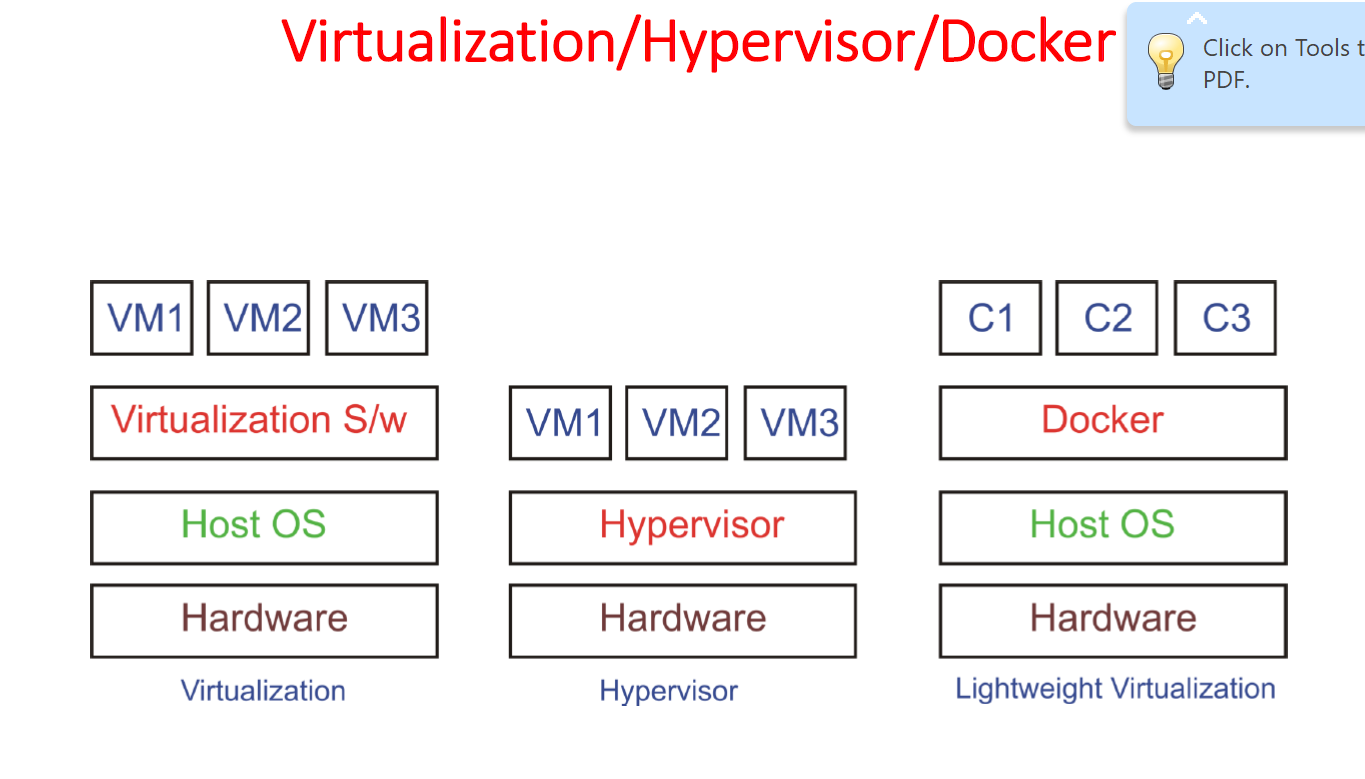
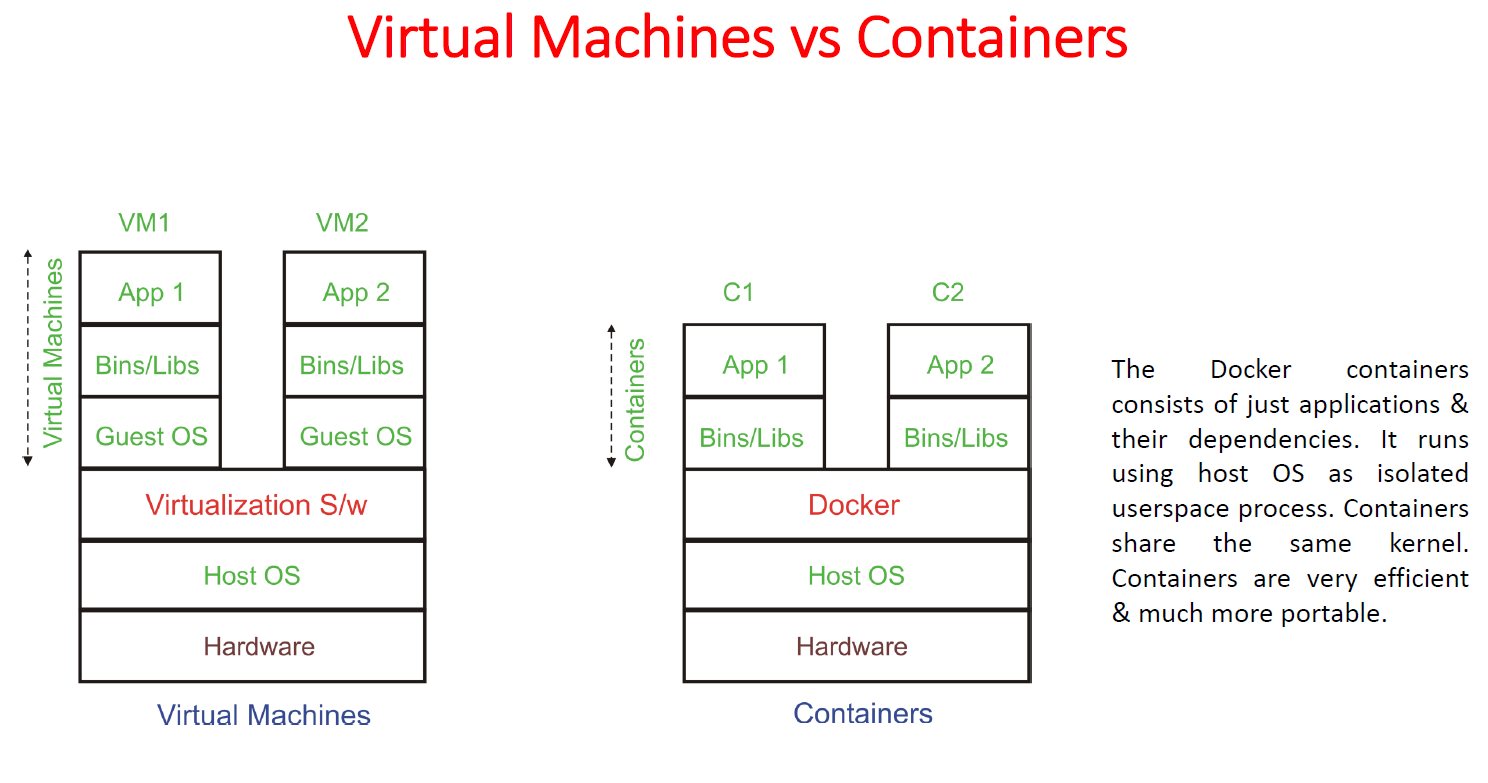
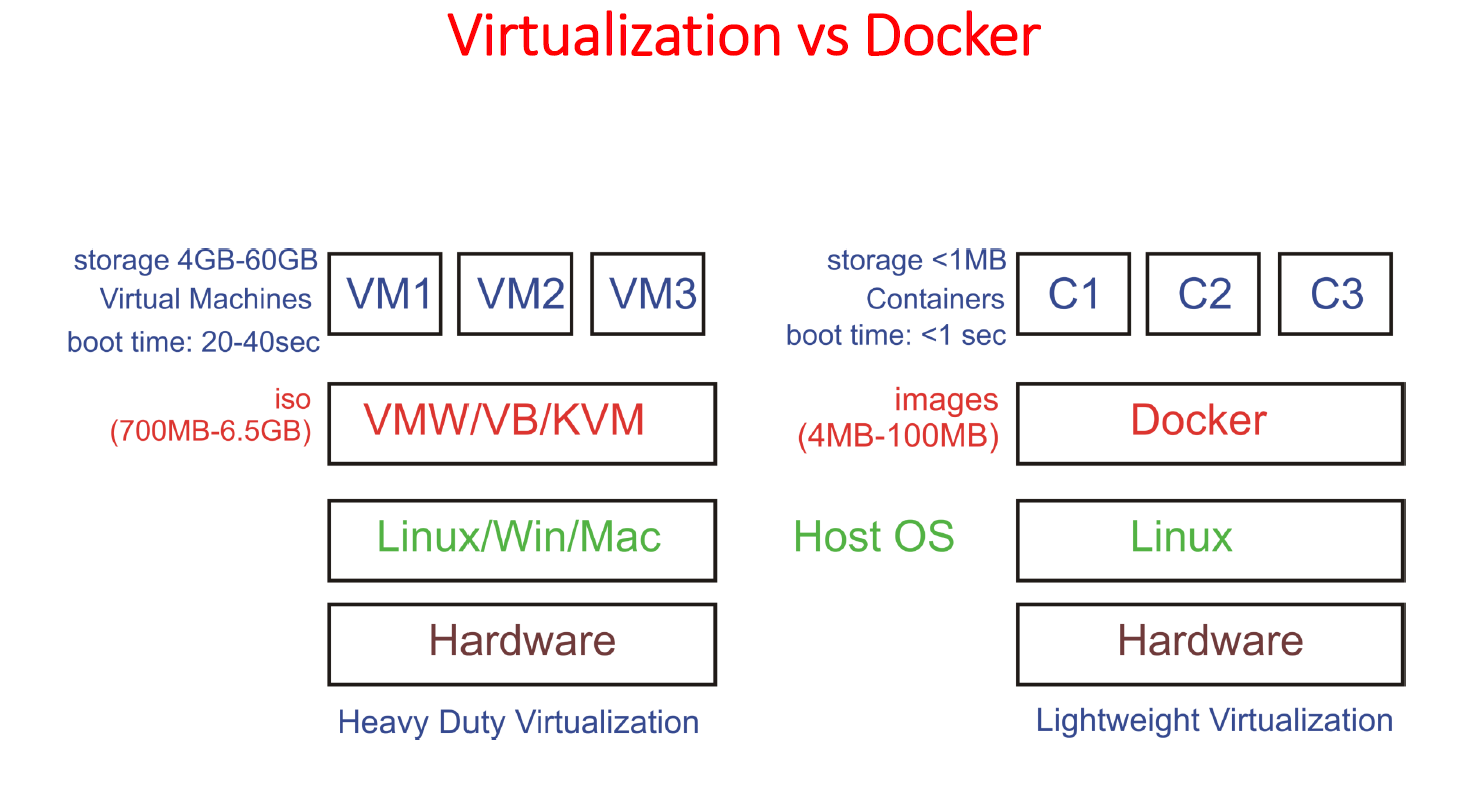
Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

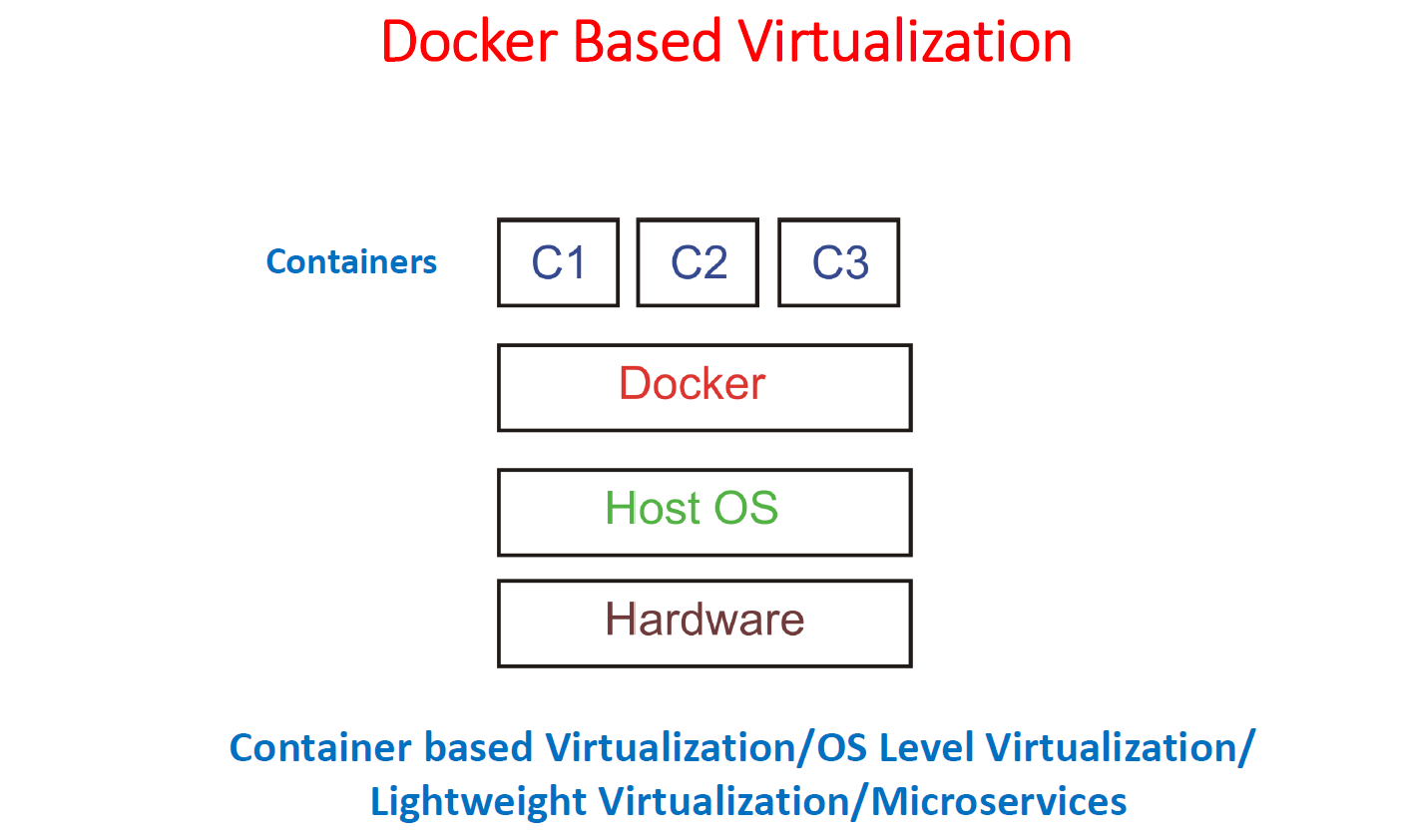
The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.

Advantages of docker:









Features of Docker

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.

2. With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.

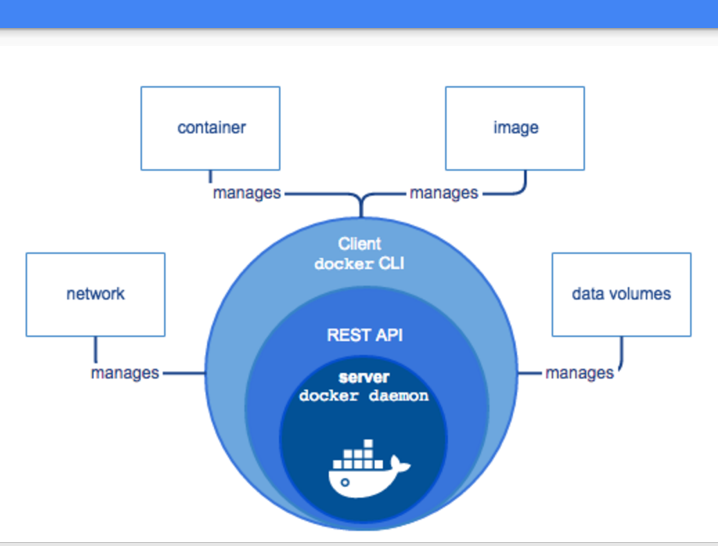
3. You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.

4. Since Docker containers are pretty lightweight, they are very easily scalable.

Components of Docker

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker has the following components



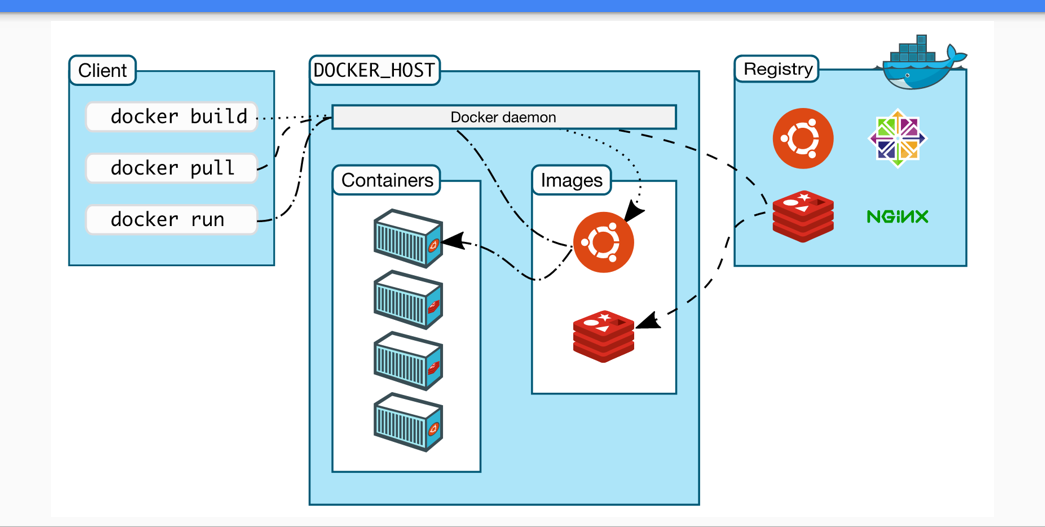
1. Docker for Linux − It allows one to run Docker containers on the Linux OS.

2. Docker Engine − It is used for building Docker images and creating Docker containers.

3. Docker Hub − This is the registry which is used to host various Docker images.

4. Docker Compose − This is used to define applications using multiple Docker containers.

Architecture of Docker:



Installing Docker on Linux

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Method 1: Using Official Docker Repository

Method 2: Using Script

Method 3: Using Extra Repository

yum install -y yum-utils device-mapper-persistent-data lvm2

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

yum -y install docker-ce

systemctl start docker

systemctl enable docker

Method 1: SET UP THE REPOSITORY

===============================

1. Install required packages. yum-utils provides the yum-config-manager utility, and device-mapper-persistent-data and lvm2 are required by the devicemapper storage driver.

# yum install yum-utils device-mapper-persistent-data lvm2

2. Use the following command to set up the stable repository. You always need the stable repository, even if you want to install builds from the edge or test repositories as well.

# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

3. Optional: Enable the edge and test repositories. These repositories are included in the docker.repo file above but are disabled by default. You can enable them alongside the stable repository.

# yum-config-manager --enable docker-ce-edge

# yum-config-manager --enable docker-ce-test

You can disable the edge or test repository by running the yum-config-manager command with the --disable flag. To re-enable it, use the --enable flag. The following command disables the edge repository.

# yum-config-manager --disable docker-ce-edge

INSTALL DOCKER CE

1. Install the latest version of Docker CE, or go to the next step to install a specific version:

# sudo yum install docker-ce

If prompted to accept the GPG key, verify that the fingerprint matches 060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35, and if so, accept it.

Got multiple Docker repositories?

If you have multiple Docker repositories enabled, installing or updating without specifying a version in the yum install or yum update command always installs the highest possible version, which may not be appropriate for your stability needs.

Docker is installed but not started. The docker group is created, but no users are added to the group.

2. To install a specific version of Docker CE, list the available versions in the repo, then select and install:

a. List and sort the versions available in your repo. This example sorts results by version number, highest to lowest, and is truncated:

# yum list docker-ce --showduplicates | sort -r

# yum install -y --setopt=obsoletes=0 docker-ce-17.03.2.ce-1.el7.centos docker-ce-selinux-17.03.0.ce-1.el7.centos

docker-ce.x86\_64 18.09.0.ce-1.el7.centos docker-ce-stable

The list returned depends on which repositories are enabled, and is specific to your version of CentOS (indicated by the .el7 suffix in this example).

b. Install a specific version by its fully qualified package name, which is the package name (docker-ce) plus the version string (2nd column) up to the first hyphen, separated by a hyphen (-), for example, docker-ce-18.03.0.ce.

# sudo yum install docker-ce-<VERSION STRING> yum install yum install docker-ce-

Docker is installed but not started. The docker group is created, but no users are added to the group.

c. Start Docker.

# sudo systemctl start docker

d. Verify that docker is installed correctly by running the hello-world image.

# sudo docker run hello-world or docker version

Method 2: Using the script

===============================

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh

Method 3: Using repository

===============================

yum -y install docker

systemctl start docker

systemctl enable docker

docker -v

#To verify Docker is installed successfully and that you can run docker commands without sudo run the following command which will download a test image, run it in a container, print a “Hello from Docker” message and exit:

docker container run hello-world

DOCKER IMAGES

#To search the Docker Hub repository for an image just use the search subcommand. For example, to search for the CentOS image, run:

docker search centos

docker search --limit 5 centos

#If we want to download the official build of CentOS 7, we can do that by using the image pull subcommand:

docker image pull centos

#Depending on your Internet speed, the download may take a few seconds or a few minutes. Once the image is downloaded we can list the images with:

docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest e38bc07ac18e 3 weeks ago 1.85kB

centos latest e934aafc2206 4 weeks ago 199MB

#If for some reason you want to delete an image you can do that with the image rm [image\_name] subcommand:

docker image rm centos

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DOCKER - CONTAINERS

#An instance of an image is called a container. A container represents a runtime for a single application, process, or service.

#It may not be the most appropriate comparison but if you are a programmer you can think of a Docker image as class and Docker container as an instance of a class.

#We can start, stop, remove and manage a container with the docker container subcommand.

#The following command will start a Docker container based on the CentoOS image. If you don’t have the image locally, it will download it first:

docker container run centos

#The switch -it allows us to interact with the container via the command line. To start an interactive container type:

docker container run -it centos /bin/bash

docker container run -dit centos /bin/bash --> if we want to run permanently in the background

or

ctrl+p+q ===> to come out of container without exiting

#To list active containers, type:

docker container ls

To view both active and inactive containers, pass it the -a switch:

docker container ls -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

79ab8e16d567 centos "/bin/bash" 22 minutes ago Up 22 minutes ecstatic\_ardinghelli

c55680af670c centos "/bin/bash" 30 minutes ago Exited (0) 30 minutes ago modest\_hawking

c6a147d1bc8a hello-world "/hello" 20 hours ago Exited (0) 20 hours ago sleepy\_shannon

To delete a container multiple containers just copy the container ID (or IDs) from above and paste them after the container rm subcommand:

docker container rm c55680af670c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To see running container

docker ps

To see container full information

docker inspect <container id >

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To view the running containers

docker ps

To view the list of containers that are not active

docker ps -a

To stop a contaner

docker stop <container id or container name>

To start a container

docker start <container id or container name>

To stop any riunning container

docker stop $(docker ps -q)

now To remove a unwanted container

docker rm <container id or container name>

To see the container id or container name

docker ps -a

To remove all the containers

docker rm $(docker ps -a -q)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DOCKER-IMAGES

We must know the exact repository name to pull corresponding docker images. So use "docker search" command to find the correct repository name to download the particular docker images.

For Hello-World:

[root@docker-host ~]# docker search hello-world

For CentOS:

[root@docker-host ~]# docker search centos

#How to list available docker images?

Use "docker images" command to know the list of docker images available in your local docker host.

[root@docker-host ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/centos latest 49f7960eb7e4 2 weeks ago 200 MB

docker.io/hello-world latest e38bc07ac18e 2 months ago 1.85 kB

[root@docker-host ~]#

#How to know detailed information about a Docker Images?

Use "docker inspect" command to know detailed information about a Docker Images available locally using Docker Image ID.

This will give lot of information about Docker Images as below.

[root@docker-host ~]# docker inspect 49f7960eb7e4

#How to get history of a Docker Images?

Use "docker history" command along with Docker Image ID to get a history of a Docker Images.

[root@docker-host ~]# docker history e38bc07ac18e

How to remove or delete Docker Images?

use "docker images" to find the Docker Image ID or Image Name and use "docker rmi" command to remove or delete docker images.

[root@docker-host ~]# docker rmi 49f7960eb7e4

To remove multiple docker images, mention the docker images ID separated by spaces as below.

[root@docker-host ~]# docker rmi 49f7960eb7e4 47fjay738290 987tr78rt5g5

To remove all docker images in a single command,

[root@docker-host ~]# docker rmi $(docker images -a -q)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dockerfile Instructions Explained with Examples

vim dockerfile

FROM docker.io/centos-6.6.0

MAINTAINER Devops Engineer

RUN yum update && yum -y install httpd

RUN mkdir -p /data/myscript

WORKDIR /data/myscript

CMD python app.py

after you created the above file then run the below command

docker build .

another dockerfile for more easy understanding

vim dockerfile --> creating the dockerfile via vim editor

FROM docker.io/centos

MAINTAINER admin

RUN yum update -y && yum -y install httpd

RUN echo "Welcome to our homepage created using dockerfile" > /var/www/html/index.html

EXPOSE 80

CMD apachectl -D FOREGROUND

after you created the above file then run the below command

docker build .

[root@server1 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 6f3baff9b180 2 hours ago 200MB

<none> <none> 3b1943a5f478 2 hours ago 200MB

<none> <none> 7770d3b057fc 3 hours ago 362MB

centos latest 5182e96772bf 2 weeks ago 200MB

nginx latest c82521676580 4 weeks ago 109MB

hello-world latest 2cb0d9787c4d 6 weeks ago 1.85kB

[root@server1 ~]# docker tag 6f3baff9b180 vijaycnl.com/httpdserver

[root@server1 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

vijaycnl.com/httpdserver latest 6f3baff9b180 2 hours ago 200MB

<none> <none> 3b1943a5f478 2 hours ago 200MB

<none> <none> 7770d3b057fc 3 hours ago 362MB

centos latest 5182e96772bf 2 weeks ago 200MB

nginx latest c82521676580 4 weeks ago 109MB

hello-world latest 2cb0d9787c4d 6 weeks ago 1.85kB

Explanation

1. Used FROM instruction to specify the image.

2. Used MAINTAINER instruction to specify information about the author.

3. Used RUN instruction to update all packages and install the httpd package

4. Again used RUN instruction to create a homepage "index.html" under the apache default directory /var/www/html.

5. Used EXPOSE instruction to open the listening port 80 of httpd service.

6. Used CMD instruction to run a apache service command as an executable when container is launched.

[root@docker-host /]# docker images

docker run -dit --name cloudnloud-web -p 8080:80 -v /var/tmp/:/usr/local/apache2/htdocs/ httpd:2.4

8080 port --> base linux machine port

80 --> docker image port

/var/tmp/ --> base linux machine directory

/usr/local/apache2/htdocs/ --> docker image directory -->(goto docker hub and check it)

vim /var/tmp/docker.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Learn Docker at cloudnloud.com</title>

</head>

<body>

<h1>Learn Docker With Us</h1>

</body>

</html>

Create a index.html under the path /var/tmp and try login in and see the difference

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Purging All Unused or Dangling Images, Containers, Volumes, and Networks

Docker provides a single command that will clean up any resources — images, containers, volumes, and networks — that are dangling (not associated with a container):

docker system prune

To additionally remove any stopped containers and all unused images (not just dangling images), add the -a flag to the command:

docker system prune -a

Removing Docker Images --> Remove one or more specific images

Use the docker images command with the -a flag to locate the ID of the images you want to remove. This will show you every image, including intermediate image layers. When you've located the images you want to delete, you can pass their ID or tag to docker rmi:

List:

docker images -a

Remove:

docker rmi <Image>

Remove dangling images

Docker images consist of multiple layers. Dangling images are layers that have no relationship to any tagged images. They no longer serve a purpose and consume disk space. They can be located by adding the filter flag, -f with a value of dangling=true to the docker images command. When you're sure you want to delete them, you can use the docker images purge command:

Note: If you build an image without tagging it, the image will appear on the list of dangling images because it has no association with a tagged image. You can avoid this situation by providing a tag when you build, and you can retroactively tag an images with the docker tag command.

List:

docker images -f dangling=true

Remove:

docker images purge

Removing images according to a pattern

You can find all the images that match a pattern using a combination of docker images and grep. Once you're satisfied, you can delete them by using awk to pass the IDs to docker rmi. Note that these utilities are not supplied by Docker and are not necessarily available on all systems:

List:

docker images -a | grep "pattern"

Remove:

docker images -a | grep "pattern" | awk '{print $3}' | xargs docker rmi

Remove all images

All the Docker images on a system can be listed by adding -a to the docker images command. Once you're sure you want to delete them all, you can add the -q flag to pass the Image ID to docker rmi:

List:

docker images -a

Remove single image

docker image remove <image id>

Remove all the images

docker rmi $(docker images -a -q)

Removing Containers --> Remove one or more specific containers

Use the docker ps command with the -a flag to locate the name or ID of the containers you want to remove:

List:

docker ps -a

Remove:

docker rm ID\_or\_Name ID\_or\_Name

Remove a container upon exit

If you know when you’re creating a container that you won’t want to keep it around once you’re done, you can run docker run --rm to automatically delete it when it exits.

Run and Remove:

docker run --rm image\_name

Remove all exited containers

You can locate containers using docker ps -a and filter them by their status: created, restarting, running, paused, or exited. To review the list of exited containers, use the -f flag to filter based on status. When you've verified you want to remove those containers, using -q to pass the IDs to the docker rm command.

List:

docker ps -a -f status=exited

Remove:

docker rm $(docker ps -a -f status=exited -q)

Remove containers using more than one filter

Docker filters can be combined by repeating the filter flag with an additional value. This results in a list of containers that meet either condition. For example, if you want to delete all containers marked as either Created (a state which can result when you run a container with an invalid command) or Exited, you can use two filters:

List:

docker ps -a -f status=exited -f status=created

Remove:

docker rm $(docker ps -a -f status=exited -f status=created -q)

Remove containers according to a pattern

You can find all the containers that match a pattern using a combination of docker ps and grep. When you're satisfied that you have the list you want to delete, you can use awk and xargs to supply the ID to docker rmi. Note that these utilities are not supplied by Docker and not necessarily available on all systems:

List:

docker ps -a | grep "pattern”

Remove:

docker ps -a | grep "pattern" | awk '{print $3}' | xargs docker rmi

Stop and remove all containers

You can review the containers on your system with docker ps. Adding the -a flag will show all containers. When you're sure you want to delete them, you can add the -q flag to supply the IDs to the docker stop and docker rm commands:

List:

docker ps -a

Remove:

docker stop $(docker ps -a -q)

docker rm $(docker ps -a -q)

Remove a container and its volume

If you created an unnamed volume, it can be deleted at the same time as the container with the -v flag. Note that this only works with unnamed volumes. When the container is successfully removed, its ID is displayed. Note that no reference is made to the removal of the volume. If it is unnamed, it is silently removed from the system. If it is named, it silently stays present.

Remove:

docker rm -v container\_name

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

How to Build Docker Images with DockerFile

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Docker container images can be built in three ways,

1. Commit

2. Dockerfile

3. Compose

we will build Docker images using Dockerfile as per our requirements and then upload them on to Docker Hub Registry or Private registry.

Docker Keywords:

FROM:

======

The From keyword tells the Docker to use this image as Base Image for building the Docker container image. If the Base image is not found locally, Docker will fetch the image either from Docker Hub or Private Registry server.

Docker file must have FROM on the first line of instruction; you can use FROM multiple times within a single Dockerfile to create multiple images.

Example,

FROM centos:7

LABEL:

======

Use LABEL keyword to have your information like “Project name, Vendor, Version, Release date, Licensing. Maintainer, etc..” recorded onto Docker images.

Example,

LABEL project="CloudNLoud openshift project"

LABEL maintainer "vijay@cloudnloud.com"

RUN:

======

The RUN instruction is used to execute commands on the Docker image (centos:7) and commit the results. For simple understanding, use this RUN to install packages, run shell commands, and almost any commands you use in an operating system.

Example,

RUN yum makecache

RUN yum install -y httpd

RUN echo "This Page Designed by ITzGeek, for Docker Build"

CMD:

======

The CMD is used to run the application installed on your Docker image. For Ex, if you want to run a Apache server when a container is launched from a Docker image; you should add the following line.

Note: JSON format is used when you want to run a “command” without a shell.

Example,

CMD ["apache2","-DFOREGROUND"]

To run an application with a shell and get a bash prompt,

CMD service httpd restart && /bin/bash

PS: If you have more than one CMD then only the last CMD will take effect.

ENTRYPOINT:

===========

The ENTRYPOINT is also similar to CMD.

Example,

ENTRYPOINT service httpd restart && /bin/bash

You can also use ENTRYPOINT and CMD together. When combining ENTRYPOINT and CMD, ENTRYPOINT will take arguments from CMD. To start the httpd service when a container is launched from a Docker image, you need to add the following line.

ENTRYPOINT [ "/usr/sbin/httpd" ]

CMD ["-D", "FOREGROUND"]

EXPOSE:

======

The EXPOSE instruction will be used to expose the application (httpd) port from a container.

Example,

EXPOSE 80

ENV:

======

Use ENV keyword to set an environmental variable or shell variable for your application. If you want to set a JAVA variable, you can add the following line.

Example,

ENV JAVA\_HOME=/usr/jdk1.8.0\_05

ADD Or COPY:

============

The function of ADD or COPY are similar, COPY is preferred. COPY helps you to copy the local files into the container. ADD has some advanced features such as local file auto-tar extraction to the image and remote download support.

Example,

COPY requirements.txt /tmp/

ADD http://example.com/requirements.txt /tmp

VOLUME:

======

With VOLUME, you can create a mount point inside the container and make it available to native hosts or other containers. “VOLUME /data” creates “/data” mount point inside the container, the files inside “/data” will be persistent and same can be attached to an external host using -v or –volumes-from when launching a container from Docker image.

VOLUME /data

USER:

The USER sets the username or UID for RUN, CMD and ENTRYPOINT keywords.

Example,

USER root

COPY test.txt /

The above example will make the owner of the copied file as root.

Find more Keywords here. (https://docs.docker.com/engine/reference/builder/)

1. Creating DockerFile:

# mkdir /mydocker ; cd /mydocker

# cat Dockerfile

FROM centos

LABEL project="CloudNLoud openshift project"

LABEL maintainer "vijay@cloudnloud.com"

RUN yum -y install httpd

EXPOSE 82

VOLUME /var/www/html

ENTRYPOINT [ "/usr/sbin/httpd" ]

CMD ["-D", "FOREGROUND"]

2. Building Docker Image

run from the directory where docker file is present

# docker build -t mycentos:httpdv1.0 .

Sending build context to Docker daemon 2.048kB

Step 1/8 : FROM centos

latest: Pulling from library/centos

a02a4930cb5d: Pull complete

Digest: sha256:184e5f35598e333bfa7de10d8fb1cebb5ee4df5bc0f970bf2b1e7c7345136426

Status: Downloaded newer image for centos:latest

---> 1e1148e4cc2c

Step 2/8 : LABEL project="CloudNLoud openshift project"

---> Running in 0e01047a6645

Removing intermediate container 0e01047a6645

---> c1a687ef99e1

Step 3/8 : LABEL maintainer "vijay@cloudnloud.com"

---> Running in 9922d0e9aa15

Removing intermediate container 9922d0e9aa15

---> 8f471f85c2e9

Step 4/8 : RUN yum -y install httpd

---> Running in 85405003d3f0

Loaded plugins: fastestmirror, ovl

Determining fastest mirrors

\* base: repo1.ash.innoscale.net

\* extras: repo1.ash.innoscale.net

\* updates: mirror.net.cen.ct.gov

Resolving Dependencies

--> Running transaction check

---> Package httpd.x86\_64 0:2.4.6-88.el7.centos will be installed

--> Processing Dependency: httpd-tools = 2.4.6-88.el7.centos for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: system-logos >= 7.92.1-1 for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: /etc/mime.types for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: libaprutil-1.so.0()(64bit) for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: libapr-1.so.0()(64bit) for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Running transaction check

---> Package apr.x86\_64 0:1.4.8-3.el7\_4.1 will be installed

---> Package apr-util.x86\_64 0:1.5.2-6.el7 will be installed

---> Package centos-logos.noarch 0:70.0.6-3.el7.centos will be installed

---> Package httpd-tools.x86\_64 0:2.4.6-88.el7.centos will be installed

---> Package mailcap.noarch 0:2.1.41-2.el7 will be installed

--> Finished Dependency Resolution

Dependencies Resolved

================================================================================

Package Arch Version Repository Size

================================================================================

Installing:

httpd x86\_64 2.4.6-88.el7.centos base 2.7 M

Installing for dependencies:

apr x86\_64 1.4.8-3.el7\_4.1 base 103 k

apr-util x86\_64 1.5.2-6.el7 base 92 k

centos-logos noarch 70.0.6-3.el7.centos base 21 M

httpd-tools x86\_64 2.4.6-88.el7.centos base 90 k

mailcap noarch 2.1.41-2.el7 base 31 k

Transaction Summary

================================================================================

Install 1 Package (+5 Dependent packages)

Total download size: 24 M

Installed size: 31 M

Downloading packages:

warning: /var/cache/yum/x86\_64/7/base/packages/apr-util-1.5.2-6.el7.x86\_64.rpm: Header V3 RSA/SHA256 Signature, key ID f4a80eb5: NOKEY

Public key for apr-util-1.5.2-6.el7.x86\_64.rpm is not installed

Importing GPG key 0xF4A80EB5:

Userid : "CentOS-7 Key (CentOS 7 Official Signing Key) <security@centos.org>"

Fingerprint: 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5

Package : centos-release-7-6.1810.2.el7.centos.x86\_64 (@CentOS)

From : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

--------------------------------------------------------------------------------

Total 42 MB/s | 24 MB 00:00

Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : apr-1.4.8-3.el7\_4.1.x86\_64 1/6

Installing : apr-util-1.5.2-6.el7.x86\_64 2/6

Installing : httpd-tools-2.4.6-88.el7.centos.x86\_64 3/6

Installing : centos-logos-70.0.6-3.el7.centos.noarch 4/6

Installing : mailcap-2.1.41-2.el7.noarch 5/6

Installing : httpd-2.4.6-88.el7.centos.x86\_64 6/6

Verifying : mailcap-2.1.41-2.el7.noarch 1/6

Verifying : apr-util-1.5.2-6.el7.x86\_64 2/6

Verifying : httpd-tools-2.4.6-88.el7.centos.x86\_64 3/6

Verifying : httpd-2.4.6-88.el7.centos.x86\_64 4/6

Verifying : apr-1.4.8-3.el7\_4.1.x86\_64 5/6

Verifying : centos-logos-70.0.6-3.el7.centos.noarch 6/6

Installed:

httpd.x86\_64 0:2.4.6-88.el7.centos

Dependency Installed:

apr.x86\_64 0:1.4.8-3.el7\_4.1

apr-util.x86\_64 0:1.5.2-6.el7

centos-logos.noarch 0:70.0.6-3.el7.centos

httpd-tools.x86\_64 0:2.4.6-88.el7.centos

mailcap.noarch 0:2.1.41-2.el7

Complete!

Removing intermediate container 85405003d3f0

---> b3c850ce53c5

Step 5/8 : EXPOSE 82

---> Running in 8775b2989374

Removing intermediate container 8775b2989374

---> 14c9c9e3d56d

Step 6/8 : VOLUME /var/www/html

---> Running in ea43e37fa702

Removing intermediate container ea43e37fa702

---> d3d0b8866c83

Step 7/8 : ENTRYPOINT [ "/usr/sbin/httpd" ]

---> Running in 7f45ada821db

Removing intermediate container 7f45ada821db

---> 08edd4050fec

Step 8/8 : CMD ["-D", "FOREGROUND"]

---> Running in e048d619c690

Removing intermediate container e048d619c690

---> c6d05e5e6f2b

Successfully built c6d05e5e6f2b

Successfully tagged mycentos:httpdv1.0

[root@shanker-master mydocker]#

Output:

REPOSITORY TAG IMAGE ID CREATED SIZE

mycentos httpdv1.0 c6d05e5e6f2b 2 minutes ago 305MB ===> this is the images we have just created

httpd <none> ef1dc54703e2 12 days ago 132MB

centos latest 1e1148e4cc2c 5 weeks ago 202MB

3. Launching Container

Create a directory called “/data” on Docker host, “/data” directory will hold the files of Apache DocumentRoot “/var/www/html”

# mkdir /data

Launch a container using the following command.

# docker run -td -p 82:80 -v /data:/var/www/html --name=shanker mycentos:httpdv1.0

bf16b37df3faa33710258b6d32995467c3a4109ea8888b0a6b57ac42d143ce88

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bf16b37df3fa mycentos:httpdv1.0 "/usr/sbin/httpd -D …" 6 seconds ago Up 5 seconds 82/tcp, 0.0.0.0:82->80/tcp shanker

# echo "This is an example image for openshift demo in CNL" > /data/index.html

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bf16b37df3fa mycentos:httpdv1.0 "/usr/sbin/httpd -D …" 3 minutes ago Up 3 minutes 82/tcp, 0.0.0.0:82->80/tcp shanker

Volumes in docker

We have two types empheral and persistent

Persistent : permanent --🡪 volumes,bind mount

Volumes managed by docker itself

Volumes: (/var/lib/docker) paths where volume will be stored

Docker volume create vol1

Docker inspect vol1

Docker container run -it –name c1 vol1:/vol centos

Docker container run -it –name=c4 –volume-from c2 alpine(sharing data between container)

Docker container run -it –name c1 vol1:/vol:ro centos (sharing in read only mode)

Docker container run -it -name –mount source=/vol1,target=vol2 alpine

Bindmount :

Attaching the volume from a host machine to container

Docker container run -it –name c1 /dir1:/dir1 centos

Steps for configuring Swarn Cluster:

1. Update the following lines in /etc/hosts file on all the servers (ALL HOSTS)

10.142.0.2 shanker-master.us-east1-b.c.folkloric-grid-221412.internal shanker-master – It will act as manager who will manage Docker engine or hosts or worker node and it will work as Docker engine as well.

10.142.0.7 shanker-client.us-east1-b.c.folkloric-grid-221412.internal shanker-client – it will acts Docker engine or Worker Node

10.142.0.8 shanker-client2.us-east1-b.c.folkloric-grid-221412.internal shanker-client2 – it will acts Docker engine or Worker Node

2. Install Docker Engine on all the hosts (ALL HOSTS)

yum install yum-utils device-mapper-persistent-data lvm2 -y #Install the required dependencies:

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo #Add the stable Docker repository by typing:

yum install docker-ce #Now that we have Docker repository enabled, we can install the latest version of Docker CE (Community Edition) using yum by typing:

systemctl start docker #Once the Docker package is installed, we start the Docker daemon with:

systemctl status docker #To verify that the Docker service is running type:

systemctl enable docker #Enable the Docker service to be automatically started at boot time:

3.FirwallD in CentOS 7 can conflict with Docker; it is recommended to disable the service. (ALL HOSTS)

systemctl stop firewalld;systemctl disable firewalld

When firewalld is started or restarted it will remove the DOCKER chain from iptables, it prevents Docker from working properly.

If you still want to use systemd, firewalld is must be started before Docker service. In case if you start or restart firewalld after Docker, you will have to restart the Docker daemon

4. Allowing Non-root access (ALL HOSTS) -- If required

As you can see in my command, for CentOS, I had to run Docker as a root user. To avoid this, you can follow below procedure to allow non-root users to run Docker containers.

Create a group called docker if it does not exist, run the following commands with root privileges.

# groupadd docker

Add a user that is to be a part of docker group, replace “shanker” with your own username.

# useradd shanker

Add a user to docker group.

# usermod -aG docker shanker

5. Firewall ---- (ALL HOSTS) if required

You would need to open following ports on the firewall for a swarm cluster to work properly.

CentOS / Fedora:

firewall-cmd --permanent --add-port=7946/tcp

firewall-cmd --permanent --add-port=4789/udp

firewall-cmd --permanent --add-port=7946/udp

firewall-cmd --permanent --add-port=2376/tcp

firewall-cmd --permanent --add-port=2377/tcp

firewall-cmd --permanent --add-port=80/tcp ## We are Testing Docker Swarm with WebService

firewall-cmd --reload

6. Initialize the swarm or cluster using ‘docker swarm init’ command (Only On Master)

Run the below command from the manager node(dkmanager) to initialize the cluster.

docker swarm init --advertise-addr 10.142.0.2

Output:

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-3vutvzatjjbx76hsdzeyuc20lopv9hu0tt0m9gde4sjzjiiqk5-ahtqr9his0wt3y4rpuok6tg5t 10.142.0.2:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

7. Add Worker Nodes to swarm or cluster ( Only on client )

[root@shanker-client ~]# docker swarm join --token SWMTKN-1-3vutvzatjjbx76hsdzeyuc20lopv9hu0tt0m9gde4sjzjiiqk5-ahtqr9his0wt3y4rpuok6tg5t 10.142.0.2:2377

This node joined a swarm as a worker.

[root@shanker-client2 ~]# docker swarm join --token SWMTKN-1-3vutvzatjjbx76hsdzeyuc20lopv9hu0tt0m9gde4sjzjiiqk5-ahtqr9his0wt3y4rpuok6tg5t 10.142.0.2:2377

This node joined a swarm as a worker.

8.

[root@shanker-master ~]# docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

yhbos141zyns4l81z1zdpp49i shanker-client Ready Active 18.09.1

mazfq2pler506gdv6b5e17s13 shanker-client2 Ready Active 18.09.1

zkfp9j3tukxmkmozp8xzbjgvk \* shanker-master Ready Active Leader 18.09.1

[root@shanker-master ~]# docker info

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 18.09.1

Storage Driver: overlay2

Backing Filesystem: xfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog

Swarm: active =====> SWARN service is active

NodeID: zkfp9j3tukxmkmozp8xzbjgvk

Is Manager: true

ClusterID: cugseksnitcxhy09tajokbki7

Managers: 1

Nodes: 3

Default Address Pool: 10.0.0.0/8

SubnetSize: 24

Orchestration:

Task History Retention Limit: 5

Raft:

Snapshot Interval: 10000

Number of Old Snapshots to Retain: 0

Heartbeat Tick: 1

Election Tick: 10

Dispatcher:

Heartbeat Period: 5 seconds

CA Configuration:

Expiry Duration: 3 months

Force Rotate: 0

Autolock Managers: false

Root Rotation In Progress: false

Node Address: 10.142.0.2

Manager Addresses:

10.142.0.2:2377

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9754871865f7fe2f4e74d43e2fc7ccd237edcbce

runc version: 96ec2177ae841256168fcf76954f7177af9446eb

init version: fec3683

Security Options:

seccomp

Profile: default

Kernel Version: 3.10.0-957.1.3.el7.x86\_64

Operating System: CentOS Linux 7 (Core)

OSType: linux

Architecture: x86\_64

CPUs: 1

Total Memory: 1.795GiB

Name: shanker-master

ID: 7K4D:CWI6:FCDI:LBKC:HQY5:7VWX:AKGF:G53G:SGVK:AG6H:RJ4C:GO6N

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Labels:

Experimental: false

Insecure Registries:

127.0.0.0/8

Live Restore Enabled: false

Product License: Community EngineLaunching service in Docker Swarm mode

9. Launching service in Docker Swarm mode

# docker service create -p 80:80 --name webserver --replicas 3 httpd or ngnix

o11m8j6dz0t00tcq6xqpivftu

overall progress: 3 out of 3 tasks

1/3: running [==================================================>]

2/3: running [==================================================>]

3/3: running [==================================================>]

verify: Service converged

# docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

o11m8j6dz0t0 webserver replicated 3/3 httpd:latest \*:80->80/tcp

yzslwcx14ijs webserver1 replicated 2/2 httpd:latest \*:85->80/tcp

# docker service ps webserver

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

ulh9j6el05ix webserver.1 httpd:latest shanker-client2 Running Running 3 minutes ago

kqyizvrc6j9l webserver.2 httpd:latest shanker-client Running Running 3 minutes ago

nega6gspgc3v webserver.3 httpd:latest shanker-master Running Running 3 minutes ago

check the URL http:<<ip>>:80

10. Now Test Container Self Healing

Container self healing is the important feature of docker swarm mode. As the name suggest if anything goes wrong with container , manager will make sure at least 5 container must be running for the service “webserver”. Let’s remove the container from workernode2 and see whether a new container is launched or not.

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8e6078c9305b httpd:latest "httpd-foreground" 4 minutes ago Up 4 minutes 80/tcp webserver1.2.jkyewf67966yab8dh64mh4xh7

4815d5313246 httpd:latest "httpd-foreground" 5 minutes ago Up 5 minutes 80/tcp webserver.3.nega6gspgc3vfoonyg6nmi4sh

# docker rm 8e6078c9305b -f ( you can try removing in cient also)

8e6078c9305b

# docker service ps webserver

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

ulh9j6el05ix webserver.1 httpd:latest shanker-client2 Running Running 7 minutes ago

kqyizvrc6j9l webserver.2 httpd:latest shanker-client Running Running 7 minutes ago

gmhaas2xuhq5 webserver.3 httpd:latest shanker-master Running Running about a minute ago

nega6gspgc3v \\_ webserver.3 httpd:latest shanker-master Shutdown Failed about a minute ago "task: non-zero exit (137)"

As per above output we can see a new container is launched on dkmanager node because one of the container on workernode2 is removed

11. Scale up and Scale down containers associated to a Service

In Docker swarm mode we can scale up and scale down containers or tasks. Let’s scale up the containers to 7 for the service ‘webserver‘

# docker service scale webserver=7

webserver scaled to 7

overall progress: 7 out of 7 tasks

1/7: running [==================================================>]

2/7: running [==================================================>]

3/7: running [==================================================>]

4/7: running [==================================================>]

5/7: running [==================================================>]

6/7: running [==================================================>]

7/7: running [==================================================>]

verify: Service converged

# docker service ps webserver

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

ulh9j6el05ix webserver.1 httpd:latest shanker-client2 Running Running 17 minutes ago

kk6c47gisl2c webserver.2 httpd:latest shanker-client Running Running 2 minutes ago

kqyizvrc6j9l \\_ webserver.2 httpd:latest shanker-client Shutdown Failed 2 minutes ago "task: non-zero exit (137)"

ryvj0bi1661f webserver.3 httpd:latest shanker-master Running Running 3 minutes ago

gmhaas2xuhq5 \\_ webserver.3 httpd:latest shanker-master Shutdown Failed 3 minutes ago "task: non-zero exit (137)"

nega6gspgc3v \\_ webserver.3 httpd:latest shanker-master Shutdown Failed 10 minutes ago "task: non-zero exit (137)"

k3kpuc11l7nw webserver.4 httpd:latest shanker-client Running Running about a minute ago

e3sep8ly5kpa webserver.5 httpd:latest shanker-client Running Running about a minute ago

sm28l98jeqjo webserver.6 httpd:latest shanker-client2 Running Running about a minute ago

lkqar5ll8dnz webserver.7 httpd:latest shanker-master Running Running about a minute ago

Let’s Scale down container to 4 for the service webserver

# docker service scale webserver=2

webserver scaled to 2

overall progress: 2 out of 2 tasks

1/2: task: non-zero exit (137)

2/2: running [==================================================>]

verify: Service converged

[root@shanker-master ~]# docker service ps webserver

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

ulh9j6el05ix webserver.1 httpd:latest shanker-client2 Running Running 19 minutes ago

kk6c47gisl2c webserver.2 httpd:latest shanker-client Running Running 4 minutes ago

kqyizvrc6j9l \\_ webserver.2 httpd:latest shanker-client Shutdown Failed 4 minutes ago "task: non-zero exit (137)"

gmhaas2xuhq5 webserver.3 httpd:latest shanker-master Shutdown Failed 5 minutes ago "task: non-zero exit (137)"

nega6gspgc3v \\_ webserver.3 httpd:latest shanker-master Shutdown Failed 12 minutes ago "task: non-zero exit (137)"

13. docker swarm –help

Docker Networks:

Overview

=========

One of the reasons Docker containers and services are so powerful is that you can connect them together, or connect them to non-Docker workloads. Docker containers and services do not even need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not. Whether your Docker hosts run Linux, Windows, or a mix of the two, you can use Docker to manage them in a platform-agnostic way.

This topic defines some basic Docker networking concepts and prepares you to design and deploy your applications to take full advantage of these capabilities.

Most of this content applies to all Docker installations. However, a few advanced features are only available to Docker EE customers.

Scope of this topic

This topic does not go into OS-specific details about how Docker networks work, so you will not find information about how Docker manipulates iptables rules on Linux or how it manipulates routing rules on Windows servers, and you will not find detailed information about how Docker forms and encapsulates packets or handles encryption.

See Docker and iptables and Docker Reference Architecture: Designing Scalable, Portable Docker Container Networks for a much greater depth of technical detail.

Network drivers

===============

Docker’s networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

1. bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate. See bridge networks.

2. host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly. host is only available for swarm services on Docker 17.06 and higher. See use the host network.

3. overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.

4. macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host’s network stack. See Macvlan networks.

5. none: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services. See disable container networking.

6. Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors. See the vendor’s documentation for installing and using a given network plugin.

Network driver summary

======================

1. User-defined bridge networks are best when you need multiple containers to communicate on the same Docker host.

2. Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.

3. Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.

4. Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.

5. Third-party network plugins allow you to integrate Docker with specialized network stacks.

Docker EE networking features

The following two features are only possible when using Docker EE and managing your Docker services using Universal Control Plane (UCP):

The HTTP routing mesh allows you to share the same network IP address and port among multiple services. UCP routes the traffic to the appropriate service using the combination of hostname and port, as requested from the client.

Session stickiness allows you to specify information in the HTTP header which UCP uses to route subsequent requests to the same service task, for applications which require stateful sessions.

Networking tutorials ( https://docs.docker.com/network/)

====================

Now that you understand the basics about Docker networks, deepen your understanding using the following tutorials:

Standalone networking tutorial

Host networking tutorial

Overlay networking tutorial

Macvlan networking tutorial

Networking with standalone containers

=======================================

Two types

1. Use the default bridge network demonstrates how to use the default bridge network that Docker sets up for you automatically. This network is not the best choice for production systems.

2. Use user-defined bridge networks shows how to create and use your own custom bridge networks, to connect containers running on the same Docker host. This is recommended for standalone containers running in production.

Although overlay networks are generally used for swarm services, Docker 17.06 and higher allow you to use an overlay network for standalone containers. That’s covered as part of the tutorial on using overlay networks.

Use the default bridge network

===============================

1. #docker network ls

NETWORK ID NAME DRIVER SCOPE

86b57085e456 bridge bridge local

bac70a3565cd docker\_gwbridge bridge local

fba0e298a921 host host local

hlwb2ttuecag ingress overlay swarm

a680cc3b683e none null local

2. Start 2 alpine projects

# docker run -dit --name alpine1 alpine ash

# docker run -dit --name alpine2 alpine ash

Check that both containers are actually started:

# docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

602dbf1edc81 alpine "ash" 4 seconds ago Up 3 seconds alpine2

da33b7aa74b0 alpine "ash" 17 seconds ago Up 16 seconds alpine1

Inspect the bridge network to see what containers are connected to it.

# docker network inspect bridge

{

"Name": "alpine1",

"EndpointID": "46c044a645d6afc42ddd7857d19e9dcfb89ad790afb5c239a35ac0af5e8a5bc5",

"MacAddress": "02:42:ac:11:00:02",

"IPv4Address": "172.17.0.2/16",

"IPv6Address": ""

}

# docker attach alpine1 ===> Login to check the ip address

# ip addr show

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue state UP

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::42:acff:fe11:2/64 scope link

valid\_lft forever preferred\_lft forever

3. Stop and remove both containers.

# docker container stop alpine1 alpine2

# docker container rm alpine1 alpine2

Create the alpine-net network. You do not need the --driver bridge flag since it’s the default, but this example shows how to specify it.

[root@shanker-master data]# docker network ls

NETWORK ID NAME DRIVER SCOPE

86b57085e456 bridge bridge local

bac70a3565cd docker\_gwbridge bridge local

fba0e298a921 host host local

hlwb2ttuecag ingress overlay swarm

a680cc3b683e none null local

[root@shanker-master data]# docker network create --driver bridge alpine-net

b18eed4ee77eddf46c8de92a6d337b1da3dbfa4b19e8a5f7e0308a395f60382c

[root@shanker-master data]# docker network ls

NETWORK ID NAME DRIVER SCOPE

b18eed4ee77e alpine-net bridge local

86b57085e456 bridge bridge local

bac70a3565cd docker\_gwbridge bridge local

fba0e298a921 host host local

hlwb2ttuecag ingress overlay swarm

a680cc3b683e none null local

# docker network inspect alpine-net

[

{

"Name": "alpine-net",

"Id": "b18eed4ee77eddf46c8de92a6d337b1da3dbfa4b19e8a5f7e0308a395f60382c",

"Created": "2019-01-10T20:48:38.562541434Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.19.0.0/16",

"Gateway": "172.19.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"29576d21ff6b784caa4c2edde7d8a0fd0681ba83ad8b0a72534e6ac876b31a22": {

"Name": "alpine2",

"EndpointID": "3d82d67c53e0f16c2737935bfd36e0f7d3aa80bc8fae7cd796153f14140efd5b",

"MacAddress": "02:42:ac:13:00:03",

"IPv4Address": "172.19.0.3/16",

"IPv6Address": ""

},

"fb92baa1152c35325795766e46d98188af663b06e030a80b074e59d321a8f864": {

"Name": "alpine1",

"EndpointID": "ffeb9b6e74f3fd8f58699d0084a7766cb919d9998f2b991635174be9d989256b",

"MacAddress": "02:42:ac:13:00:02",

"IPv4Address": "172.19.0.2/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {}

}

]

Notice that this network’s gateway is 172.19.0.1, as opposed to the default bridge network, whose gateway is 172.17.0.1. The exact IP address may be different on your system.

Create your 3 containers. Notice the --network flags. You can only connect to one network during the docker run command, so you need to use docker network connect afterward to connect alpine4 to the bridge network as well.

# docker run -dit --name alpine1 --network alpine-net alpine ash

Unable to find image 'alpine:latest' locally

latest: Pulling from library/alpine

cd784148e348: Pull complete

Digest: sha256:46e71df1e5191ab8b8034c5189e325258ec44ea739bba1e5645cff83c9048ff1

Status: Downloaded newer image for alpine:latest

fb92baa1152c35325795766e46d98188af663b06e030a80b074e59d321a8f864

# docker run -dit --name alpine2 --network alpine-net alpine ash

29576d21ff6b784caa4c2edde7d8a0fd0681ba83ad8b0a72534e6ac876b31a22

# docker run -dit --name alpine3 alpine ash

c37522480e299ebf0415bd031c54aed280f208094643cb6e8f662524d52d353b

# docker network connect bridge alpine2

# docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

29576d21ff6b alpine "ash" About a minute ago Up About a minute alpine2

c37522480e29 alpine "ash" About a minute ago Up About a minute alpine3

fb92baa1152c alpine "ash" 2 minutes ago Up 2 minutes alpine1

bf16b37df3fa mycentos:httpdv1.0 "/usr/sbin/httpd -D …" 35 minutes ago Up 35 minutes 82/tcp, 0.0.0.0:82->80/tcp shanker

13e28f03b658 httpd:latest "httpd-foreground" About an hour ago Up About an hour 80/tcp webserver1.2.jjoj4nn0el34l96qt48efbgcm

# docker network inspect bridge

# docker network inspect alpine-net

4. Stop and remove all containers and the alpine-net network.

# docker container stop alpine1 alpine2 alpine3

# docker container rm alpine1 alpine2 alpine3

# docker network rm alpine-net

Preparing===>Ready ====> Starting ====> Running

Networking with overlay networks ( Swarm Networks)

==================================================

1. Use the default overlay network:- demonstrates how to use the default overlay network that Docker sets up for you automatically when you initialize or join a swarm. This network is not the best choice for production systems.

2. Use user-defined overlay networks:- shows how to create and use your own custom overlay networks, to connect services. This is recommended for services running in production.

3. Use an overlay network for standalone containers:- shows how to communicate between standalone containers on different Docker daemons using an overlay network.

4. Communicate between a container and a swarm service:- sets up communication between a standalone container and a swarm service, using an attachable overlay network. This is supported in Docker 17.06 and higher.

Use the default overlay network

===============================

1. Please follow the steps for docker swarm setup

2.

[root@shanker-master data]# docker node ls --filter role=manager

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

zkfp9j3tukxmkmozp8xzbjgvk \* shanker-master Ready Active Leader 18.09.1

[root@shanker-master data]# docker node ls --filter role=worker

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

yhbos141zyns4l81z1zdpp49i shanker-client Ready Active 18.09.1

mazfq2pler506gdv6b5e17s13 shanker-client2 Ready Active 18.09.1

[root@shanker-master data]#

[root@shanker-master data]# docker network ls

NETWORK ID NAME DRIVER SCOPE

86b57085e456 bridge bridge local

bac70a3565cd docker\_gwbridge bridge local

fba0e298a921 host host local

hlwb2ttuecag ingress overlay swarm

a680cc3b683e none null local

3.Create the services ( To be done on Manager/Master Node )

[root@shanker-master data]# docker network create -d overlay nginx-net

niibbf95yqbqntrltl0poqzr5

[root@shanker-master data]# docker network ls

NETWORK ID NAME DRIVER SCOPE

86b57085e456 bridge bridge local

bac70a3565cd docker\_gwbridge bridge local

fba0e298a921 host host local

hlwb2ttuecag ingress overlay swarm

niibbf95yqbq nginx-net overlay swarm

a680cc3b683e none null local

On master, create a 5-replica Nginx service connected to nginx-net. The service will publish port 8080 to the outside world. All of the service task containers can communicate with each other without opening any ports.

Note: Services can only be created on a manager.

# docker service create --name my-nginx --publish target=80,published=8080 --replicas=5 --network nginx-net nginx

c4l6iecnodf8961hk0zp5p2ou

overall progress: 5 out of 5 tasks

1/5: running [==================================================>]

2/5: running [==================================================>]

3/5: running [==================================================>]

4/5: running [==================================================>]

5/5: running [==================================================>]

verify: Service converged

4. Inspect the network and service

# docker inspect network nginx-net

# docker service inspect my-nginx

5. Create a new network nginx-net-2, then update the service to use this network instead of nginx-net

[root@shanker-master data]# docker network create -d overlay nginx-net-2

ndui47nzuoyneslhs311i6u9x

[root@shanker-master data]# docker service update \

> --network-add nginx-net-2 \

> --network-rm nginx-net \

> my-nginx

my-nginx

overall progress: 5 out of 5 tasks

1/5: running [==================================================>]

2/5: running [==================================================>]

3/5: running [==================================================>]

4/5: running [==================================================>]

5/5: running [==================================================>]

verify: Service converged

6. Run "docker service ls" to verify that the service has been updated and all tasks have been redeployed. Run docker network inspect nginx-net to verify that no containers are connected to it. Run the same command for nginx-net-2 and notice that all the service task containers are connected to it.

Note: Even though overlay networks are automatically created on swarm worker nodes as needed, they are not automatically removed.

Clean up the service and the networks. From manager, run the following commands. The manager will direct the workers to remove the networks automatically.

root@shanker-master data]# docker service rm my-nginx webserver webserver1

my-nginx

webserver

webserver1

[root@shanker-master data]# docker network rm nginx-net nginx-net-2

nginx-net

nginx-net-2

Method 2: Use a user-defined overlay network

============================================

1. Create the user-defined overlay network.

[root@shanker-master data]# docker network create -d overlay my-overlay

x77aq2gwwt7k1yu0z2xi62ynt

2. Start a service using the overlay network and publishing port 80 to port 8080 on the Docker host.

[root@shanker-master data]# docker service create --name my-nginx --network my-overlay --replicas 1 --publish published=8080,target=80 nginx:latest

nho9alp9swenvszknf3cd0fq1

overall progress: 1 out of 1 tasks

1/1: running [==================================================>]

verify: Service converged

3. Run docker network inspect my-overlay and verify that the my-nginx service task is connected to it, by looking at the Containers section.

[root@shanker-master data]# docker network inspect my-overlay

[

{

"Name": "my-overlay",

"Id": "x77aq2gwwt7k1yu0z2xi62ynt",

"Created": "2019-01-10T21:23:11.782013538Z",

"Scope": "swarm",

"Driver": "overlay",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "10.0.2.0/24",

"Gateway": "10.0.2.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"fc109c6a6bc06a91b5999d79230c02bd8c9f96935ab6b23a181846882577408a": {

"Name": "my-nginx.1.nyfd9px11gaqlbs6t0milnc24",

"EndpointID": "dba5077ea7b88508ddac5398abfccc8fda44a008ea737f1977e784429833cf38",

"MacAddress": "02:42:0a:00:02:03",

"IPv4Address": "10.0.2.3/24",

"IPv6Address": ""

},

"lb-my-overlay": {

"Name": "my-overlay-endpoint",

"EndpointID": "fc1e0526602228e180228dc6e30565f4d74fe61de698121bb3b94fafacd1aeed",

"MacAddress": "02:42:0a:00:02:04",

"IPv4Address": "10.0.2.4/24",

"IPv6Address": ""

}

},

"Options": {

"com.docker.network.driver.overlay.vxlanid\_list": "4099"

},

"Labels": {},

"Peers": [

{

"Name": "d1de11d43028",

"IP": "10.142.0.2"

}

]

}

]

4. Remove the service and network

[root@shanker-master data]# docker service rm my-nginx

my-nginx

[root@shanker-master data]# docker network rm my-overlay

my-overlay

[root@shanker-master data]#

KUBERBNETS

SINGLE NODE

1. Update the /etc/hosts ===> On both master and client

10.142.0.13 test-client1.us-east1-b.c.folkloric-grid-221412.internal test-client1 # Added by Google

10.142.0.12 test-master.us-east1-b.c.folkloric-grid-221412.internal test-master

2. modprobe br\_netfilter;echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables ===> On both master and client

3. swapoff -a ===> On both master and client

4. yum install yum-utils device-mapper-persistent-data lvm2 ===> On both master and client

5. yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo ===> On both master and client

6. yum install -y --setopt=obsoletes=0 docker-ce-17.03.2.ce-1.el7.centos docker-ce-selinux-17.03.0.ce-1.el7.centos ===> On both master and client

7. systemctl enable docker;systemctl start docker ===> On both master and client

8. docker version

Client:

Version: 17.03.2-ce

API version: 1.27

Go version: go1.7.5

Git commit: f5ec1e2

Built: Tue Jun 27 02:21:36 2017

OS/Arch: linux/amd64

Server:

Version: 17.03.2-ce

API version: 1.27 (minimum version 1.12)

Go version: go1.7.5

Git commit: f5ec1e2

Built: Tue Jun 27 02:21:36 2017

OS/Arch: linux/amd64

Experimental: false

9. cat <<EOF > /etc/yum.repos.d/kubernetes.repo ===> On both master and client

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kube\*

EOF

10. yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes ===> On both master and client

11. systemctl enable kubelet && systemctl start kubelet ===> On both master and client

12. cat <<EOF > /etc/sysctl.d/k8s.conf === only on master

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

13. kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10.142.0.12 ====> only on master

14. only on master

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

15. Only on Master

[root@test-master ~]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4de4c0d86c5cd2643275/Documentation/kube-flannel.yml

Output:

clusterrole.rbac.authorization.k8s.io/flannel created

clusterrolebinding.rbac.authorization.k8s.io/flannel created

serviceaccount/flannel created

configmap/kube-flannel-cfg created

daemonset.extensions/kube-flannel-ds-amd64 created

daemonset.extensions/kube-flannel-ds-arm64 created

daemonset.extensions/kube-flannel-ds-arm created

daemonset.extensions/kube-flannel-ds-ppc64le created

daemonset.extensions/kube-flannel-ds-s390x created

16. kubeadm join 10.142.0.12:6443 --token 4y13zr.3jjcn6egskm78xg8 --discovery-token-ca-cert-hash sha256:69f0f8640221684935bb47bbd9925aebde6d4086a525f7c3094759f03a4270ee ====> only on client

17. kubectl get pods --all-namespaces Only on Master

18.

[root@test-master ~]# kubectl create deployment --image nginx my-nginx Only on Master

deployment.apps/my-nginx created

[root@test-master ~]# kubectl get nodes Only on Master

NAME STATUS ROLES AGE VERSION

test-client1 Ready <none> 89s v1.13.1

test-master Ready master 5m51s v1.13.1

[root@test-master ~]# kubectl get pods Only on Master

NAME READY STATUS RESTARTS AGE

my-nginx-6cc48cd8db-flj4n 1/1 Running 0 11s

[root@test-master ~]#

