**Docker**

**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 application, ship them into containers which can be deployed anywhere.

It is an Open Source Project that automates the deployment of applications inside the S/W containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux. It is basically a tool that a packages in a “virtual container” so that it can be run on any Linux systems or distribution.

**>>When would I use Docker**

* Configuration and Simplification.
* Enhance Developer Productivity.
* Server Consolidation and Management.
* Application Isolation.
* Rapid Deployment.
* Build Management.

**Note –Docker offers you the ability to isolate your application, standardize your build and deployment process to create standard, repeatable process in your S/W in Infra.**

**>>Virtual Machines**

# It is an emulation of a specific computer system type.

# Virtualization S/W allows you to set up one operating system within another.

# Although they share same physical HW. The VM is isolated from that HW and has to communicate with it through something called hypervisor.

**>>Container**

It is an entirely isolated set of packages libraries and for application that are completely independent from its surroundings.

**>> Container Architecture (Docker)**

# Docker is a client-server application where both daemon and client can be run on the same system or you can connect Docker client with a remote Docker daemon.

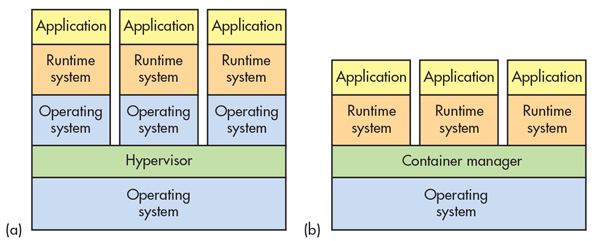
# Docker clients and daemons communicates via sockets or through a RESTful API.

# The main components of Docker are:

* Daemon
* Client
* Docker.io Registry

# Instead of virtualizing HW, containers rest on top of a single Linux instance.

# This allows Docker to leave behind a lot of the bloat associated with a full HW hypervisor.



1. Virtual Machine managed by hypervisor
2. Container System (Docker)

Some more details about the differences:

**>>Virtual Machine managed by hypervisor**

* The server is the physical server that is used to host multiple virtual machines.
* The Host OS is the base machine such as Linux or Windows.
* The Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.
* You would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.
* You would then host your applications on top of each Guest OS.

**>>Container System (Docker)**

* The server is the physical server that is used to host multiple virtual machines. So this layer remains the same.
* The Host OS is the base machine such as Linux or Windows. So this layer remains the same.
* Now comes the new generation which is the Docker engine. This is used to run the operating system which earlier used to be virtual machines as Docker containers.
* All of the Apps now run as Docker containers.

The clear advantage in this architecture is that you don’t need to have extra hardware for Guest OS. Everything works as Docker containers.

**>>Docker Hub**

It is a public registry/repository that is maintained by Docker Inc. containing a large number of images that you can download and use to build container.

* http://hub.docker.com

**>>Installation of Docker**

# yum install docker

# systemctl start docker (**Starting Docker service)**

# systemctl enable docker **(Enabling Docker at boot level)**

# docker –version **(Checking Docker version)**

Docker version 1.12.6, build 3e8e77d/1.12.6

**>>For checking Docker running properly, we can download a test image.**

# docker run hello-world

Unable to find image 'hello-world:latest' locally

Trying to pull repository docker.io/library/hello-world ...

latest: Pulling from docker.io/library/hello-world

ca4f61b1923c: Pull complete

Digest: sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1

Status: Downloaded newer image for docker.io/hello-world:latest

**Hello from Docker!**

**This message shows that your installation appears to be working correctly.**

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:

<https://docs.docker.com/engine/userguide/>

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

**Note: First It will go for searching in local library and if not found there, It will go for downloading from Docker Hub.**

In Docker, everything is based on Images. An image is a combination of a file system and parameters. Let’s take an example of the above command in Docker.

# docker run hello-world

* The Docker command is specific and tells the Docker program on the Operating System that something needs to be done.
* The run command is used to mention that we want to create an instance of an image, which is then called a container.
* Finally, "hello-world" represents the image from which the container is made.

**Note: /var/lib/docker – Docker root directory**

In Docker, everything is based on Images. An image is a combination of a file system and parameters. Let’s take an example of the following command in Docker.

# docker run hello-world

Now let’s look at how we can use the CentOS image available in Docker Hub to run CentOS on our Docker machine. We can do this by executing the following command on our machine −

# docker run centos –it /bin/bash

* Here, centos is the name of the image we want to download from Docker Hub and install on our machine.
* -it is used to mention that we want to run in interactive mode.
* /bin/bash is used to run the bash shell once CentOS is up and running.

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

From the above output, you can see that the server has oneimage: hello-world

Each image has the following attributes −

* TAG − this is used to logically tag images.
* Image ID − this is used to uniquely identify the image.
* Created − the number of days since the image was created.
* Virtual Size − the size of the image.

# docker images –q (Only show numeric IDs)

f2a91732366c

# docker images –a (Show all images)

# docker images –digests (show digests)

REPOSITORY TAG DIGEST IMAGE ID CREATED SIZE

docker.io/hello-world latest sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1 f2a91732366c 4 months ago 1.85 kB

**>>Few Docker Commands**

# docker version (Give version information of Docker)

# docker info (Gives all information about host and containers)

# docker ps (Shows the running containers)

# docker ps –a (Shows all containers running/stopped)

**>>Creating our first repo (Downloading Docker images)**

Images can be downloaded from Docker Hub using the Docker run command.

# docker run/pull image

* Image  (This is the name of the image which is used to run the container)

Pull will only download the image while run command will download and start your container if you use some arguments followed by commands.

# docker run –it centos:latest /bin/bash

Or

# docker pull centos:latest

**# docker pull centos:latest**

**Trying to pull repository docker.io/library/centos ...**

**latest: Pulling from docker.io/library/centos**

**5e35d10a3eba: Pull complete**

**Digest: sha256:dcbc4e5e7052ea2306eed59563da1fec09196f2ecacbe042acbdcd2b44b05270**

**Status: Downloaded newer image for docker.io/centos:latest**

It is visible in images now:

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

Below command will login to the container. If you exit from here. Then your container will stopped and you need to run it again.

# docker run -it docker.io/centos /bin/bash

[root@abd276797c81 /]# hostname

abd276797c81

[root@abd276797c81 /]#

**Note:**

**-i (interactive)**

**-t (to connect to tty)**

**>>Removing Docker Images**

Docker images can be removed using “rmi” as shown below:

# docker rmi ImageID

**>>Inspecting Docker Images**

# docker inspect Image-Name (Information about the image/container)

**# docker inspect centos**

**>>Docker Containers**

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

Container are instances of Docker images that can be run using Docker run command. The basic purpose of Docker is to run

**>>>Running a container**

To run a container in interactive mode, run below command:

# docker run -it centos /bin/bash

[root@d1b4678b0552 /]#hostname

d1b4678b0552

**>>>Listing of containers**

# docker ps (It will list running container only)

# docker ps –a (It will list running & stopped container)

With below command, you can see all the commands that were run with an image via a container.

# docker history Image-name

# docker history centos

IMAGE CREATED CREATED BY SIZE COMMENT

2d194b392dd1 4 weeks ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B

<missing> 4 weeks ago /bin/sh -c #(nop) LABEL name=CentOS Base ... 0 B

<missing> 4 weeks ago /bin/sh -c #(nop) ADD file:8d83f3e2c14f39e... 195 MB

**>>Working with Docker**

# docker start Container-ID/names **(Starting a Docker)**

# docker start d1b4678b0552

d1b4678b0552

# docker stop Container-ID/names **(Stopping a Docker)**

# docker stop d1b4678b0552

d1b4678b0552

**Note: You can also restart a container using “restart”.**

# docker top Container-ID/names **(you can see top processes within container)**

# docker rm Container-ID/names **(To delete a container - Container need to be in**

**stopped condtion)**

# docker stats Container-ID/names **(To show CPU and Memory utilization of container)**

**# docker stats abd276797c81**

**CONTAINER CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS**

**abd276797c81 0.00% 372 KiB / 1.796 GiB 0.02% 648 B / 648 B 0 B / 0 B 1**

# docker attach Container-ID/names **(login to the running container)**

**Note: Once, you exit from attached container then again it will go on stopped condtion. To avoid this condtion, exit with this command – Ctrl+P+Q**

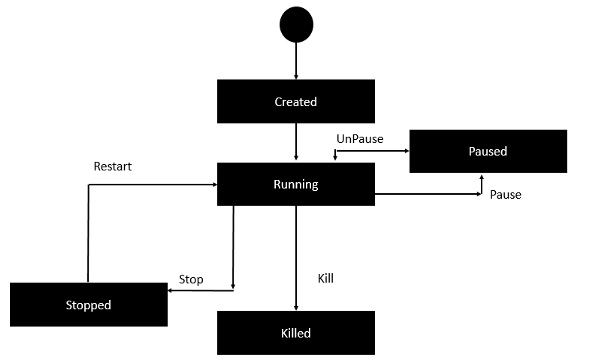
# docker pause Container-ID/names **(pause the processes in running contaimer)**

**Note: You cannot attach to a paused container, unpause it first.**

# docker unpause Container-ID/names **(unpause the paused container)**

# docker kill Container-ID/names **(kill running containers)**

**>>Docker – Container Lifecycle**



* Initially, the Docker container will be in the created state.
* Then the Docker container goes into the running state when the Docker run command is used.
* The Docker kill command is used to kill an existing Docker container.
* The Docker pause command is used to pause an existing Docker container.
* The Docker stop command is used to pause an existing Docker container.
* The Docker run command is used to put a container back from a stopped state to a running state.

**>>Docker – Container and Shells**

By default, when you launch a container, you will also use a shell commandwhile launching the container.

# docker run –it centos /bin/bash

We used this command to create a new container and then used the Ctrl+P+Q command to exit out of the container. It ensures that the container still exists even after we exit from the container.

We can verify that the container still exists with the Docker ps command. If we had to exit out of the container directly, then the container itself would be destroyed.

Now there is an easier way to attach to containers and exit them cleanly without the need of destroying them. One way of achieving this is by using the nsenter command.

Before we use the nsenter command, we need to get the Process ID of the container, because this is required by the nsenter command. We can get the Process ID via the Docker inspect command and filtering it via the Pid.

# docker inspect abd276797c81 | grep -i Pid

"Pid": 5056,

"PidMode": "",

"PidsLimit": 0,

**Nsenter is a method allows one to attach container without exiting it.**

Syntax:

# nsenter –m –u –n –p –i –t containerID command

Options:

* -u is used to mention the Uts namespace
* -m is used to mention the mount namespace
* -n is used to mention the network namespace
* -p is used to mention the process namespace
* -i s to make the container run in interactive mode.
* -t is used to connect the I/O streams of the container to the host OS.
* containerID − This is the ID of the container.
* Command − This is the command to run within the container.

# nsenter -m -u -n -p -i -t 5056 /bin/bash

[root@abd276797c81 /]#

**>>Packaging a customized container**

# docker commit -m "Created Image for me" -a "Vijay\_Kumar" amazing\_jennings vijay\_image

sha256:1435a6e8552b6d853e621bf5c0473dda89fb7c24447d4000cd809d1e99a6a44d

Options: (docker commit)

-a, --author string Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")

-c, --change list Apply Dockerfile instruction to the created image (default [])

--help Print usage

-m, --message string Commit message

-p, --pause Pause container during commit (default true)

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

vijay\_image latest 1435a6e8552b 7 seconds ago 402 MB

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

After creating this image you can create container and can login and check having all what you did.

# docker run -it vijay\_image /bin/bash

[root@150034494f41 /]#

**>>Docker – File and Building image via Docker file**

Docker also gives you the capability to create your own Docker images, and it can be done with the help of Docker Files. A Docker File is a simple text file with instructions on how to build your images.

Step 1 − Create a file called Docker File and edit it using vim. Please note that the name of the file has to be "Dockerfile" with "D" as capital.

# vi Dockerfile

Step 2 – Build a docker file using below instrunctions:

# cat Dockerfile

#### First Docker File ##########

FROM vijay\_image

MAINTAINER krvij.t@gmail.com

RUN echo "This is a test file for newly created image via Docker file" > test-file.txt

RUN useradd image

RUN echo "Image Created"

* The first line "#### First Docker File ##########" is a comment. You can add comments to the Docker File with the help of the # command
* The next line has to start with the FROM keyword. It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the vijay\_image image.
* The next command is the person who is going to maintain this image. Here you specify the MAINTAINER keyword and just mention the email ID.
* The RUN command is used to run instructions against the image. In our case, we will create a text file using echo command and create a user named “image”.
* The last command is used to display a message to the user.

**Docker build**command used to build own Docker image. Below is the syntax of command

# docker build –t ImageName:tagName dir

Options:

* -t − is to mention a tag to the image
* ImageName − This is the name you want to give to your image.
* TagName − This is the tag you want to give to your image.
* Dir − The directory where the Docker File is present.

Step 3 – Build image using docker build command as shown below:

**# docker build -t docker.io/myimage-file:1.0 /docker/first/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM vijay\_image

---> 37229231bba0

Step 2/5 : MAINTAINER krvij.t@gmail.com

---> Running in 70e3bdcdc47f

---> 676916b8fb4f

Removing intermediate container 70e3bdcdc47f

Step 3/5 : RUN echo "This is a test file for newly created image via Docker file" > test-file.txt

---> Running in 1a2075aec7ac

---> a7e60abc8bed

Removing intermediate container 1a2075aec7ac

Step 4/5 : RUN useradd image

---> Running in ab0a77b86d47

---> 32b923a144c3

Removing intermediate container ab0a77b86d47

Step 5/5 : RUN echo "Image Created"

---> Running in ab5d2c723c4a

Image Created

---> b0ead461c43d

Removing intermediate container ab5d2c723c4a

Successfully built b0ead461c43d

**# docker images**

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**docker.io/myimage-file 1.0 b0ead461c43d 27 seconds ago 196 MB**

docker.io/vijay\_image latest 37229231bba0 8 minutes ago 195 MB

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

You can start a container using the newly created image and can check:

# docker run -it docker.io/myimage-file:1.0 /bin/bash

[root@2d23df6328cc /]# ls

anaconda-post.log bin dev etc home lib lib64 media mnt opt proc root run sbin srv sys test-file.txt tmp usr var

[root@2d23df6328cc /]# cat test-file.txt

This is a test file for newly created image via Docker file

[root@2d23df6328cc /]# cat /etc/passwd | grep -i image

image:x:1000:1000::/home/image:/bin/bash

[root@2d23df6328cc /]#

**Options used in Docker file:**

FROM – from centos, from reddy, from scratch.

MAINTAINER – reference who maintains docker file

RUN – runs the necessary commands

CMD –

ENTRYPOINT – the service keeps on running.

ENV – set environment variable

WORKDIR –

USER –

EXPOSE – exposes the port

VOLUME – mounted volume to be used.

ADD – transfers data from local data to image file.

**/docker is the directory for Docker builds here for testing.**

**>>>USER and RUN**

**# cat Dockerfile**

# Docker file based on the latest centos image

# Non-Privledge user entry

FROM centos:latest

MAINTAINER kumar\_vijay

RUN useradd -ms /bin/bash testuser

USER testuser

RUN echo "Image Created Successfully"

**# docker build -t docker.io/myimage-nonroot:1.0 /docker/RunAsUser/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM centos:latest

---> e934aafc2206

Step 2/5 : MAINTAINER kumar\_vijay

---> Running in 7160ed30c861

---> ef70a23b1848

Removing intermediate container 7160ed30c861

Step 3/5 : RUN useradd -ms /bin/bash testuser

---> Running in 294aa051fcec

---> cf809da264eb

Removing intermediate container 294aa051fcec

Step 4/5 : USER testuser

---> Running in 72438fa4b50f

---> a5308d5700c4

Removing intermediate container 72438fa4b50f

Step 5/5 : RUN echo "Image Created Successfully"

---> Running in dd2e4894ccaa

Image Created Successfully

---> e65c46265882

Removing intermediate container dd2e4894ccaa

Successfully built e65c46265882

**# docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage-nonroot 1.0 e65c46265882 About a minute ago 199 MB

**# docker run -it docker.io/myimage-nonroot:1.0 /bin/bash**

[testuser@273c6d4954a3 /]$

**Note: Now if you use some commands after “USER” then it will fail as it will need access which the created user does not have. Please see below logs:**

**# docker build -t docker.io/myimage-failed:1.0 /docker/failed/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM centos:latest

---> e934aafc2206

Step 2/5 : MAINTAINER kumar\_vijay

---> Using cache

---> ef70a23b1848

Step 3/5 : RUN useradd -ms /bin/bash testuser1

---> Running in 47a82c10998d

---> 951f4c3aeecc

Removing intermediate container 47a82c10998d

Step 4/5 : USER testuser1

---> Running in 0c9c50aecdbf

---> 32eac35904e4

Removing intermediate container 0c9c50aecdbf

Step 5/5 : RUN echo "Image Created Successfullyi" >> /export.txt

---> Running in 8a1bbb063f12

/bin/sh: /export.txt: Permission denied

The command '/bin/sh -c echo "Image Created Successfullyi" >> /export.txt' returned a non-zero code: 1

**# docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

**<none><none> 32eac35904e4 15 seconds ago 199 MB**

docker.io/myimage-nonroot 1.0 e65c46265882 8 minutes ago 199 MB

docker.io/nginx latest b175e7467d66 2 weeks ago 109 MB

docker.io/centos latest e934aafc2206 2 weeks ago 199 MB

docker.io/kumarvij/support 2.0 b0ead461c43d 2 weeks ago 196 MB

docker.io/jenkins latest 7b210b6c238a 5 weeks ago 801 MB

localhost:5000/centos-p latest 2d194b392dd1 7 weeks ago 195 MB

docker.io/registry 2 d1fd7d86a825 3 months ago 33.3 MB

docker.io/hello-world latest f2a91732366c 5 months ago 1.85 kB

**>>>CMD vs RUN**

**# cat Dockerfile**

# Use centos:latest image

# it is for CMD vs RUN

FROM centos:latest

MAINTAINER kumarvijay

RUN useradd -ms /bin/bash user

CMD "echo" "Hello world"

USER user

**# docker build -t docker.io/myimage-runvscmd:1.0 /docker/echoserver/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM centos:latest

---> e934aafc2206

Step 2/5 : MAINTAINER kumarvijay

---> Running in 4aed8c673660

---> d0777be63f6c

Removing intermediate container 4aed8c673660

Step 3/5 : RUN useradd -ms /bin/bash user

---> Running in 00be7b982dba

---> 5469fbc23bc8

Removing intermediate container 00be7b982dba

Step 4/5 : CMD "echo" "Hello world"

---> Running in ef9bc25f32f2

---> a9b74b045193

Removing intermediate container ef9bc25f32f2

Step 5/5 : USER user

---> Running in 4bb6d8518aa5

---> 01c83b731b24

Removing intermediate container 4bb6d8518aa5

Successfully built 01c83b731b24

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage-runvscmd 1.0 01c83b731b24 50 seconds ago 199 MB

**# docker run -it docker.io/myimage-runvscmd:1.0**

**Hello world**

# docker run -it docker.io/myimage-runvscmd:1.0 /bin/bash

[user@8572092ff799 /]$

**RUN –** command is used for building the base image.

Runs during the building of image.

**CMD –** command is used to execute a simple command but is not executed during the build. It sets up the command to run when the container is initiated using the image we just built.

Runs during initiation of container/application run.

**>>>ENV**

**# cat Dockerfile**

# Use centos:latest image

# Its for Environmental Variable

FROM centos:latest

MAINTAINER krvij.t@gmail.com

RUN useradd test-image

ENV JAVA\_BIN /usr/java/jdk1.8.0/jre/bin

RUN echo "Image Successfully Created"

**# docker build -t docker.io/myimage\_java\_env:1.0 /docker/env-java/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM centos:latest

---> e934aafc2206

Step 2/5 : MAINTAINER krvij.t@gmail.com

---> Running in 3e710ca9c990

---> d5281ab212af

Removing intermediate container 3e710ca9c990

Step 3/5 : RUN useradd test-image

---> Running in d1facf93cff1

---> bd1eb9d1adc5

Removing intermediate container d1facf93cff1

Step 4/5 : ENV JAVA\_BIN /usr/java/jdk1.8.0/jre/bin

---> Running in 077c34b7c6b0

---> e4e607bf15e3

Removing intermediate container 077c34b7c6b0

Step 5/5 : RUN echo "Image Successfully Created"

---> Running in 7ee0cb207cb3

Image Successfully Created

---> 47d76f788537

Removing intermediate container 7ee0cb207cb3

Successfully built 47d76f788537

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage\_java\_env 1.0 47d76f788537 19 seconds ago 199 MB

# docker run -it docker.io/myimage\_java\_env:1.0 /bin/bash

[root@9e2e367d213c /]## env | grep -i JAVA\_BIN

JAVA\_BIN=/usr/java/jdk1.8.0/jre/bin

[root@9e2e367d213c /]#

Same can be done by the below command using RUN in Dockerfile.

**RUN echo "export JAVA\_BIN=/usr/java/jdk1.8.0/jre/bin" >> /root/.bash\_profile**

**>>>EntryPoint**

**# cat Dockerfile**

# Use centos:latest image

# It is for testing of EntryPoint

FROM centos:latest

MAINTAINER krvij.t@gmail.com

RUN useradd -ms /bin/bash user

ENTRYPOINT echo "Hello World Again"

USER user

**# docker build -t docker.io/myimage\_entry:1.0 /docker/entry/**

Sending build context to Docker daemon 2.048 kB

Step 1/5 : FROM centos:latest

---> e934aafc2206

Step 2/5 : MAINTAINER krvij.t@gmail.com

---> Using cache

---> d5281ab212af

Step 3/5 : RUN useradd -ms /bin/bash user

---> Running in 366c7b55913a

---> 98bac023ce1e

Removing intermediate container 366c7b55913a

Step 4/5 : ENTRYPOINT echo "Hello World Again"

---> Running in ff6ffd19178a

---> 9af5e6840481

Removing intermediate container ff6ffd19178a

Step 5/5 : USER user

---> Running in de9979e9e168

---> ef52aa849d32

Removing intermediate container de9979e9e168

Successfully built ef52aa849d32

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage\_entry 1.0 ef52aa849d32 5 seconds ago 199 MB

# docker run -it docker.io/myimage\_entry:1.0 /bin/bash

Hello World Again

# docker run -it docker.io/myimage\_entry:1.0

Hello World Again

**>>>Difference between CMD and Entrypoint**

**--------**

# docker run docker.io/myimage-runvscmd:1.0

Hello world

# docker run docker.io/myimage\_entry:1.0

Hello World Again

--------

# docker run -it docker.io/myimage-runvscmd:1.0 /bin/bash

[user@313a0a2a2f8e /]$

# docker run -it docker.io/myimage\_entry:1.0 /bin/bash

Hello World Again

--------

# docker run docker.io/myimage-runvscmd:1.0 /bin/echo "Hi"

Hi

# docker run docker.io/myimage\_entry:1.0 /bin/echo "Hi"

Hello World Again

**>>>EXPOSE**

**# cat Dockerfile**

# Use centos:latest image

# Will start apache server on all containers

FROM centos:latest

MAINTAINER krvij.t@gmail.com

RUN yum install -y httpd net-tools

RUN echo "This is a custom index file build during image creation" > /var/www/html/index.html

EXPOSE 80

ENTRYPOINT apachectl "-DFOREGROUND"

# docker build -t docker.io/myimage\_apacheinstall:1.0 /docker/apacheinstall/

Sending build context to Docker daemon 2.048 kB

Step 1/6 : FROM centos:latest

---> e934aafc2206

Step 2/6 : MAINTAINER krvij.t@gmail.com

---> Using cache

---> d5281ab212af

Step 3/6 : RUN yum install -y httpd net-tools

---> Running in c2dc6f69dd07

Loaded plugins: fastestmirror, ovl

Determining fastest mirrors

\* base: centos.excellmedia.net

\* extras: centos.excellmedia.net

\* updates: centos.excellmedia.net

Resolving Dependencies

--> Running transaction check

---> Package httpd.x86\_64 0:2.4.6-67.el7.centos.6 will be installed

--> Processing Dependency: httpd-tools = 2.4.6-67.el7.centos.6 for package: httpd-2.4.6-67.el7.centos.6.x86\_64

--> Processing Dependency: system-logos >= 7.92.1-1 for package: httpd-2.4.6-67.el7.centos.6.x86\_64

--> Processing Dependency: /etc/mime.types for package: httpd-2.4.6-67.el7.centos.6.x86\_64

--> Processing Dependency: libaprutil-1.so.0()(64bit) for package: httpd-2.4.6-67.el7.centos.6.x86\_64

--> Processing Dependency: libapr-1.so.0()(64bit) for package: httpd-2.4.6-67.el7.centos.6.x86\_64

---> Package net-tools.x86\_64 0:2.0-0.22.20131004git.el7 will be installed

--> 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 0:2.4.6-67.el7.centos.6 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 2.4.6-67.el7.centos.6 updates 2.7 M

net-tools x86\_64 2.0-0.22.20131004git.el7 base 305 k

Installing for dependencies:

apr x86\_64 1.4.8-3.el7\_4.1 updates 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 2.4.6-67.el7.centos.6 updates 88 k

mailcap noarch 2.1.41-2.el7 base 31 k

Transaction Summary

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

Install 2 Packages (+5 Dependent packages)

Total download size: 25 M

Installed size: 32 M

Downloading packages:

Public key for apr-1.4.8-3.el7\_4.1.x86\_64.rpm is not installed

warning: /var/cache/yum/x86\_64/7/updates/packages/apr-1.4.8-3.el7\_4.1.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

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

Total 3.3 MB/s | 25 MB 00:07

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

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-7-4.1708.el7.centos.x86\_64 (@CentOS)

From : /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 1/7

Installing : apr-util-1.5.2-6.el7.x86\_64 2/7

Installing : httpd-tools-2.4.6-67.el7.centos.6.x86\_64 3/7

Installing : centos-logos-70.0.6-3.el7.centos.noarch 4/7

Installing : mailcap-2.1.41-2.el7.noarch 5/7

Installing : httpd-2.4.6-67.el7.centos.6.x86\_64 6/7

Installing : net-tools-2.0-0.22.20131004git.el7.x86\_64 7/7

Verifying : net-tools-2.0-0.22.20131004git.el7.x86\_64 1/7

Verifying : mailcap-2.1.41-2.el7.noarch 2/7

Verifying : httpd-2.4.6-67.el7.centos.6.x86\_64 3/7

Verifying : apr-util-1.5.2-6.el7.x86\_64 4/7

Verifying : httpd-tools-2.4.6-67.el7.centos.6.x86\_64 5/7

Verifying : apr-1.4.8-3.el7\_4.1.x86\_64 6/7

Verifying : centos-logos-70.0.6-3.el7.centos.noarch 7/7

Installed:

httpd.x86\_64 0:2.4.6-67.el7.centos.6

net-tools.x86\_64 0:2.0-0.22.20131004git.el7

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 0:2.4.6-67.el7.centos.6

mailcap.noarch 0:2.1.41-2.el7

Complete!

---> ce84adf5d82c

Removing intermediate container c2dc6f69dd07

Step 4/6 : RUN echo "This is a custom index file build during image creation" > /var/www/html/index.html

---> Running in c5721f9556cf

---> 2fc4c50a09f4

Removing intermediate container c5721f9556cf

Step 5/6 : EXPOSE 80

---> Running in 69e8682dea71

---> 7803093b8083

Removing intermediate container 69e8682dea71

Step 6/6 : ENTRYPOINT apachectl "-DFOREGROUND"

---> Running in c6ead7ccce7c

---> f137c97b10a2

Removing intermediate container c6ead7ccce7c

Successfully built f137c97b10a2

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage\_apacheinstall 1.0 f137c97b10a2 About a minute ago 335 MB

# docker run -d --name apache docker.io/myimage\_apacheinstall:1.0

3cf7319e2691ee6d5355c56b820afb2ac445ca31409ce27e4df2d3f4b1072a19

# docker inspect 3cf7319e2691 | grep -i IPAddress

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2",

You can check now using elinks:

# elinks http://172.17.0.2

# docker exec apache /bin/cat /var/www/html/index.html

This is a custom index file build during image creation

**>>>WORKDIR**

**# cat Dockerfile**

# use centos:latest image

# its for setting workdir

FROM centos:latest

MAINTAINER krvij.t@gmail.com

WORKDIR /newtemp

CMD pwd

# docker build -t docker.io/myimage\_workdir:1.0 /docker/workdir/

Sending build context to Docker daemon 2.048 kB

Step 1/4 : FROM centos:latest

---> e934aafc2206

Step 2/4 : MAINTAINER krvij.t@gmail.com

---> Using cache

---> d5281ab212af

Step 3/4 : WORKDIR /newtemp

---> fcaf5a50edde

Removing intermediate container cd4740753948

Step 4/4 : CMD pwd

---> Running in 061726983ee0

---> 3243f0eb6f93

Removing intermediate container 061726983ee0

Successfully built 3243f0eb6f93

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage\_workdir 1.0 3243f0eb6f93 10 seconds ago 199 MB

# docker run -it docker.io/myimage\_workdir:1.0 /bin/bash

[root@a24b3db302a0 newtemp]#

[root@a24b3db302a0 newtemp]# pwd

/newtemp

**>>Few Important Options with Docker**

**Management Commands:**

* container Manage containers
* image Manage images
* network Manage networks
* node Manage Swarm nodes
* plugin Manage plugins
* secret Manage Docker secrets
* service Manage services
* stack Manage Docker stacks
* swarm Manage Swarm
* system Manage Docker
* volume Manage volumes

**Commands:**

* attach Attach to a running container
* build Build an image from a Dockerfile
* commit Create a new image from a container's changes
* cp Copy files/folders between a container and the local filesystem
* create Create a new container
* diff Inspect changes on a container's filesystem
* events Get real time events from the server
* exec Run a command in a running container
* export Export a container's filesystem as a tar archive
* history Show the history of an image
* images List images
* import Import the contents from a tarball to create a filesystem image
* info Display system-wide information
* inspect Return low-level information on Docker objects
* kill Kill one or more running containers
* load Load an image from a tar archive or STDIN
* login Log in to a Docker registry
* logout Log out from a Docker registry
* logs Fetch the logs of a container
* pause Pause all processes within one or more containers
* port List port mappings or a specific mapping for the container
* ps List containers
* pull Pull an image or a repository from a registry
* push Push an image or a repository to a registry
* rename Rename a container
* restart Restart one or more containers
* rm Remove one or more containers
* rmi Remove one or more images
* run Run a command in a new container
* save Save one or more images to a tar archive (streamed to STDOUT by default)
* search Search the Docker Hub for images
* start Start one or more stopped containers
* stats Display a live stream of container(s) resource usage statistics
* stop Stop one or more running containers
* tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE
* top Display the running processes of a container
* unpause Unpause all processes within one or more containers
* update Update configuration of one or more containers
* version Show the Docker version information
* wait Block until one or more containers stop, then print their exit codes

**>>Docker – Public Repositories**

Public repositories can be used to host Docker images which can be used by everyone else. An example is the images which are available in Docker Hub. Most of the images such as Centos, Ubuntu, and Jenkins are all publicly available for all. We can also make our images available by publishing it to the public repository on Docker Hub.

Let’s check what images we have.

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage-file 1.0 b0ead461c43d 2 hours ago 196 MB

docker.io/vijay\_image latest 37229231bba0 2 hours ago 195 MB

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

I would upload the myimage-file:1.0.

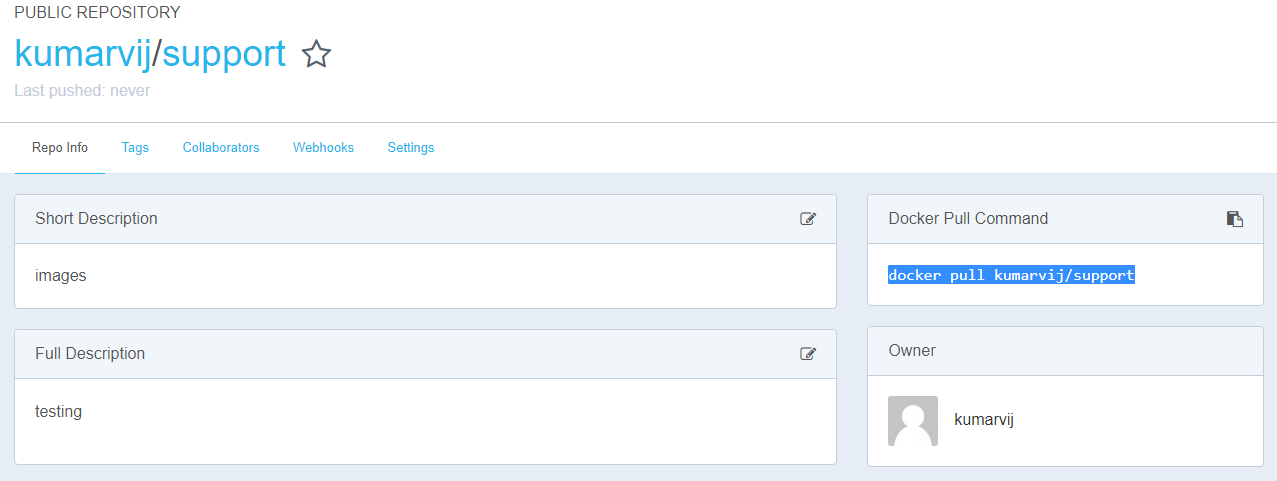
Follow below steps if you have not created your id and any repository on Docker hub.

Step 1 − Log into Docker Hub and create your repository. This is the repository where your image will be stored. Go to <https://hub.docker.com/>and log in with your credentials.

Step 2 − Click the button "Create Repository" on the above screen and create a repository with the name “support”. Make sure that the visibility of the repository is public.

Once the repository is created, make a note of the pull command which is attached to the repository.

# docker pull kumarvij/support



Step 3 − Now go back to the Docker Host. Here we need to tag our myimage-file to the new repository created in Docker Hub. We can do this via the Docker **tag** command.

**docker tag**

This method allows one to tag an image to the relevant repository.

Syntax

docker tag imageID Repositoryname

Options:

imageID − This is the ImageID which needs to be tagged to the repository.

Repositoryname − This is the repository name to which the ImageID needs to be tagged to.

# docker tag b0ead461c43d kumarvij/support:2.0

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/myimage-file 1.0 b0ead461c43d 2 hours ago 196 MB

**kumarvij/support 2.0 b0ead461c43d 2 hours ago 196 MB**

docker.io/vijay\_image latest 37229231bba0 2 hours ago 195 MB

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

Step 4 − Issue the Docker **login** command to login into the Docker Hub repository from the command prompt. The Docker login command will prompt you for the username and password to the Docker Hub repository.

# docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: kumarvij

Password:

Login Succeeded

Step 5 − Once the image has been tagged, it’s now time to push the image to the Docker Hub repository. We can do this via the Docker **push** command.

**# docker push**

This method allows one to push images to the Docker Hub.

Syntax

# docker push Repositoryname

Options:

Repositoryname − This is the repository name which needs to be pushed to the Docker Hub.

# docker push kumarvij/support:2.0

The push refers to a repository [docker.io/kumarvij/support]

98aa8b67a9ba: Pushed

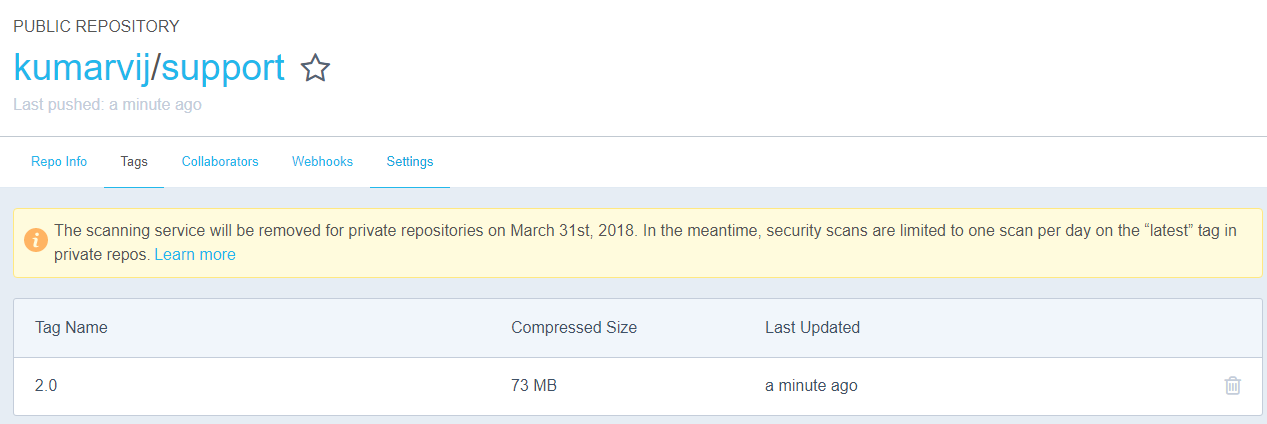
671dc6928ef0: Pushed

bfec9d87149b: Pushed

b03095563b79: Mounted from library/centos

2.0: digest: sha256:2be172fcb383c4d96eabf7583e9f07617cc918e7108b9177da3abc7d86d3390a size: 1151

If you go back to the Docker Hub page and go to your repository, you will see the tag name in the repository.



Now let’s try to pull the repository we uploaded onto our Docker host. Let’s first delete the images, docker.io/myimage-file:1.0 and kumarvij/support:2.0, from the local Docker host. Let’s use the Docker pull command to pull the repository from the Docker Hub.

# docker pull kumarvij/support:2.0

Trying to pull repository docker.io/kumarvij/support ...

2.0: Pulling from docker.io/kumarvij/support

5e35d10a3eba: Already exists

2dbaadcdb610: Already exists

66c1cf5c01a2: Pull complete

675b3dc141ab: Pull complete

Digest: sha256:2be172fcb383c4d96eabf7583e9f07617cc918e7108b9177da3abc7d86d3390a

Status: Downloaded newer image for docker.io/kumarvij/support:2.0

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/kumarvij/support 2.0 b0ead461c43d 3 hours ago 196 MB

docker.io/vijay\_image latest 37229231bba0 3 hours ago 195 MB

docker.io/centos latest 2d194b392dd1 4 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

**>>Docker – Managing Ports**

In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host. Let’s look at an example of how this can be achieved.

In our example, we are going to download the Jenkins container from Docker Hub. We are then going to map the Jenkins port number to the port number on the Docker host.

Login to the Docker Hub and find Jenkins image for testing of this service. You will be seen the pull command to pull this image. Now, go to your Docker server and run pull command to pull the Jenkins image.

# docker pull jenkins

Using default tag: latest

Trying to pull repository docker.io/library/jenkins ...

latest: Pulling from docker.io/library/jenkins

c73ab1c6897b: Pull complete

1ab373b3deae: Pull complete

b542772b4177: Pull complete

57c8de432dbe: Pull complete

da44f64ae999: Pull complete

0bbc7b377a91: Pull complete

1b6c70b3786f: Pull complete

d9bbcf733166: Pull complete

b1d3e8de8ec6: Pull complete

c1455927bc48: Pull complete

1d3c626322f1: Pull complete

23612106c74c: Pull complete

596d10c47bfa: Pull complete

62e8d5201cdb: Pull complete

2c3d92fb5e98: Pull complete

bc5965f9d105: Pull complete

6816953234b3: Pull complete

e49ca30dec01: Pull complete

0713317dfd5a: Pull complete

522ab7b13eb6: Pull complete

Digest: sha256:93263adb6ab1ecb240b342a9e62e782c5b46d4d87cd01830021d1dfe89acb518

Status: Downloaded newer image for docker.io/jenkins:latest

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/kumarvij/support 2.0 b0ead461c43d 5 days ago 196 MB

docker.io/vijay\_image latest 37229231bba0 5 days ago 195 MB

docker.io/jenkins latest 7b210b6c238a 3 weeks ago 801 MB

docker.io/centos latest 2d194b392dd1 5 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

Now, you can start a container and check which ports are using by Jenkins.

# docker inspect 0d8943d6ad21

"Ports": {

"50000/tcp": null,

"8080/tcp": null

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

0d8943d6ad21 docker.io/jenkins "/bin/tini -- /usr..." About a minute ago Up About a minute 8080/tcp, 50000/tcp hungry\_stonebraker

The output of the inspect command gives a JSON output. If we observe the output, we can see that there is a section of "ExposedPorts" and see that there are two ports mentioned. One is the data port of 8080 and the other is the control port of 50000.

To run Jenkins and map the ports, you need to change the Docker run command and add the ‘p’ option which specifies the port mapping. So, you need to run the following command −

# docker run -p 8080:8080 -p 50000:50000 jenkins

The left-hand side of the port number mapping is the Docker host port to map to and the right-hand side is the Docker container port number.

When you open the browser and navigate to the Docker host on port 8080, you will see Jenkins up and running. Below is in explained way:

# docker run -p 8080:8080 -p 50000:50000 docker.io/jenkins

Running from: /usr/share/jenkins/jenkins.war

webroot: EnvVars.masterEnvVars.get("JENKINS\_HOME")

Apr 11, 2018 1:16:02 PM Main deleteWinstoneTempContents

WARNING: Failed to delete the temporary Winstone file /tmp/winstone/jenkins.war

Apr 11, 2018 1:16:02 PM org.eclipse.jetty.util.log.JavaUtilLog info

INFO: Logging initialized @3667ms

Apr 11, 2018 1:16:02 PM winstone.Logger logInternal

INFO: Beginning extraction from war file

Apr 11, 2018 1:16:05 PM org.eclipse.jetty.util.log.JavaUtilLog warn

WARNING: Empty contextPath

Apr 11, 2018 1:16:06 PM org.eclipse.jetty.util.log.JavaUtilLog info

INFO: jetty-9.2.z-SNAPSHOT

Apr 11, 2018 1:16:08 PM org.eclipse.jetty.util.log.JavaUtilLog info

INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet

Jenkins home directory: /var/jenkins\_home found at: EnvVars.masterEnvVars.get("JENKINS\_HOME")

Apr 11, 2018 1:16:10 PM org.eclipse.jetty.util.log.JavaUtilLog info

INFO: Started w.@47404bea{/,file:/var/jenkins\_home/war/,AVAILABLE}{/var/jenkins\_home/war}

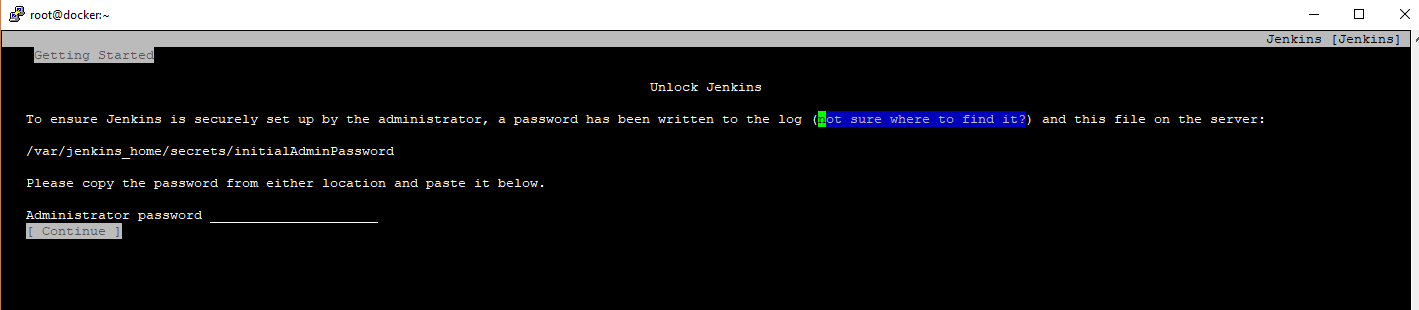
Apr 11, 2018 1:16:10 PM org.eclipse.jetty.util.log.JavaUtilLog info

INFO: Started ServerConnector@1c80e49b{HTTP/1.1}{0.0.0.0:8080}

Apr 11, 2018 1:16:10 PM org.eclipse.jetty.util.log.JavaUtilLog info

……..

# elinks 172.17.0.2:8080



Taking another example of **nginx:**

# docker pull ngnix

Using default tag: latest

Trying to pull repository docker.io/library/ngnix ...

^C

[root@docker ~]# docker pull nginx:latest

Trying to pull repository docker.io/library/nginx ...

latest: Pulling from docker.io/library/nginx

2a72cbf407d6: Pull complete

04b2d3302d48: Pull complete

e7f619103861: Pull complete

Digest: sha256:db7a32d2a577ac1daaf7e028b66002341df21c115dab9b14f768ae7593bb48b1

Status: Downloaded newer image for docker.io/nginx:latest

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/nginx latest b175e7467d66 19 hours ago 109 MB

docker.io/kumarvij/support 2.0 b0ead461c43d 5 days ago 196 MB

docker.io/vijay\_image latest 37229231bba0 5 days ago 195 MB

docker.io/jenkins latest 7b210b6c238a 3 weeks ago 801 MB

docker.io/centos latest 2d194b392dd1 5 weeks ago 195 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

# docker run -d docker.io/nginx

cd9f5960013d080b2fed8ebd126a54594d961873d5910209f2814a3648750a3c

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

cd9f5960013d docker.io/nginx "nginx -g 'daemon ..." 5 seconds ago Up 4 seconds 80/tcp dazzling\_dijkstra

You can check it is working on port 80. Now, you can inspect its IP address/ports etc. Now, we will redirect local port 80/8080 to ngnix remote port 80

# docker run -d docker.io/nginx

cd9f5960013d080b2fed8ebd126a54594d961873d5910209f2814a3648750a3c

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

cd9f5960013d docker.io/nginx "nginx -g 'daemon ..." 5 seconds ago Up 4 seconds 80/tcp dazzling\_dijkstra

# docker stop cd9f5960013d

cd9f5960013d

# docker run -d -p 80:80 docker.io/nginx

0a461ee759b93f067326440a4dd1e7695e99797262d2ce7f3f7f16843926cb3c

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

0a461ee759b9 docker.io/nginx "nginx -g 'daemon ..." 4 seconds ago Up 2 seconds 0.0.0.0:80->80/tcp affectionate\_meitner

# docker stop 0a461ee759b9

0a461ee759b9

# docker run -d -p 8080:80 docker.io/nginx

ca9563a6b20ce7295f76aff2e05d7251e4e90a6a150be716f0214bae48dcfa3a

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ca9563a6b20c docker.io/nginx "nginx -g 'daemon ..." 3 seconds ago Up 1 second 0.0.0.0:8080->80/tcp sharp\_ritchie

**>>Docker – Private registries**

You might have the need to have your own private repositories. You may not want to host the repositories on Docker Hub. For this, there is a repository container itself from Docker. Please look to the process how we can download and use the container for registry.

Use docker run command to download the private registry.

# docker run -d -p 5000:5000 --name registry registry:2

Unable to find image 'registry:2' locally

Trying to pull repository docker.io/library/registry ...

2: Pulling from docker.io/library/registry

81033e7c1d6a: Pull complete

b235084c2315: Pull complete

c692f3a6894b: Pull complete

ba2177f3a70e: Pull complete

a8d793620947: Pull complete

Digest: sha256:672d519d7fd7bbc7a448d17956ebeefe225d5eb27509d8dc5ce67ecb4a0bce54

Status: Downloaded newer image for docker.io/registry:2

86ac479113ba5015c79f4631b442d61cc01d398aef83cc28068afb29d5aa9a44

The following points need to be noted about the above command –

* **Registry** is the container managed by Docker which can be used to host private repositories.
* The port number exposed by the container is 5000. Hence with the **–p command**, we are mapping the same port number to the 5000 port number on our localhost.
* We are just tagging the registry container as **“2”**, to differentiate it on the Docker host.
* The **–d** option is used to run the container in detached mode. This is so that the container can run in the background

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

86ac479113ba registry:2 "/entrypoint.sh /e..." About a minute ago Up About a minute 0.0.0.0:5000->5000/tcp registry

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/nginx latest b175e7467d66 38 hours ago 109 MB

docker.io/kumarvij/support 2.0 b0ead461c43d 5 days ago 196 MB

docker.io/vijay\_image latest 37229231bba0 5 days ago 195 MB

docker.io/jenkins latest 7b210b6c238a 3 weeks ago 801 MB

docker.io/centos latest 2d194b392dd1 5 weeks ago 195 MB

docker.io/registry 2 d1fd7d86a825 3 months ago 33.3 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

Now let’s tag one of our existing images so that we can push it to our local repository. In our example, since we have the centos image available locally, we are going to tag it to our private repository and add a tag name of centos.

# docker tag 2d194b392dd1 localhost:5000/centos-p

The following points need to be noted about the above command –

* 2d194b392dd1 refers to the Image ID for the centos image.
* localhost:5000 is the location of our private repository.
* We are tagging the repository name as centos-p in our private repository.

# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/nginx latest b175e7467d66 38 hours ago 109 MB

docker.io/kumarvij/support 2.0 b0ead461c43d 5 days ago 196 MB

docker.io/vijay\_image latest 37229231bba0 5 days ago 195 MB

docker.io/jenkins latest 7b210b6c238a 3 weeks ago 801 MB

docker.io/centos latest 2d194b392dd1 5 weeks ago 195 MB

localhost:5000/centos-p latest 2d194b392dd1 5 weeks ago 195 MB

docker.io/registry 2 d1fd7d86a825 3 months ago 33.3 MB

docker.io/hello-world latest f2a91732366c 4 months ago 1.85 kB

Now push the image to our private repository.

# docker push localhost:5000/centos-p

The push refers to a repository [localhost:5000/centos-p]

b03095563b79: Pushed

latest: digest: sha256:8c7ac054adab3692f7026d49fd1c4df69aa6a138b2f076b432d2ac0164c022d3 size: 529

It’s time to delete the local images of centos.

# docker rmi docker.io/centos

Untagged: docker.io/centos:latest

Untagged: [docker.io/centos@sha256:dcbc4e5e7052ea2306eed59563da1fec09196f2ecacbe042acbdcd2b44b05270](mailto:docker.io/centos@sha256:dcbc4e5e7052ea2306eed59563da1fec09196f2ecacbe042acbdcd2b44b05270)

# docker rmi 2d194b392dd1

Untagged: localhost:5000/centos-p:latest

Untagged: localhost:5000/centos-p@sha256:8c7ac054adab3692f7026d49fd1c4df69aa6a138b2f076b432d2ac0164c022d3

Deleted: sha256:2d194b392dd16955847a14f969b2dd319251471ffa6356be6d8f16c5bf53db9b

Now that we don’t have any centos images on our local machine, we can now use the following Docker pull command to pull the centos image from our private repository.

# docker pull localhost:5000/centos-p

Using default tag: latest

Trying to pull repository localhost:5000/centos-p ...

latest: Pulling from localhost:5000/centos-p

5e35d10a3eba: Already exists

Digest: sha256:8c7ac054adab3692f7026d49fd1c4df69aa6a138b2f076b432d2ac0164c022d3

Status: Downloaded newer image for localhost:5000/centos-p:latest

**>>Docker – Container Linking**

Container Linking allows multiple containers to link with each other. It is a better option than exposing ports.

We are taking example with Jenkins image. You can either download image or can work with downloaded image. You can specify a name to the container by using the –-name option. This will be our source container.

# docker run --name=source-iso -d jenkins

9c0ae957a26913742407625d87c77f13a27c0fcb48b3df3f0ad04dd784485e3a

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9c0ae957a269 jenkins "/bin/tini -- /usr..." 12 seconds ago Up 10 seconds 8080/tcp, 50000/tcp **source-iso**

Next, it is time to launch the destination container, but this time, we will link it with our source container. For our destination container, we will use the standard centos image we have.

# docker run --name=reca --link=source-iso:alias-src -it centos:latest /bin/bash

[root@804ba930fbf7 /]#

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

804ba930fbf7 centos:latest "/bin/bash" About a minute ago Up 3 seconds reca

9c0ae957a269 jenkins "/bin/tini -- /usr..." 3 minutes ago Up 3 minutes 8080/tcp, 50000/tcp source-iso

Just go and attach to this container.

# docker attach reca

Then run the env command. You will notice new variables for linking with the source container.

**>>Docker – Storage**

Docker has multiple storage drivers that allow one to work with the underlying storage devices. The following table shows the different storage drivers along with the technology used for the storage drivers.

|  |  |
| --- | --- |
| Technology | Storage Driver |
| OverlayFS | overlay or overlay2 |
| AUFS | aufs |
| Btrfs | brtfs |
| Device Manager | devicemanager |
| VFS | vfs |
| ZFS | zfs |

Let us now discuss some of the instances in which you would use the various storage drivers −

AUFS

* This is a stable driver; can be used for production-ready applications.
* It has good memory usage and is good for ensuring a smooth Docker experience for containers.
* There is a high-write activity associated with this driver which should be considered.
* It’s good for systems which are of Platform as a service type work.

Devicemapper

* This is a stable driver; ensures a smooth Docker experience.
* This driver is good for testing applications in the lab.
* This driver is in line with the main Linux kernel functionality.

Btrfs

* This driver is in line with the main Linux kernel functionality.
* There is a high-write activity associated with this driver which should be considered.
* This driver is good for instances where you maintain multiple build pools.

Ovelay

* This is a stable driver and it is in line with the main Linux kernel functionality.
* It has a good memory usage.
* This driver is good for testing applications in the lab.

ZFS

* This is a stable driver and it is good for testing applications in the lab.
* It’s good for systems which are of Platform-as-a-Service type work.

You can check storage driver info using below command:

# docker info

In Docker, you have a separate volume that can shared across containers. These are known as data volumes. Some of the features of data volume are –

* They are initialized when the container is created.
* They can be shared and also reused amongst many containers.
* Any changes to the volume itself can be made directly.
* They exist even after the container is deleted.

# docker run -it --name voltest1 -v /mydata centos:latest /bin/bash

[root@1d58a67ba35d /]# ls -ld mydata/

drwxr-xr-x 2 root root 4096 Apr 16 10:10 mydata/

[root@1d58a67ba35d /]#

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1d58a67ba35d centos:latest "/bin/bash" 40 seconds ago Up 36 seconds voltest1

-v used for volume to be created on container

--name used for name of container

[root@1d58a67ba35d /]# df -h

Filesystem Size Used Avail Use% Mounted on

overlay 30G 9.5G 19G 34% /

tmpfs 920M 0 920M 0% /dev

tmpfs 920M 0 920M 0% /sys/fs/cgroup

/dev/mapper/rootvg-rootlv 30G 9.5G 19G 34% /mydata

shm 64M 0 64M 0% /dev/shm

tmpfs 920M 0 920M 0% /proc/scsi

tmpfs 920M 0 920M 0% /sys/firmware

[root@1d58a67ba35d /]#

After creating file and directory:

# ls /mydata/

first second

[root@1d58a67ba35d /]#

Now, inspect it and you can find the location of data in /mydata of container on host.

# docker inspect voltest1 | grep -i volume

"VolumeDriver": "",

"VolumesFrom": null,

"Type": "volume",

"Source": "/var/lib/docker/volumes/c717a5de01c6f36326c78835a3bd7319a4563ebaa0c2bf74711e743ab83fdf5a/\_data",

"Volumes": {

# cd /var/lib/docker/volumes/c717a5de01c6f36326c78835a3bd7319a4563ebaa0c2bf74711e743ab83fdf5a/\_data

# pwd

/var/lib/docker/volumes/c717a5de01c6f36326c78835a3bd7319a4563ebaa0c2bf74711e743ab83fdf5a/\_data

# ll

total 4

-rw-r--r-- 1 root root 0 Apr 16 15:42 first

drwxr-xr-x 2 root root 4096 Apr 16 15:42 second

Now it’s time to map host data to container.

# mkdir host\_data

# cd host\_data/

# echo "testing from outside" > message.txt

# mkdir hosts

# ll

total 8

drwxr-xr-x 2 root root 4096 Apr 17 14:47 hosts

-rw-r--r-- 1 root root 21 Apr 17 14:47 message.txt

# pwd

/root/host\_data

# docker run -it --name voltest2 -v /root/host\_data:/mydata centos:latest /bin/bash

[root@1adb88b04f01 /]# ls /mydata/

hosts message.txt

[root@1adb88b04f01 /]# cat /mydata/message.txt

testing from outside

[root@1adb88b04f01 /]#

Directory on the Docker host get attached to the container. Now if you go inside the container you will find all the data over there. Moreover if you add any directory or file to the same location then it will also reflect to the container.

**>>Changing the storage driver of a container**

Below is the storage driver of running container “voltest1”.

# docker inspect voltest1 | grep -i driver

"Driver": "overlay2",

"VolumeDriver": "",

"GraphDriver": {

"Driver": "local",

Now we will change the driver of an image to something else while launching a container.

# docker run -it --volume-driver flocker centos:latest /bin/bash

[root@4aad9df53e7a /]#

[root@4aad9df53e7a /]#

Or

# docker run -itd --volume-driver flocker centos:latest (--detach Run container in background and

print container ID)

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4aad9df53e7a centos:latest "/bin/bash" 7 seconds ago Up 5 seconds optimistic\_pare

# docker inspect optimistic\_pare | grep -i driver

"Driver": "overlay2",

"VolumeDriver": "flocker",

"GraphDriver": {

**>>Docker Networking**

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an ifconfig on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host.This is a bridge between the Docker Host and the Linux Host.

# ifconfig

**docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500**

**inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0**

**ether 02:42:66:3e:f3:43 txqueuelen 0 (Ethernet)**

**RX packets 0 bytes 0 (0.0 B)**

**RX errors 0 dropped 0 overruns 0 frame 0**

**TX packets 0 bytes 0 (0.0 B)**

**TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0**

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.81.129.143 netmask 255.255.252.0 broadcast 10.81.131.255

inet6 fe80::f8e1:4a2e:621f:4bff prefixlen 64 scopeid 0x20<link>

ether 52:54:00:ef:69:b4 txqueuelen 1000 (Ethernet)

RX packets 1277153 bytes 918639534 (876.0 MiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 571866 bytes 809471758 (771.9 MiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0x10<host>

loop txqueuelen 1 (Local Loopback)

RX packets 162235 bytes 28132341 (26.8 MiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 162235 bytes 28132341 (26.8 MiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Note: By default Docker uses 172.17.X.X subnet and then uses DHCP to assign IP to all the containers.

**>>>listing all Docker network**

# docker network ls

NETWORK ID NAME DRIVER SCOPE

f26f0ee26036 bridge bridge local

12f2a1445904 host host local

22e7bb9456ca none null local

# docker network ls --no-trunc (Give full Network ID)

NETWORK ID NAME DRIVER SCOPE

f26f0ee26036cda8cf3e69e4b526f455656c714c1bdc6dd59a23ac36eb298083 bridge bridge local

12f2a1445904cb93f70ebaa42229813baf75d83042eddae6b105dac7f73c5704 host host local

22e7bb9456ca281bd87fdc64c06a27b081e201bbb129175ec68b3c483371cc66 none null local

# docker network inspect bridge (Inspect a Network)

Now let’s run a container and you will able to see container details attached to the bridge network.

Note: You can go to man page of a docker subcommands as shown below:

# man docker network create

**>>>Docker network creation and removal**

Create new network having driver “bridge”, IP Subnet “172.18.0.0/24”, gateway “172.18.0.1” with named “mybridge01”.

Default network driver is also “bridge” means if you will not mention any driver name then it will pick bridge by default.

# docker network create --subnet 172.18.0.0/24 --gateway 172.18.0.1 --driver bridge mybridge01

a66ad9fe8d9d7021c1e9be6ab9452579f97a88a4638dd9d8cd1e488003f1f1f8

# docker network ls

NETWORK ID NAME DRIVER SCOPE

f26f0ee26036 bridge bridge local

12f2a1445904 host host local

**a66ad9fe8d9d mybridge01 bridge local**

22e7bb9456ca none null local

It will also visible in ifconfig command:

**# ifconfig**

**br-a66ad9fe8d9d: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500**

**inet 172.18.0.1 netmask 255.255.255.0 broadcast 0.0.0.0**

**ether 02:42:55:14:0c:4d txqueuelen 0 (Ethernet)**

**RX packets 0 bytes 0 (0.0 B)**

**RX errors 0 dropped 0 overruns 0 frame 0**

**TX packets 0 bytes 0 (0.0 B)**

**TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0**

**--**

--

--

--

--

--

Now you can attach a new container while launching it. See below how you can do that:

docker run -it --network=mybridge01 docker.io/centos /bin/bash

[root@658a50968395 /]#

You can inspect network and machine.

# docker inspect 658a50968395 | grep -i IPAddress

"SecondaryIPAddresses": null,

"IPAddress": "",

"IPAddress": "172.18.0.2",

# docker network inspect mybridge01

--

**--**

"IPAM": {

"Driver": "default",

"Options": {},

"Config": [

{

"Subnet": "172.18.0.0/24",

"Gateway": "172.18.0.1"

}

]

},

--

--

**>>>Removal of Docker network**

# docker network rm name/network id

**Note: no process linked to that network should be running.**

# docker network rm mybridge01

mybridge01

# docker network ls

NETWORK ID NAME DRIVER SCOPE

f26f0ee26036 bridge bridge local

12f2a1445904 host host local

22e7bb9456ca none null local

**Note: bridge – host level networking, overlay – clustering**

**>>Docker – Logging**

Docker has logging mechanisms in place which can be used to debug issues as and when they occur. There is logging at the daemon level and at the container level. Let’s look at the different levels of logging.

Daemon Logging

At the daemon logging level, there are four levels of logging available −

* Debug − It details all the possible information handled by the daemon process.
* Info − It details all the errors + Information handled by the daemon process.
* Errors − It details all the errors handled by the daemon process.
* Fatal − It only details all the fatal errors handled by the daemon process.

Please follow below steps to enable the logging.

Step 1 – Stop the docker daemon if it is already running.

# systemctl stop docker

Step 2 – now start the docker daemon with following options as shown below:

# dockerd -l debug &

**Note: if it will give you storage driver error then you need to use –s for specifying the storage driver.**

# dockerd -l debug -s=overlay2 &

Now, if you execute any Docker command such as docker images, the Debug information will also be sent to the console.

# docker ps

DEBU[0200] Calling GET /\_ping

DEBU[0200] Unable to determine container for /

DEBU[0200] {Action=\_ping, Username=root, LoginUID=0, PID=12047}

DEBU[0200] Calling GET /v1.26/containers/json

DEBU[0200] Unable to determine container for containers

DEBU[0200] {Action=json, Username=root, LoginUID=0, PID=12047}

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

# docker ps -a

DEBU[0203] Calling GET /\_ping

DEBU[0203] Unable to determine container for /

DEBU[0203] {Action=\_ping, Username=root, LoginUID=0, PID=12053}

DEBU[0203] Calling GET /v1.26/containers/json?all=1

DEBU[0203] Unable to determine container for containers

DEBU[0203] {Action=json, Username=root, LoginUID=0, PID=12053}

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

658a50968395 docker.io/centos "/bin/bash" 46 minutes ago Exited (137) 24 minutes ago nervous\_chandrasekhar

65a41e472111 centos:latest "/bin/bash" 7 days ago Exited (0) 16 minutes ago hardcore\_goodall

4aad9df53e7a centos:latest "/bin/bash" 7 days ago Exited (137) 7 days ago optimistic\_pare

fc56f1cc0a03 centos:latest "/bin/bash" 7 days ago Exited (0) 7 days ago musing\_pike

1adb88b04f01 centos:latest "/bin/bash" 7 days ago Exited (255) 31 hours ago voltest2

1d58a67ba35d centos:latest "/bin/bash" 8 days ago Exited (255) 31 hours ago voltest1

**>>>Logging**

Logging is also available at the container level.

# docker logs containerID

# docker logs 1d58a67ba35d

[root@1d58a67ba35d /]# ls

anaconda-post.log bin dev etc home lib lib64 media mnt mydata opt proc root run sbin srv sys tmp usr var

[root@1d58a67ba35d /]# cd mydata/

[root@1d58a67ba35d mydata]# ls

first second

[root@1d58a67ba35d mydata]# ll

total 4

-rw-r--r-- 1 root root 0 Apr 16 10:12 first

drwxr-xr-x 2 root root 4096 Apr 16 10:12 second

[root@1d58a67ba35d mydata]# PQ

bash: PQ: command not found

**>>>Events**

# docker events (now do any activity and it will be visible in docker events)

# docker events –since ‘1h’ (events done in last one hour)

# docker events –filter event=attach (events will be shown once container got attached)

# docker events –filter=die –filter=stop (you can add multiple filters)

**>>Docker – Compose**

Docker Compose is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

Download the necessary file for the docker-compose command:

# curl -L https://github.com/docker/compose/releases/download/1.11.2/docker-compose-Linux-x86\_64 -o /usr/local/bin/docker-compose

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 617 0 617 0 0 426 0 --:--:-- 0:00:01 --:--:-- 426

100 8066k 100 8066k 0 0 330k 0 0:00:24 0:00:24 --:--:-- 826k

# ls -l /usr/local/bin/docker-compose

-rw-r--r-- 1 root root 8260272 May 3 12:37 /usr/local/bin/docker-compose

We need to make it executable.

# chmod +x /usr/local/bin/docker-compose

# ls -l /usr/local/bin/docker-compose

-rwxr-xr-x 1 root root 8260272 May 3 12:37 /usr/local/bin/docker-compose

# docker-compose version

docker-compose version 1.11.2, build dfed245

docker-py version: 2.1.0

CPython version: 2.7.13

OpenSSL version: OpenSSL 1.0.1t 3 May 2016

[root@docker ~]#

**>>>Create your first docker compose file**

Now let’s go ahead and create our first Docker Compose file. All Docker Compose files are YAML files. You can create one using the vim editor. So execute the following command to create the compose file:

# cat docker-compose.yml

version: '2'

services:

databases:

image: mysql

ports:

- "3306:3306"

environment:

- MYSQL\_ROOT\_PASSWORD=password

- MYSQL\_USER=user

- MYSQL\_PASSWORD=password

- MYSQL\_DATABASE=demodb

web:

image: nginx

Let’s take a close look at the various details of this file:

* The database and web keyword are used to define two separate services. One will be running our mysql database and the other will be our nginx web server.
* The image keyword is used to specify the image from dockerhub for our mysql and nginx containers
* For the database, we are using the ports keyword to mention the ports that need to be exposed for mysql.
* And then, we also specify the environment variables for mysql which are required to run mysql.

Now let’s run our Docker Compose file using the following command:

# docker-compose up

Pulling databases (mysql:latest)...

Trying to pull repository docker.io/library/mysql ...

latest: Pulling from docker.io/library/mysql

f2aa67a397c4: Pull complete

1accf44cb7e0: Pull complete

2d830ea9fa68: Pull complete

740584693b89: Pull complete

4d620357ec48: Pull complete

ac3b7158d73d: Pull complete

a48d784ee503: Pull complete

f122eadb2640: Pull complete

3df40c552a96: Pull complete

da7d77a8ed28: Pull complete

f8aaeb9a4f41: Pull complete

5422e3698c48: Pull complete

Digest: sha256:04ff0d061fb05f5849415f644806584cd4a0a0baac07c8d9a8ad71fd629f845c

Status: Downloaded newer image for docker.io/mysql:latest

Creating compose\_databases\_1

Creating compose\_web\_1

Attaching to compose\_web\_1, compose\_databases\_1

^CGracefully stopping... (press Ctrl+C again to force)

Stopping compose\_databases\_1 ...

Stopping compose\_web\_1 ...

Killing compose\_databases\_1 ... done

Before killing this you can see it was running.

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ff5d285e4b3a mysql "docker-entrypoint..." About a minute ago Up About a minute 0.0.0.0:3306->3306/tcp compose\_databases\_1

33459c1f47d2 nginx "nginx -g 'daemon ..." About a minute ago Up About a minute 80/tcp compose\_web\_1

**>>Docker – Setting Node.js**

Node.js is a JavaScript framework that is used for developing server-side applications. It is an open source framework that is developed to run on a variety of operating systems. Node.js is a popular framework for development. Below are the steps:

First pull an image from docker hub named node.

# docker pull node

Using default tag: latest

Trying to pull repository docker.io/library/node ...

latest: Pulling from docker.io/library/node

f2b6b4884fc8: Pull complete

4fb899b4df21: Pull complete

74eaa8be7221: Pull complete

2d6e98fe4040: Pull complete

452c06dec5fa: Pull complete

7b3c215894de: Pull complete

cc0e8883f3a1: Pull complete

c2f5b6c538cd: Pull complete

Digest: sha256:77179a1b2c4618d2d7abc292cd949b262e0b5d0b5443d00f0132c5209bc551de

Status: Downloaded newer image for docker.io/node:latest

Just create a simple node.js file as shown below:

# cat helloworld.js

console.log('hello world')

To run our Node.js script using the Node Docker container, we need to execute the following statement:

# docker run -it --rm --name=helloworld -v "$PWD":/docker/nodejs/ -w /docker/nodejs/ node node helloworld.js

hello world

The following points need to be noted about the above command:

* The –rm option is used to remove the container after it is run.
* We are giving a name to the container called “HelloWorld”.
* We are mentioning to map the volume in the container which is /usr/src/app to our current present working directory. This is done so that the node container will pick up our HelloWorld.js script which is present in our working directory on the Docker Host.
* The –w option is used to specify the working directory used by Node.js.
* The first node option is used to specify to run the node image.
* The second node option is used to mention to run the node command in the node container.
* And finally we mention the name of our script.

We will then get the following output. And from the output, we can clearly see that the Node container ran as a container and executed the HelloWorld.js script.

**>>Docker – Cloud**

The Docker Cloud is a service provided by Docker in which you can carry out the following operations:

* Nodes − you can connect the Docker Cloud to your existing cloud providers such as Azure and AWS to spin up containers on these environments.
* Cloud Repository − Provides a place where you can store your own repositories.
* Continuous Integration − Connect with Github and build a continuous integration pipeline.
* Application Deployment − Deploy and scale infrastructure and containers.
* Continuous Deployment − Can automate deployments.

Below is the URL:

[**https://cloud.docker.com**](https://cloud.docker.com/)

You can link your Docker Cloud to your Existing Cloud like AWS, Azure or etc.