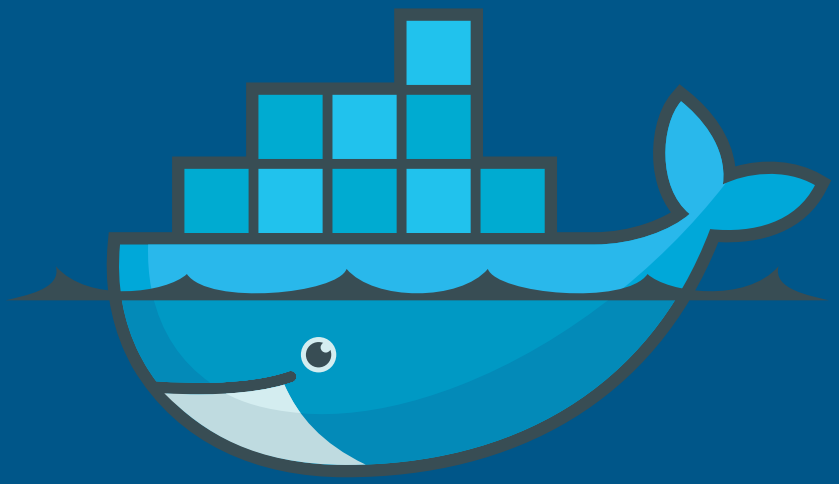
RapidValue™
Driving Digital Excellence

A Lightweight Virtualization Platform for Developers



docker

A RapidValue Solutions Whitepaper

Table of Contents

Executive Summary

Docker is a platform for developers and system administrators, which allows developing, running and deploying applications. Docker lets you, quickly, assemble applications from components and convert them into containers. It allows you to get your code tested and deployed into production in a prompt manner.

Unlike virtual machines, containers do not have high overheads and hence, enable more efficient usage of the underlying system and resources. The benefit of Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development. Docker, once installed on any OS/ Server, can run the containers on it. The containers in Docker, usually, contain the following components:

- The cut down version of most OS like Ubuntu, Centos etc.
- The dependencies that are required for the application to run.
- The actual application files along with developer code.
- The database files.

For the above components, you can use separate containers or the same containers. The power of Docker is displayed when you are able to link separate containers of the application with the database containers. By doing so, you are able to maintain different variants of the database for testing, production etc. It will only, take a single command to change the database containers, without making any changes to the configuration of the container, which contains the application data.

This whitepaper by RapidValue Solutions explains the step-by-step installation of Docker in devices with different OS, also, addresses the advantages and usefulness of Docker. Docker makes it easier to create, deploy, and run applications by using Containers. Containers allow a developer to package up an application. Docker is a tool that is designed to benefit, both, the developers and system administrators. The developers can focus on writing code without worrying about the system that it will, ultimately, be running on. Docker gives flexibility and also, potentially reduces the number of systems needed.

Step-by-Step Installation

Docker Engine is supported on Linux, Cloud, Windows and OS X. The detailed steps used for the installation of Docker in different OS are as follows:

Basic Commands

1. RUN

   This will pull an image from the Docker hub and run that image.

   ```
   docker run hello-world
   ```

   Also, you can run an image that already exists in our system.

   ```
   docker run -i -t hello-world
   ```

2. TAG

   This is used to tag an image into a repository. It will help you to create a new version of a repository. So first you can use the above "RUN" command to download the image from dockerhub, if you need a new version of a Docker-whale repository.

   ```
   docker tag 7d9495d03763 maryatdocker/docker-whale:latest
   ```

   In this, "`7d9495d03763`" is the image id, `maryatdocker` is the registry host username, `docker-whale:latest` is the image name with the latest tag.

3. PUSH

   This command is used to push an image or a repository to the registry. The changes should be committed before doing the push operation.

   ```
   docker push maryatdocker/docker-whale
   ```

   In this, `maryatdocker` is the hosted username, `docker_whale` is the image name that you want to push.

4. PULL

   You can pull an image or a repository from the registry with this command.

```
docker pull maryatdocker/docker-whale
```

In this, `maryatdocker` is the hosted username, `docker_whale` is the image name that you want to pull.

5. PS

This command will list the entire running containers from your local system.

```
docker ps
```

It will list all the running containers.

```
docker ps -a
```

It will list all the containers, including stopped and running containers.

6. Commit

This is used to commit the changes of an image.

```
docker commit container_id  maryatdocker/ubuntu:version-1
```

Here, maryatdocker is the hosted username,

`ubuntu:version-1` is the image name with the `version-1` tag

7. Stop

Command which is used to stop a given container or stop all the container at once.

```
docker stop $(docker ps -a -q)
```

It will stop all the running containers.

```
docker stop 1a45f654s
```

It will stop the container 1a45f654s, where 1a45f654s is the id of the container.

8. Rm

This is used to remove the stopped container or a given container. Container should stop

before getting removed from the system.

```
docker rm $(docker ps -a -q)
```

It will remove all the stopped containers.

```
docker rm 1a45f654s
```

It will remove container 1a45f654s.

9. Rmi

The command is used to remove all the images of a given image, using the image id.

```
docker rmi $(docker images -q)
```

It will remove all the images.

```
docker rmi -f 12254
```

It will remove an image with an id of 12254.

10. Exec

After running a container in order to execute commands inside it we can use the command Exec.

```
docker exec -it <imagename> bash
```

You can, also, pass running tasks with the exec command.

```
docker exec <container_id> echo "Hello from container!"
```

Many more Docker management commands, Image commands, Container commands, Hub and registry commands, Network and connectivity commands, and Shared data volume commands are listed here.

The Usage and Benefits

01. It is a collaborative working system. If you have QA, System administrators, developers and release engineers, all can work together to get the code into production.

02. You can create a standard container format that lets developers focus only on their applications inside containers, while system administrators and operators can work on running the container in our deployment. This segregation of tasks simplifies the management and deployment of code.

03. The platform supports easy implementation of new containers, makes it easy to apply the changes and increases the visibility of the changes. These help to understand how an application works and how it is built.

04. It is a simple platform to use, for development, testing and deployment of applications.

05. As the Docker runs on many platforms, it is easy to move the applications around. You can, quickly move an application from a testing environment into the Cloud and back, whenever required.

06. Certain Opensource projects are developed on principle of an application executing a single task and running that task perfectly and efficiently. In docker this principle is implemented such that in a running container only one process will be allowed to run i.e. we cannot have a single image running multiple services, but instead we have to create an image that runs on multiple linked containers and each container running only one service.

Creating Dockerfile

Dockerfile is the descriptor file which contains the command that you have to manually type into get a container running, then get the dependencies installed, along with the application.

01. Create a new folder named dockerexample inside the /opt/

02. Inside the folder create new file and name it "Dockerfile".

03. Change the working directory : `cd /opt/dockerexample`

04. Usually, when you run a command like `docker run -it ubuntu:14.04`, it downloads the image to local if not present. So, in the Dockerfile, add the line-

```
FROM ubuntu:14.04
```

05. Now in the command line build an image from this Dockerfile using the following command.

```
docker build -t dockerexample
```

The -t arguments is to specify the image name dockerexample. The dot at the end of the command is to notify the location of the file.

06. Make sure that there is no other file in the folder, otherwise it will increase the image size and slow down the process.

07. Now run:

```
docker run -it dockerexample /bin/bash
```

08. To install dependencies in to the image we can add them to the Dockerfile using the following syntax i.e.

```
<command> <statement>
```

eg. `RUN sudo apt-get update`

09. To build a complete environment with the dependencies, there is a sample code:

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y \
     nano \
     python2.7 \
     python-pip \
RUN apt-get clean
```

In this file, you first import the OS, then use the RUN command to run apt-get to install dependencies. It is advised to use the format shown in the file i.e. to use "\" to separate each dependencies name. Always run an apt-get clean command at the EOF to delete all garbage files.

Usage

01. When updating the Dockerfile on Github for a new build or on your local machine make sure to add it, using a new RUN command, at the end of the file. Docker uses cache to build images, so it compares the Dockerfile with the images currently present in the host from the top to bottom line by line. Hence, always add at the end so that it does not re-run the previous commands.

02. In order to maintain different versions from the same Dockerfile, you tag the images of the same name using -t flag and adding tags at the end.

```
docker build -t dockerexample:v2.9
```

Then, commit the image and push it to Dockerhub.

Docker Hub

All images that you download are taken from an online repository called Docker Hub. It hosts a myriad of images which contains self-running containers, containing the application. For example, you can download and run the MySQL container, which runs by itself, without any OS containers. You need to create an account.

01. Sign-up for a new account.

02. Open Docker Hub on your profile page.

03. Choose Create Repository.

04. Give Repository Name and Short Description and set as public repo.

05. Start an existing image and then make changes to the files and then push it as follows:

```
docker run -it ubuntu:14.04 bash
```

06. Enter the container and run:

```
apt-get install python python-pip
mkdir app
cd /app
touch test_file.txt
```

07. Open a new terminal and run:

```
docker ps -a
```

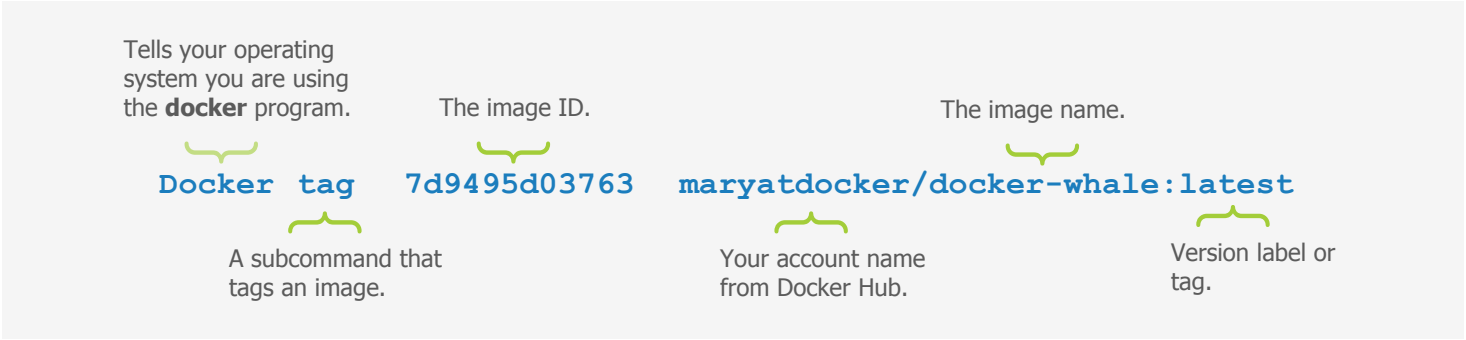
08. This will list all the containers that are running. Then, look at the name of the container that you are currently running. A shortcut can be used

```
docker ps -l
```

This will display the last running container. Then, get the ID of the container.

09. Now, run the command in terminal.

```
docker commit <container-id>  <username>/ubuntu:v2.0
```



10. Now, push the image to Dockerhub.

```
docker push <username>/ubuntu:v2.0
```

Github

Dockerhub has the functionality to connect to Github repositories to create automated builds.

01. First, create a new Github repo with name "`docker-ubuntu`".

02. Then, add a file name "`dockerfile`".

03. Copy the code from the above example and paste it.

04. Then, commit to Github.

05. Log into Dockerhub.

06. Go to the upper right corner, there is a link to "create automated builds".

07. Then, select Github from the screen that gets displayed.

08. It will then, display all the repos from your Github account. Now, select "`docker-ubuntu`" from the list and click create.

09. This will, automatically, build an image when you push any changes to the Docker file on Github. This simplifies the process of building on the local machine and then, pushing to the server.

Docker Volumes

As the data in Docker containers is lost when the container is stopped, in order to save the state of the container, you have to commit it every time. You can use Docker volumes to persist the data. Docker volumes is a specially designed directory, within one or more containers, which bypasses the Docker images file system i.e. it replaces a directory on the container with a directory from the host machine. Hence, it mirrors any changes that are done at both places.

01. Adding data volume

a. You can use -v flag to mount volumes with Docker. While creating containers, you can use -v multiple times to mount multiple volumes.

```
docker run -d -P --name web -v /webapp training/webapp python app.py
```

The above command mounts the container directory `/webapp` to docker default mount directory. If you want to mount to a specific location, you can use this format.

```
docker run -v host-dir:container-dir
```

b. The container-dir should be absolute path like `/src/webapp` and the host-dir can either be absolute path or name of the volume. If you specify absolute path, the Docker mounts to the path you specify and if you supply a name, Docker creates a named volume in the Docker application folder. Other containers can, also, mount this volume, using the same volume name.

c. You can use Docker inspect command to locate the volume's information.

```
$ docker inspect web
```

02. Mount host directory as data volume

a. This command mounts the host directory, `/src/webapp`, into the container at `/opt/webapp`

```
docker run -d -P --name web -v /src/webapp:/opt/webapp
training/webapp python app.py
```

b. Docker volumes default to mount in read-write mode, but you can also, set it to be mounted read-only.

```
$ docker run -d -P --name web -v /src/webapp:/opt/webapp:ro
training/webapp python app.py
```

3. Creating and mounting a data volume container

a. Create a named Data Volume Container. This one is mounting postgreSQL data volume with a name dbdata. This volume can be used in other containers with the name specified.

```
docker create -v /dbdata --name dbdata training/postgres /bin/true
```

b. You can then use the `--volumes-from` flag to mount the `/dbdata` volume in another container.

```
docker run -d --volumes-from dbdata --name db1 training/postgres
```

c. You can use multiple `--volumes-from` parameters to bring together multiple data volumes from multiple containers. The volumes can be mounted from different locations.

d. If you want to remove volumes you have to use `docker rm -v` command on the last container with reference to the volume or the volume stays in dangling state and uses the disk space.

e. Another useful function you can perform with volumes is that you can use them for backups, restores or migrations. You can do this by using the `--volumes-from` flag to create a new container that mounts that volume, like so:

f. This one creates a dbdata volume backup in current directory.

```
docker run --volumes-from dbdata -v $(pwd):/backup ubuntu tar cvf
/backup/backup.tar /dbdata
```

g. Shared volumes could be dangerous unless the application using the shared volumes is designed to prevent data corruption. You can also, specify containers to be read only to reduce chances of corruption.

Linking Containers

You can link different containers together and easily, switch the database environments. Below are the steps explaining how you can create a Django project and connect different database with it.

1. **Step-1**

a. Create a new Dockerfile in the project directory named `docker-blog-os` and the Dockerfile contains the following:

```
FROM ubuntu
Run apt-get update && apt-get install -y \
python \
python-pip \
python-virtualenv
RUN mkdir /blog
WORKDIR /blog
ADD requirements.txt / blog
RUN pip install -r requirements.txt
ADD ./blog/
```

b. In this image you keep only the necessary OS and other supporting files that are needed for the project blog. In the directory `docker-blog-os`, you need to create a requirements.txt, which will contain the following:

```
django==1.9
mysql-python
```

c. It will get installed automatically, while building the above image. You build this image by using the build command.

```
docker build -t blog-dependencies
```

d. This will create a new image called blog-dependencies.

2. **Step-2**

a. Create a new directory `docker-blog-db` and add Dockerfile

b. Dockerfile, which contains the following contents, will be a MySQL image

```
FROM mysql
```

c. Then, run the build command-

```
docker build -t blog-mysql-development-db
```

d. This will create a new image called `blog-mysql-development-db` and you need to keep the development version of database in this image.

e. Similar to this, you can create another image for keeping the production database. The same steps are followed here, also.

3. Step 3

a. In this step, you run the MySQL image and link this with the project image. So, you have two running containers.

```
docker run --name mysqlapp -e MYSQL_ROOT_PASSWORD="123" -d rapid/blog-mysql-development-db
```

This command allows you to run the MySQL image. In this, `mysqlapp` is the name that is given to the MySQL image. `MYSQL_ROOT_PASSWORD` is the argument for passing the MySQL root password for authentication. `rapid/blog-mysql-development-db` here `rapid` is the owned user and `blog-mysql-development-db` is the image name.

b. After running the dB container, you need to run another container to link it to dB container by using the `--link` tag to create a database in MySQL of the dB container.

```
docker run -it --link=mysqlapp:blog-mysql-development-db --rm blog-mysql-development-db sh -c 'exec mysqladmin -h$MYSQL_PORT_3306_TCP_ADDR -P$MYSQL_PORT_3306_TCP_PORT -uroot -p create myblog'
```

After running this command and creating database, it will exit the container. This is done in order for Django to have a database called "`myblog`".

4. Step 4

a. Now, you can run the Django application.

```
docker run -it -p 8000:8000 --link mysqlapp:blog-mysql-development-db blog-dependencies
```

b. This will run the Django project, using the development MySQL database.

Conclusion

Docker helps the developers by giving them the freedom to define environments, and create and deploy applications in a much faster and easier way. The platform helps the development and IT operations teams to become more agile and possess greater control as they are now, able to build, ship, and run any application, anywhere.

Deploying scalable services, more securely and reliably, on a wide variety of platforms are some of the other benefits of Docker. Application containers using Docker are gaining momentum and as technology and the associated processes mature, they will be able to address many issues and thus, make it simple for developers to build and deploy applications.

If you want more information about Docker and know the difference between Docker and virtual machines then watch this video.

If you'd like more information on this topic, please do write to the authors, Zadeek Ummer and Sakeer P. (Python and Djando experts) at zadeek.ummer@rapidvaluesolutions.com and sakeer.p@rapidvaluesolutions.com.

RapidValue has a team of domain experts and mobility consultants to help you build innovative and comprehensive mobile applications for your enterprise. If you need guidance on building your first enterprise mobile application, please write to contactus@rapidvaluesolutions.com, we'll be happy to hear from you.

About RapidValue

A global leader in digital transformation for enterprise providing end-to-end mobility, omni-channel, IoT and cloud solutions. Armed with a large team of experts in consulting, UX design, application development, integration and testing, along with experience delivering projects worldwide, in mobility and cloud, we offer a wide range of services across industry verticals. We deliver services to the world's top brands, fortune 1000 companies, Multinational companies and emerging start-ups. We have offices in the United States, the United Kingdom and India.

www.rapidvaluesolutions.com

www.rapidvaluesolutions.com/blog

+1 877.643.1850

contactus@rapidvaluesolutions.com

March 2016