID, longer UUID Or name.
- Status of the container (Up / Exited)
- Name of the container

show the last container which you have created (stopped/running)
sudo docker container ls -l

Naming the container
docker run --name gamut -it ubuntu /bin/bash

Note: Two containers can't have the same name.

Rename a container
$ docker rename db-server3 db-server-name3

Deleting a container by giving it's name or ID
$ docker rm ID/name

Delete all (running/stoped) containers at once
$ docker rm -f `docker container ls -a -q`
$ docker rm -f $(docker container ls -a -q)
$ docker rm -f $(docker ps -a -q)

Delete running containers only
$ docker rm -f $(docker container ls -q)
$ docker rm -f $(docker ps -q)

list stopped containers only
$ docker container ls -f status=exited

Starting a stopped container
sudo docker start gamut
sudo docker stop aa3f365f0fdocker4e
sudo docker restart gamut

Attaching to a running container
docker attach gamut

docker attach b1b1c8dc1939

Run a linux command remotely in a container Or Get an independent terminal from a container remotely (from Host)
$ docker exec -it tomcat-server /bin/bash

Stoping a container from 'host machine'
- docker stop dgamut (Gracefully stop the container)
- docker kill dgamut (Forcibly stop the container)

Shortcut Keys
Ctrl + p + q - push a running container in background mode.
Ctrl + d - short cut to 'stop' a container.

Inspecting the container's processes from host machine
$ docker top <container-name>

Show last 2 containers (stopped/running)
$ docker ps -a -n2

Create a container in a background mode (without termial access)
$ docker run -it -d ubuntu /bin/bash

Find More About The Container
- 'docker inspect' comamnd interrogates the container and returns complete information about it.

Ex: image name, IP, Memory details, hostname ..etc

Examples:
$ docker inspect <container-name>
$ docker inspect -f '{{.Config.Hostname}}' tomcat-server1
$ docker inspect -f '{{.NetworkSettings.Networks.bridge.IPAddress}}' tomcatserver1

Note: use --format (OR) -f

List all container names
$ docker inspect --format "{{.Name}}" $(docker ps -a -q) | tr -d '/'

STATS:
==========
Display usage statistics of a container
	$ docker stats --no-stream <container-name>
	$ docker stats --no-stream --all

	$ docker stats --no-stream --format {{.MemUsage}} sleepy_shannon
	$ docker stats --no-stream --format {{.CPUPerc}} sleepy_shannon

#
Allocating memory for a container (below command allocates 1 GB RAM)
	$ docker run -it --name tomcat-server -m 1g ubuntu /bin/bash
	$ docker run -it --name tomcat-server -m 1024m ubuntu /bin/bash

#
Updating memory of an existing container
	$ docker update -m 2024m tomcat-server

CPU Allocation
$ docker run -it --cpus="2" --name jenkins-server ubuntu /bin/bash
$ docker upddate --cpus="2" jenkins-server

--> weekend-7:00 batch

Push a container in sleep mode
$ docker pause db-server1
$ docker unpause db-server1

--> Jenkins

DAY-4:
Docker Images
================
Agenda:
	- Understand docker Images and application
	- Advantages of Docker Images

- Create docker Image for your application
- Share/publish your Image
- Examine Docker repositories that hold images

- Docker images are the building blocks for creating container
- From images, we launch containers.

Advantages of Images in Build and Deployments OR DevOps world!

a. Works In my machine problem.
b. Developers can quickly setup local development environments as we can include all dependencies in the image and create containers.
c. Is there an Issue? don't spend time to troubleshoot it. Just throw the machine which has the issue away and create new instantly.
d. Auto scale your environment very easilly.
e. No need to live with complex, redundant configurations. You can create disposable environments.
f. You can leverage/utilizes local machines's computing power when you need to test your code on multiple machines, instead of waiting for DevOps team to supply or wasting extra computing power. you already have 500GB, 16GB RAM right? are you utilising it? NO! then why again you need extra hardware?
g. You can create new environments within few minutes (ex: create new performance testing environment within few minutes before the release)

Listing docker images
- $ docker image ls

- Images live in '/var/lib/docker/image/overlay2/imagedb/content/sha256'
- Containers live in '/var/lib/docker/containers'

Building our own Image
We have 2 Ways to create docker image:
1. docker "commit"
2. docker "build" cmd & Dockerfile

Creating docker image using "docker commit" command
==========
PROJECT-1:

Goal: Create the docker image to ship the application code along with nginx configurations.

- Create container
    $ docker run -it --name nginx-container ubuntu /bin/bash

- Install nginx manually
    $ apt-get update
    $ apt-get install -y nginx

- Deploy some application code into '/var/www/html' (this is deployment path for nginx)
    ex: deploy below index.html as a code

```
=======
<html>
    <body>
      <h1 style="color:red;">Wiculty Learning Solutions</h1>
    </body>
</html>
=======
```

- Create docker image from the container (OR)
- Convert docker container as docker image..
    $ docker commit nginx-container nageshvkn/nginx-img

Syntax: $ docker commit <container-name> <image-name>

- Check if image has been created
    $ docker image ls

- Push the newly created image to docker hub
      - Create an account in 'https://hub.docker.com/'.
    $ docker login
    $ docker push nageshvkn/nginx-img

Note: Now you have succussfully containerized your application and published the iamge to DockerHub. Customers can spin millions of new containers using the above docker image.

Note: To verify your image as an user, create a container as shown below. Remove existing image that you have created so that you can abserve image download from

Docker hub clearly. (to remove the image.. $ docker rmi nageshvkn/nginx-img)

$ docker run -it nageshvkn/nginx-img /bin/bash

- Launch the application to test if application is configured along with dependencies.

http://172.17.0.2:80

Note:
start/stop/restart nginx server:
==========\==========\==========
$ sudo service nginx start
$ sudo service nginx stop
$ sudo service nginx restart
$ sudo service nginx status

Note:
uninstall nginx using below comamnd
$ sudo apt-get purge nginx nginx-common

PROJECT-2:
==========
DAY-6:
Creating docker image using "docker build" command
==========

- mkdir gamutgurus
- cd gamutgurus
- touch Dockerfile

--> 'gamutgurus' directory is called "context" or "build context".
It contains the code, files or other data that you want to include in the image.

- Write Dokckerfile:

```
FROM ubuntu:16.04
MAINTAINER "info@gamutgurus.com"
RUN apt-get update
RUN apt-get install -y nginx
COPY index.html /var/www/html
```

ENTRYPOINT service nginx start && bash

index.html:
=======
```
<html>
    <body style="background-color:powderblue;">
      <h1 style="color:red;">Gamug Gurus Online Training Portal</h1>
       </body>
</html>
```

Building docker image:
```
$ cd gamutgurus
$ docker build -t "nageshvkn/nginx-img" .
```

Note: Building the image if 'Dockerfile' has different name.
Use "-f <YourDockerfileName>" option.
Example: $ docker build -f MyDockerfile -t="nageshvkn/nginx-img" .

Listing docker image
```
$ docker image ls
```

Create an account in docker hub

Pushing custom images to docker repository
```
$ docker login
$ docker push nageshvkn/nginx-image
```

#
Testing Image

Remove the local image so that it will be downloaded from Docker Hub.
```
$ docker rmi nageshvkn/nginx-image (OR)
$ docker image rm nageshvkn/nginx-image
```

Creating a new container from our image
```
$ docker run -it --name nginx-container nageshvkn/nginx-img /bin/bash
```

Verify if nginx is running from the container.
```
$ http://172.17.0.2:80
```

#
User Images Syntax:
	nageshvkn/nginx-img (username/imagename)

Official Images Syntax:
	ubuntu

Specifying Image via tags
	- ubuntu:16.04
	 ubuntu- is image name
	 16.04 - is called tag

Pulling the images
	- docker pull tomcat

Searching docker images in docker hub
	- docker search puppet
	- docker search jenkins

Deleting an Image
	- docker rmi gamut/nginx

Deleting all Images
	- docker rmi $(docker images -q)

DAY-7:
#
Docker Image layers
==========
- Checking the history of our docker image
	$ docker history <Image-Id>

Build image without using build cache
	- Use --no-cache option
	$ docker build --no-cache -t="username/imagename" .

Container creation process - Deep dive

How contaner is created:

Writable Layer
-
Gamutkart application
Apache image
nginx image
-
Ubuntu Base Image(rootfs)
-
Bootfs:
    cgroups, namespace, lxc, devicemapper/aufs..etc.
Kernel

Interview Notes:
==========
#
Docker Benefits:
==========
1. Portability:

ship environments from one type of infrastructure to another without building up new VMs and tearing down the old ones.

2. Quick deployment/Teardown:

With a single command, you can spin up new containers or tear down existing ones. If you want to clone a new VM, you are looking at waiting for close to or over a few hours. With Docker, it will take a few minutes to achieve what you need.

3. Consistency:

No more of the "works on my machine!" excuse. As everyone uses the same images to work, consistency is always guaranteed. All the container will act the same in your environment as it will on others.

4. Managing infrastructure-like code:

When it comes to upgrades, you can simply update your Dockerfile, and then tear down the old one. This helps not only with updates, but it can also help with rollbacks as well.

Volumes:
==========
List all volumes available in host machine
	$ docker volume ls

Create a new Volume
	$ docker volume create deployment_code

Check Mount point directory
	$ docker inspect deployment_code

Mount Volume(deployment_code) to a new container
	$ docker run -it -v deployment_code:/deployment_code ubuntu:16.04 /bin/bash

Create 'Read-only' Volumes
	$ docker run -it -v deployment_code:/deployment_code:ro ubuntu:16.04 /bin/bash

Removing a Volume
	$ docker volume rm deployment_code

Remove all unused Volumes
	$ docker volume prune

Note:
Creating Volume with your own directory in host machine (OR)
Crreating Volume with existing directory in the host machine
	$ docker run -it -v /home/gamut/Distros:/Distros ubuntu /bin/bash

List down all containers which are using a particular volume
	$ docker ps -a --filter volume=deployment_code

#
Docker Registry
Docker Repository
Image tags

DAY-8:

Gamutkart Real-time application
==========

Agenda:
How do you containerize or dockerize your application?
Can you explain how you have implememnted Docker for your application?

1. Clone the source code from Git or any other V.C.S
    $ git clone https://github.com/nageshvkn/gamutkart2.git

2. Build the code using your favourate build tool Maven/ANT
    $ mvn install

3. Create docker image for the application(gamutkart2) with
   war file, tomcat,jdk...etc using below Dockerfile.
    Dockerfile:

    FROM ubuntu:16.04
    MAINTAINER "info@gamutgurus.com"
    RUN apt-get update
    RUN apt-get install -y openjdk-8-jdk
    ENV JAVA_HOME /usr
    ADD apache-tomcat-8.5.38.tar.gz /root
    COPY target/gamutkart.war /root/apache-tomcat-8.5.38/webapps
    ENTRYPOINT /root/apache-tomcat-8.5.38/bin/startup.sh && bash

4. Build the Image using below command
    $ docker build -t "nageshvkn/gamutkart-img" .

4A. Push the image to docker hub.
    $ docker push nageshvkn/gamutkart-img

5. Run below shell script to create an environment with give no. containers
    $ ./create-env.sh 10

6. Observer all containers created using above script ($ docker ps)

7. Launch the gamutkart application from all containers.
    $ http://IP:8080/gamutkart

Docker and Jenkins Integration