**Docker Commands**

1. **Install docker on ubuntu**

* **Uninstall docker first**  
  Older versions of Docker were called docker, docker.io, or docker-engine. If these are installed, uninstall them:  
  **$ sudo apt-get remove docker docker-engine docker.io containerd runc**
* **Set up the repository**Update the apt package index and install packages to allow apt to use a repository over HTTPS: **$ sudo apt-get update**

**$ sudo apt-get install \**

**ca-certificates \**

**curl \**

**gnupg \**

**lsb-release**

* **Add Docker’s official GPG key:  
  $ sudo mkdir -p /etc/apt/keyrings**

**$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg**

* **Use the following command to set up the repository:  
  echo \**

**"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \**

**$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

* **Install Docker Engine**Update the apt package index, and install the *latest version* of Docker Engine, containerd, and Docker Compose, or go to the next step to install a specific version:  
  **$ sudo apt-get update**

**$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin**

* **Verify that Docker Engine**   
  is installed correctly by running the hello-world image.  
  **$ sudo docker run hello-world**

1. **Uninstall docker**

* Uninstall the Docker Engine, CLI, Containerd, and Docker Compose packages:  
  **$ sudo apt-get purge docker-ce docker-ce-cli containerd.io docker-compose-plugin**

1. **Check docker version**

* **Docker version**  
  This command is used to get the currently installed version of docker  
  **$ sudo docker –version**

1. **Docker info**

* **Docker info**  
  Get detailed information about docker installed on the system including the kernel version, number of containers and images, etc.  
  **$ sudo docker info**

1. **Docker image**

* **Docker pull  
  Usage: docker pull <image name>**  
  This command is used to pull images from the **docker repository**(hub.docker.com)  
  **$ sudo docker pull ubuntu**
* **Docker push  
  Usage: docker push <username/image name>**  
  This command is used to push an image to the docker hub repository  
  **$ sudo docker push <username/image name>**
* **Docker images  
  Usage: docker images**   
  This command lists all the locally stored docker images  
  **$ sudo docker images**
* **Docker rmi  
  Usage: docker rmi <image-id>**This command is used to delete an image from local storage  
  **$** **sudo docker rmi <image-id>**
* **Docker build  
  Usage: docker build –t <path to docker file>**This command is used to build an image from a specified docker file  
  **$ sudo** **docker build <path to docker file>**
* **Searching image  
  Usage: docker search <image>**Search for a docker image on dockerhub with the name mentioned in the command.  
  **$ docker search <image>**

1. **Docker container**

* **Docker run  
  Usage: docker run -it -d –-name containername <image name>**  
  This command is used to create a container from an image  
  **$ sudo docker run -it –d -–name ubuntu:latest  
  $ sudo docker run –-name –d –p 8080:80 ubuntu:latest  
  -d** **:** is to run the container in the background  
  **-p :**is to publish the Port 80 of the container to the 8080 port of the host so that we can access the website in our host browser( in my case MAC) so the URL
* **Docker ps**This command is used to list the running containers  
  **$ sudo docker ps**
* **Docker ps –a**This command is used to show all the running and exited containers  
  **$ sudo docker ps –a**
* **Docker start  
  Usage: docker start <container id> bash**  
  This command is used to access the running container  
  **$ sudo docker start <container id> bash**
* **Docker exec   
  Usage: docker exec -it <container id> bash**  
  This command is used to access the running container  
  **$ sudo docker exec -it <container id> bash**
* **Docker stop  
  Usage: docker stop <container id>**  
  This command stops a running container.  
  **$ sudo docker stop <container id>**
* **Docker kill  
  Usage: docker kill <container id>**  
  This command kills the container by stopping its execution immediately. The difference between ‘docker kill’ and ‘docker stop’ is that ‘docker stop’ gives the container time to shutdown gracefully, in situations when it is taking too much time for getting the container to stop, one can opt to kill it  
  **$ sudo docker kill <container id>**
* **Docker commit  
  Usage: docker commit <conatainer id> <username/imagename>**  
  This command creates a new image of an edited container on the local system  
  **$ sudo docker commit <conatainer id> <username/imagename>**
* **Docker rm  
  Usage: docker rm <container id>**This command is used to delete a stopped container  
  **$ sudo docker rm <container id>**
* **Docker restart   
  Usage: docker restart <container id>**Restart the docker container with container id mentioned in the command.  
  **$ sudo docker restart <container id>**

1. **Volume**

* **Create a volume**  
  Create a volume which docker container will use to store data.  
  **$ sudo docker volume create**Run the below command if the volume got created or not. **$ sudo docker volume ls**
* **$ sudo docker volume ls**
* **$ sudo docker volume create <volume name>**
* **$ sudo docker volume rm <volume name>**
* **$ sudo docker volume prune (remove all volumes)**
* **$ sudo docker volume inspect <volume name>**
* **$ sudo pwd (to check directory)**

1. **Copying file**

* Copy a file from a docker container to the local system.  
  In this example, I am copying httpd.pid file inside a docker container with id 09ca6feb6efc to /home/geekflare/  
  **$ sudo docker cp <source><destination>**Run the command below to check if the file got copied or not.  
  **$ sudo ls**

1. **Create file**

* This command create file with extension.  
  **$ sudo touch filename.extention**

1. **Open file**

* This command open file with extension.  
  **$ sudo open filename.extention  
  $ sudo vi filename   
  (In vi insert data press “i” exit press esc + ctrl + qw / esc + ctrl + :qw!)**

1. **Checking**

* **Checking history**  
  Shows the history of a docker image with the image name mentioned in the command.  
  **$ docker history <image>**
* **Checking log**Show the logs of the docker container with contained id mentioned in the command.  
  **$ sudo docker logs <container-id>**

1. **Docker Update**

* **Updating configuration**Update container configurations. This shows all the update options.  
  **$ sudo docker update –help**Run the below command to update the CPU configuration of docker container with container id mentioned in the command.  
  **$ docker update -c 1 2f6fb3381078**

1. **Create a virtual host file**

* Create a file named **default-website.conf** and paste the following content. This is a minimal version of a virtual host file for apache. you can update the values like ServerName and alias as per your requirement.  
  **<VirtualHost \*:80>**

ServerName devopsjunction.com

ServerAlias [www.devopsjunction.com](http://www.devopsjunction.com)

ServerAdmin admin@devopsjunction.com

DocumentRoot /var/www/html/devopsjunction

**<Directory "/var/www/html/devopsjunction">**

Order allow,deny

AllowOverride All

Allow from all  
  
Require all granted

**</Directory>**

ErrorLog ${APACHE\_LOG\_DIR}/devopsjunction-error.log

CustomLog ${APACHE\_LOG\_DIR}/devopsjunction-access.log combined

**</VirtualHost>**

1. **Create a simple index.html file for our website**Now for our website, we need a welcome page, So you write one or use the following file and save in the name index.html on the same directory.  
   **<html>  
   <head>   
   <title>** DevOpsJunction - Home Page **</title>**   
   **</head>  
   <body>  
   <h2>** Welcome to Devops Junciton. Good to have you here **</h2>**  
   **</body>  
   </html>**
2. **Dockerfile structure**

* **FROM**  
  The Dockerfile FROM command specifies the base image of your Docker images. If you want to start with a bare Linux image, you can use this FROM command:

**# The base image  
FROM ubuntu:latest**

* **CMD**  
  The CMD command specifies the command line command to execute when a Docker container is started up which is based on the Docker image built from this Dockerfile. Here are a few Dockerfile CMD examples:

**CMD echo Docker container started.**

This example just prints the text Docker container started when the Docker container is started. The next CMD example runs a Java application:

**CMD java -cp /myapp/myapp.jar com.jenkov.myapp.MainClass arg1 arg2 arg3**

* **MAINTAINER**

The Dockerfile MAINTAINER command is simply used to tell who is maintaining this Dockerfile. Here is an example:

**MAINTAINER Joe Blocks <joe@blocks.com>**

The MAINTAINER instruction is not often used though, since that kind of information is also often available in GIT repositories and elsewhere.

* **COPY**

The Dockerfile COPY command copies one or more files from the Docker host (the computer building the Docker image from the Dockerfile) into the Docker image. The COPY command can copy both a file or a directory from the Docker host to the Docker image. Here is a Dockerfile COPY example:  
  
**COPY /myapp/target/myapp.jar /myapp/myapp.jar**

This example copies a single file from the Docker host at /myapp/target/myapp.jar to the Docker image at /myapp/myapp.jar . The first argument is the Docker host path (where to copy from) and the second argument is the Docker image path (where to copy to). You can also copy a directory from the Docker host to the Docker image. Here is an example:

**COPY /myapp/config/prod /myapp/config**

This example copies the directory /myapp/config/prod from the Docker host to the /myapp/config directory in the Docker image. You can als copy multiple files into a single directory in the Docker image using the COPY command. Here is an example:

**COPY /myapp/config/prod/conf1.cfg /myapp/config/prod/conf2.cfg /myapp/config/**  
  
This example copies the two files /myapp/config/prod/conf1.cfg and /myapp/conig/prod/conf2.cfg into the Docker image directory /myapp/config/ . Notice how the destination directory has to end with a / (slash) for this to work.

* **ADD**

The Dockerfile ADD instruction works in the same way as the COPY instruction with a few minor differences:  
The ADD instruction can copy and extract TAR files from the Docker host to the Docker image. The ADD instruction can download files via HTTP and copy them into the Docker image.Here are a few Dockerfile ADD examples:  
  
**ADD myapp.tar /myapp/**  
  
This example will extract the given TAR file from the Docker host into the /myapp/ directory inside the Docker image.Here is another example:

**ADD http://jenkov.com/myapp.jar /myapp/**

* **ENV**

The Dockerfile ENV command can set an environment variable inside the Docker image. This environment variable is available for the application that is started inside the Docker image with the CMD command. Here is an example:

**ENV MY\_VAR 123**

This example sets the environment variable MY\_VAR to the value 123

* **RUN**

The Dockerfile RUN command can execute command line executables within the Docker image. The RUN command is executed during build time of the Docker image, so RUN commands are only executed once. The RUN command can be used to install applications within the Docker image, or extract files, or other command line activities which are necessary to run once to prepare the Docker image for execution.

**RUN apt-get install some-needed-app**

* **ARG**

The Dockerfile ARG instruction lets you define an argument which can be passed to Docker when you build the Docker image from the Dockerfile. Here is an example:

**ARG tcpPort**

When you run the Docker command to build the Dockerfile containing the above ARG instruction, you can pass an argument to the tcpPort argument like this:

**docker build --build-arg tcpPort=8080 .**

Notice the --build-arg followed by the tcpPort=8080 . This part sets the tcpPort argument value to 8080. You can define multiple build arguments using multiple ARG instructions. Here is an example:

**ARG tcpPort**

**ARG useTls**

When building the Docker image you must provide values for all of the build arguments. You do so by repeating the --build-arg sections for each argument you want to set. Here is an example:

**docker build --build-arg tcpPort=8080 --build-arg useTls=true** .

You can set a default value for an ARG so that it becomes optional to provide a value for it when building the Docker image. If the argument is not given a value, it will be given its default value. Here is an example:

**ARG tcpPort=8080**

**ARG useTls=true**

If neither the tcpPort nor the useTls argument is set when building the Docker image for the Dockerfile containing the above ARG instructions, their argument values will be set to 8080 and true. Arguments declared by ARG are typically referenced elsewhere in your Dockerfile. You reference an ARG argument like this:

**ARG tcpPort=8080**

**ARG useTls=true**

**CMD start-my-server.sh -port ${tcpPort} -tls ${useTls}**

Notice the two references ${tcpPort} and ${useTls}. These refer to the declared ARG arguments named tcpPort and useTls .

**docker build --build-arg tcpPort=8080**

* **WORKDIR**

The WORKDIR instruction specifies what the working directory should be inside the Docker image. The working directory will be in effect for all commands following the WORKDIR instruction. Here is an example:

**WORKDIR /java/jdk/bin**

* **EXPOSE**

The Dockerfile EXPOSE instruction opens up network ports in the Docker container to the outside world. For instance, if your Docker container runs a web server, that web server will probably need port 80 open for any client to be able to connect to it. Here is an example of opening a network port using the EXPOSE command:

**EXPOSE 8080**

You can also set which protocol is allowed to communicate on the opened port. For instance, UDP or TCP. Here is an example of setting the allowed protocol also:

**EXPOSE 8080/tcp 9999/udp**

If no protocol is set (after the / ) , then the protocol is assumed to be TCP.

* **VOLUME**

The Dockerfile VOLUME instruction creates a directory inside the Docker image which you can later mount a volume (directory) to from the Docker host. In other words, you can create a directory inside the docker image, e.g. called /data which can later be mounted to a directory, e.g. called /container-data/container1 in the Docker host. The mounting is done when the container is started up. Here is an example of defining a volume (mountable directory) in a Dockerfile using the VOLUME instruction:

**VOLUME /data**

* **ENTRYPOINT**

The Dockerfile ENTRYPOINT instruction provides an entrypoint for Docker containers started from this Docker image. An entrypoint is an application or command that is executed when the Docker container is started up. In that way, ENTRYPOINT works similarly to CMD, with the difference being that using ENTRYPOINT the Docker container is shut down when the application executed by ENTRYPOINT finishes. Thus, ENTRYPOINT kind of makes your Docker image an executable command itself, which can be started up and which shut down automatically when finished. Here is a Dockerfile ENTRYPOINT example:

**ENTRYPOINT java -cp /apps/myapp/myapp.jar com.jenkov.myapp.Main**

This example will execute the Java application main class com.jenkov.myapp.Main when the Docker container is started up, and when the application shuts down, so does the Docker container.

1. **Create the Dockerfile**Create a Dockerfile with the following content on the same directory.  
   **# Define the base image on top of which we are going to customize  
   FROM** ubuntu:latest

**# Define the Createor and Maintainer of this image  
MAINTAINER** Sarav AK <aksarav@middlewareinventory.com>

**# Run the command to install apache as a prerequisite  
RUN** apt-get update && apt-get install -y apache2

**# Run the command to Create a DocumentRoot  
RUN** mkdir -p /var/www/html/devopsjunction

**# Copy the file from the Current Host Directory to the Container  
ADD** default-website.conf /etc/apache2/sites-available/  
**ADD** index.html /var/www/html/devopsjunction

**# Run one more command to change the ownership of the file we copied  
RUN** chown root:root /etc/apache2/sites-available/default-website.conf

**# Enable this website using a2ensite – command  
RUN** a2ensite default-website

**# Say that our application is exposing a Port  
EXPOSE** 80

**# the Default command and the Command to start the app in our container  
CMD** ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

1. **Docker login**

* **Docker login**This command is used to login to the docker hub repository  
  **$ sudo docker login**

1. **Docker network**

* **Docker network**The following command in docker lists the details of all the network in the cluster.  
  **$ sudo docker network ls**There are several other docker network commands.  
  **$ sudo docker network**

1. **Installing plugin**

* Install a docker plugin vieux/sshfs with debug environment set to 1. **$ sudo docker plugin install vieux/sshfs DEBUG=1**

1. **Logout**

* Logging out from dockerhub. **$ sudo docker logout**