CONTENTS

Getting Started With Docker

By Christopher M. Judd

ABOUT DOCKER

Almost overnight, Docker has become the de facto standard that developers and system administrators use for packaging, deploying, and running distributed applications. It provides tools for simplifying DevOps by enabling developers to create templates called images that can be used to create lightweight virtual machines called containers, which include their applications and all of their applications' dependencies. These lightweight virtual machines can be promoted through testing and production environments where sysadmins deploy and run them.

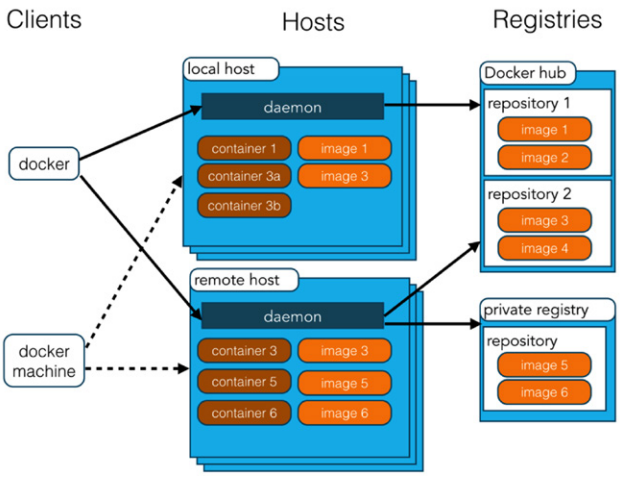
Docker makes it easier for organizations to automate infrastructure, isolate applications, maintain consistency, and improve resource utilizations.

Similar to the popular version control software Git, Docker has a social aspect, in that developers and sysadmins are able to share their images via Docker Hub.

Docker is an open-source solution that runs natively on Linux but also works on Windows and Mac using a lightweight Linux distribution and VirtualBox. Many tools have also grown up around Docker to make it easier to manage and orchestrate complex distributed applications.

DOCKER ARCHITECTURE

Docker utilizes a client-server architecture and a remote API to manage and create Docker containers built upon Linux containers. Docker containers are created from Docker images. The relationship between containers and images are analogous to the relationship between objects and classes in object-oriented programming.



Docker Images	A recipe or template for creating Docker containers. It includes the steps for installing and running the necessary software.
Docker Container	Like a tiny virtual machine that is created from the instructions found within the Docker image originated
Docker Client	Command-line utility or other tool that takes advantage of the Docker API (https://docs.docker.com/reference/api/docker_remote_api) to communicate with a Docker daemon
Docker Host	A physical or virtual machine that is running a Docker daemon and contains cached images as well as runnable containers created from images

Docker Registry	A repository of Docker images that can be used to create Docker containers. Docker Hub (https://hub.docker.com) is the most popular social example of a Docker repository.
Docker Machine	A utility for managing multiple Docker hosts, which can run locally in VirtualBox or remotely in a cloud hosting service such as Amazon Web Services, Microsoft Azure, or Digital Ocean

GETTING STARTED

INSTALLING DOCKER

For Mac and Windows the installation could not be simpler. All you need to do is download and install the Docker Toolbox found at https://www.docker.com/toolbox. The installer includes the Docker Client, Docker Machine, Compose (Mac only), Kitematic, and VirtualBox.

HOT TIP Since Docker is based on the Linux Container technologies which are not available on Mac and Windows, VirtualBox is used to run a tiny Linux kernel containing the Docker server.

At the time of this writing, installing Docker on Linux is not as easy. To install Docker on Linux you may have to install some prerequisites; check https://docs.docker.com/installation for specific instructions. For some distributions there may be packages available using its native package manager. For other distributions you will need to run:

```
curl -sSL https://get.docker.com/ | sh
```

Optionally on Linux you can install Docker-Machine as root; to do so, execute the following:

```
curl -L https://github.com/docker/machine/releases/
↳download/v0.4.0/docker-machine_linux-amd64 > /
↳usr/local/bin/docker-machine
chmod +x /usr/local/bin/docker-machine
```

If you want to create machines locally, you will also need to install VirtualBox using the instructions found at https://www.virtualbox.org/wiki/Linux_Downloads.

As of the date of this publication, Docker-Machine is still considered in beta and is not recommended for production use.

RUNNING A CONTAINER

With Docker installed you are able to begin running containers. If you don't already have the container you want to run, Docker will download the image necessary to build the container from the Docker Hub, then build and run it.

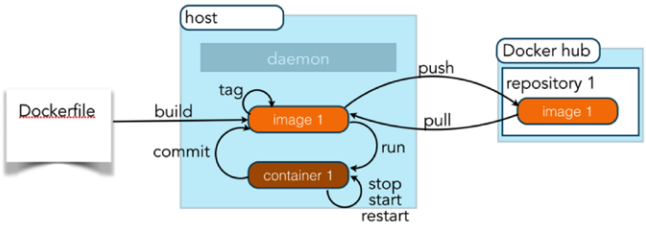
To run the simple hello-world container to make sure everything is configured properly, run the following commands:

```
docker run hello-world
```

Ultimately this command prints a message to standard output explaining all the steps that took place to display the message.

TYPICAL LOCAL WORKFLOW

Docker has a typical workflow that enables you to create images, pull images, publish images, and run containers.



The typical Docker workflow involves building an image from a Dockerfile, which has instructions on how to configure a container or pull an image from a Docker Registry such as Docker Hub. With the image in your Docker environment, you are able to run the image, which creates a container as a runtime environment with the operating systems, software, and configurations described by the image. For example, your result could be a container on the Debian operating system running a version of MySQL 5.5, which creates a specific database with users and tables required by your web application. These runnable containers can be started and stopped like starting and stopping a virtual machine or computer. If manual configurations or software installations are made, a container can then be committed to make a new image that can be later used to create containers from it. Finally, when you want to share an image with your team or the world, you can push your images to a Docker registry.

PULL IMAGE FROM DOCKER REGISTRY

The easiest way to get an image is to visit https://hub.docker.com and find an already prepared image to build a container from. There are are many certified official

accounts for common software such as MySQL, Node.js, Java, Nginx, or WordPress, but there are also hundreds of thousands of images created by ordinary people as well. If you find an image you want, such as mysql, execute the pull command to download the image.

```
docker pull mysql
```

If you don't already have the image locally, Docker will download the most current version of that image from Docker Hub and cache the image locally. If you don't want the current image and instead want a specific version, you can also add a tag to identify the desired version.

```
docker pull mysql:5.5.45
```

HOT TIP If you know you will want to run the image immediately after pulling, you can save a step by just using the run command and it will automatically pull it in the background.

BUILDING IMAGE FROM A DOCKERFILE

If you can't find what you need or don't trust the source of an image you find on Docker Hub, you can always create your own images by creating a Dockerfile. Dockerfiles contain instructions for inheriting from an existing image, where you can then add software or customize configurations.

The following is a simple example of what you might find in a file named Dockerfile:

```
FROM mysql:5.5.45
RUN echo America/New_York | tee /etc/timezone &&
↳dpkg-reconfigure --frontend noninteractive tzdata
```

This Dockerfile example shows that the image created will inherit from the certified mysql repository (specifically the 5.5.45 version of MySQL). It then runs a Linux command to update the time zone to be Eastern Time.

More details on creating a Dockerfile will be provided later.

To build this image from the directory containing the Dockerfile, run the following command:

```
docker build .
```

This command will create an unnamed image. You can see it by running the command to list images.

```
docker images
```

This displays all the locally cached images, including the ones created using the build command.

```
REPOSITORY   TAG       IMAGE ID        VIRTUAL SIZE
<none>       <none>    4b9b8b27fb42    214.4 MB
mysql        5.5.45    0da0b10c6fd8    213.5 MB
```

As you can see, the build command created an image with a repository name and tag name of <none>. This tends not to be very helpful, so you can use a `-t` option to name the image for easier usage:

```
docker build -t est-mysql .
```

Listing the images again you can see the image is much clearer.

```
REPOSITORY   TAG       IMAGE ID        VIRTUAL SIZE
est-mysql    latest    4b9b8b27fb42    214.4 MB
mysql        5.5.45    0da0b10c6fd8    213.5 MB
```

There is an alternative option to creating a custom image besides writing a Dockerfile. You can run an existing image with bash access then customize the image manually by installing software or changing configurations. When complete you can run the `docker commit` command to create an image of the running container. This is not considered a best practice since it is not repeatable or self-documenting like using the Dockerfile method.

RUNNING AN IMAGE

To run a Docker image you just need to use the `run` command followed by a local image name or one found in Docker Hub. Commonly, a Docker image will require some additional environment variables, which can be specified with the `-e` option. For long-running processes like daemons, you also need to use a `-d` option. To start the est-mysql image, you would run the following command to configure the MySQL root user's password, as documented in the Docker Hub mysql repository documentation:

```
docker run -e MYSQL_ROOT_PASSWORD=root+1 -d est-
↳mysql
```

To see the running container, you can use the Docker `ps` command:

```
docker ps
```

The `ps` command lists all the running processes, the image name they were created from, the command that was run,

any ports that software are listening on, and the name of the container.

```
CONTAINER ID   IMAGE       COMMAND
30645f307114   est-mysql   “/entrypoint.sh mysql”
PORTS          NAMES
3306/tcp       serene_brahmagupta
```

As you can see from the running processes above, the name of the container is serene_brahmagupta. This is an auto-generated name and may be challenging to maintain. So it is considered a best practice to explicitly name the container using the `--name` option to provide your name at container start up:

```
docker run --name my-est-mysql -e MYSQL_ROOT_
↳PASSWORD=root+1 -d est-mysql
```

You will notice from the `ps` output that the container is listening to port 3306, but that does not mean you are able to use the MySQL command line or MySQL Workbench locally to interact with the database, as that port is only accessible in the secure Docker environment in which it was launched. To make it available outside that environment, you must map ports using the `-p` option.

```
docker run --name my-est-mysql -e MYSQL_ROOT_
↳PASSWORD=root+1 -p 3306:3306 -d est-mysql
```

Now mysql is listening on a port that you can connect to. But you still must know what the IP address is to connect. To determine the IP address you can use the `docker-machine ip` command to figure it out.

```
docker-machine ip default
```

Using default as the machine name, which is the default machine installed with the Docker Toolbox, you will receive the IP address of the machine hosting your docker container.

With the IP address, you can now connect to MySQL using your local MySQL command line.

```
mysql -h 192.168.99.100 -u root -proot+1
```

STOPPING AND STARTING CONTAINERS

Now that you have a Docker container running, you can stop it by using the Docker `stop` command and the container name:

```
docker stop my-est-mysql
```

The entire state of the container is written to disk, so if you want to run it again in the state it was in when you shut it down, you can use the start command:

```
docker start my-est-mysql
```

TAGGING AN IMAGE

Now that you have an image that you have run and validated, it is a good idea to tag it with a username, image name, and version number before pushing it to repository. You can accomplish this by using the Docker `tag` command:

```
docker tag est-mysql javajudd/est-mysql:1.0
```

PUSH IMAGE TO REPOSITORY

Finally, you are ready to push your image to Docker Hub for the world to use or your team to use via a private repository. First, if you haven’t done so already, you will need to go https://hub.docker.com/ to create a free account. Next you need to login using the `login` command.

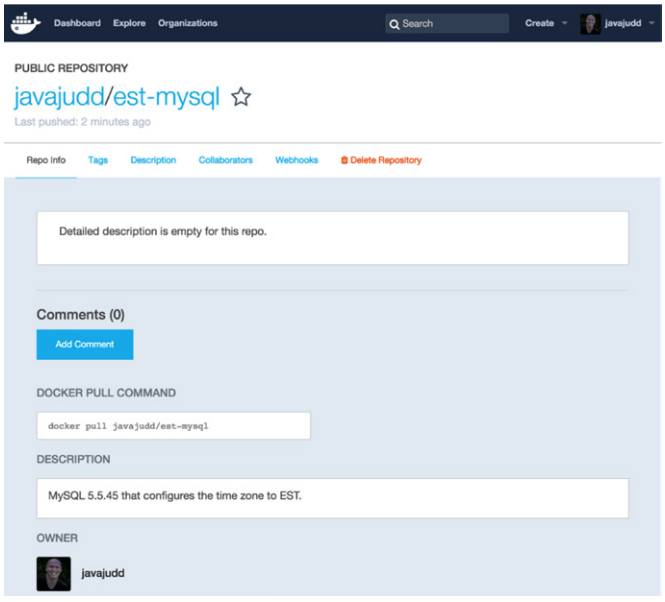
```
docker login
```

When prompted, input the username, password, and email address you registered with.

Now push your image using the `push` command, specifying your username, image name, and version number.

```
docker push javajudd/est-mysql:1.0
```

After some time you will receive a message that the repository has been successfully pushed. If you log back into your Docker Hub account, you will see the new repository.



OTHER HELPFUL COMMANDS

LIST CONTAINERS

You have already seen how the `ps` command can list the running containers, but what about all the containers, regardless of their state? By adding the `-a` option, you can see them all.

```
docker ps -a
```

With a listing of all containers, you can decide which ones to start or remove.

REMOVE CONTAINERS

When you are done using a container, rather than having it lie around, you will want to remove it to reclaim diskspace. To remove a container, you can use the `rm` command:

```
docker rm my-est-mysql
```

REMOVE IMAGES

You have already seen how the `images` command can list all the locally cached images. These images can take significant amounts of space, ranging from a megabyte to several hundred megabytes, so you will want to purge unwanted images using the `rmi` command:

```
docker rmi est-mysql
```

During the debugging cycle of creating a new image, you may generate a large amount of unwanted and unnamed images, which are denoted with a name of <none>. You can easily remove all the dangling images using the following command:

```
docker rmi $(docker images -q -f dangling=true)
```

LIST PORTS

It's often helpful to know what ports are exposed by a container, such as port 3306 for accessing a MySQL database or port 80 for accessing a web server. The `port` command can be used to display the exposed ports.

```
docker port my-est-mysql
```

LIST PROCESSES

If you need to see the processes running in a container, you can use the `top` command (similar to running the Linux `top` command):

```
docker top my-est-mysql
```

EXECUTE COMMANDS

You can execute commands in a running container using the `exec` command. To list the contents of the root of the hard drive you can, for example, do the following:

```
docker exec my-est-mysql ls /
```

If you want to ssh as root into the container, there is an equivalent exec command you can run to gain access to a bash shell, and since all the communications between the Docker client and the Docker daemon are already encrypted, it is secure.

```
docker exec -it my-est-mysql bash
```

RUN CONTAINER

The `run` command is the most complicated and featured of all the Docker commands. It can be used to do things such as manage networking settings; manage system resources such as memory, CPU, and filesystems; and configure security. Visit https://docs.docker.com/reference/run/ to see all options available.

DOCKERFILE

As you have already seen, the Dockerfile is the primary way of creating a Docker image. It contains instructions such as Linux commands for installing and configuring software. The `build` command can refer to a Dockerfile on your PATH or to a URL, such as a GitHub repository. Along with the Dockerfile, any files in the same directory or its subdirectories will also be included as part of the build process. This is helpful if you want the build to include scripts to execute or other necessary files for deployment.

HOT TIP If you wish to exclude any files or directories from being included, you have the option of using a .dockerignore file for this purpose.

INSTRUCTIONS

Instructions are executed in the order in which they are found in the Dockerfile. The Docker file can also contain line comments starting with the # character.

This table contains the list of commands available.

INSTRUCTION	DESCRIPTION
FROM	This must be the first instruction in the Dockerfile and identifies the image to inherit from
MAINTAINER	Provides visibility and credit to the author of the image

INSTRUCTION	DESCRIPTION
RUN	Executes a Linux command for configuring and installing
ENTRYPOINT	The final script or application used to bootstrap the container, making it an executable application
CMD	Provide default arguments to the ENTRYPOINT using a JSON array format
LABEL	Name/value metadata about the image
ENV	Sets environment variables
COPY	Copies files into the container
ADD	Alternative to copy
WORKDIR	Sets working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions
EXPOSE	Ports the container will listen on
VOLUME	Creates a mount point
USER	User to run RUN, CMD, and/or ENTRYPOINT instructions

DOCKERFILE EXAMPLE

This an example of the official MySQL 5.5 Dockerfile found at https://github.com/docker-library/mysql/blob/5836bc9af9deb67b68c32bebad09a0f7513da36e/5.5/Dockerfile, which uses many of the available instructions.

```
FROM debian:jessie

RUN groupadd -r mysql && useradd -r -g mysql
↳mysql
RUN mkdir /docker-entrypoint-initdb.d
RUN apt-get update && apt-get install -y perl
↳--no-install-recommends && rm -rf /var/lib/apt/
↳lists/*
RUN apt-get update && apt-get install -y libaio1
↳&& rm -rf /var/lib/apt/lists/*
RUN gpg --keyserver ha.pool.
↳sks-keyservers.net --recv-keys
↳A4A9406876FCBD3C456770C88C718D3B5072E1F5

ENV MYSQL_MAJOR 5.5
ENV MYSQL_VERSION 5.5.45

RUN apt-get update && apt-get install -y curl
↳--no-install-recommends && rm -rf /var/lib/apt/
↳lists/* \
   && curl -SL "http://dev.mysql.com/get/
↳Downloads/MySQL-$MYSQL_MAJOR/mysql-$MYSQL_
↳VERSION-linux2.6-x86_64.tar.gz" -o mysql.tar.gz \
   && curl -SL "http://mysql.he.net/Downloads/
↳MySQL-$MYSQL_MAJOR/mysql-$MYSQL_VERSION-
↳linux2.6-x86_64.tar.gz.asc" -o mysql.tar.gz.asc \
   && apt-get purge -y --auto-remove curl \
   && gpg --verify mysql.tar.gz.asc \
   && mkdir /usr/local/mysql \
   && tar -xzf mysql.tar.gz -C /usr/local/mysql
↳--strip-components=1 \
   && rm mysql.tar.gz* \
# continued ->
```

```
   && rm -rf /usr/local/mysql/mysql-test /usr/
↳local/mysql/sql-bench \
   && rm -rf /usr/local/mysql/bin/*-debug /usr/
↳local/mysql/bin/*_embedded \
   && find /usr/local/mysql -type f -name "*.a"
↳-delete \
   && apt-get update && apt-get install -y
↳binutils && rm -rf /var/lib/apt/lists/* \
   && { find /usr/local/mysql -type f -executable
↳-exec strip --strip-all '{}' + || true; } \
   && apt-get purge -y --auto-remove binutils

ENV PATH $PATH:/usr/local/mysql/bin:/usr/local/
↳mysql/scripts

RUN mkdir -p /etc/mysql/conf.d \
   && { \
        echo '[mysqld]'; \
        echo 'skip-host-cache'; \
        echo 'skip-name-resolve'; \
        echo 'user = mysql'; \
        echo 'datadir = /var/lib/mysql'; \
        echo '!includedir /etc/mysql/conf.d/'; \
        } > /etc/mysql/my.cnf

VOLUME /var/lib/mysql

COPY docker-entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]

EXPOSE 3306
CMD ["mysqld"]
```

This example Dockerfile performs the following actions:

- Extends from an existing Debian image called "debian:jessie"
- Uses the RUN instruction to configure the image by adding some groups, making a directory, and installing required software using the Debian apt-get package manager
- Runs gpg to setup some encryption with PGP
- Uses the ENV instruction to define the major and minor versions of MySQL represented in this image
- Runs a long line of commands to install and configure the system followed by another environment variable to set up the system PATH
- Uses the RUN command to create a configuration file
- Uses the VOLUME command to map a file system
- Uses the COPY command to copy and rename the script it will execute when the container starts up, followed by the ENTRYPOINT which specifies the same script to execute
- Uses EXPOSE to declare port 3306 as the standard MySQL port to be exposed
- Uses CMD to specify that the command-line argument passed to the ENTRYPOINT at container startup time is the string "mysqld"

DOCKER MACHINE

Docker Machine is another command-line utility used for managing one or more local or remote machines. Local machines are often run in separate VirtualBox instances. Remote machines may be hosted on cloud providers such as Amazon Web Services (AWS), Digital Ocean, or Microsoft Azure.

CREATE LOCAL MACHINES

When installing the Docker Toolbox, you will be given a default Docker Machine named "default." This is easy to use to get started, but at some point you may need multiple machines to segment the different containers you are running. You can use the `docker-machine create` command to do this:

```
docker-machine create -d virtualbox qa
```

This creates a new local machine using a VirtualBox image named "qa."

LIST MACHINES

If you need to see what machines you have configured you can run the `docker-machine ls` command:

```
docker-machine ls
```

START AND STOP MACHINES

Docker Machines can be started using the `docker-machine start` command.

```
docker-machine start qa
```

Once the machine is started, you have to configure the Docker command line, which Docker Daemon should be interacting with. You can do this using the `docker-machine env` command and evaluating it with `eval`.

```
docker-machine env qa
eval "$(docker-machine env qa)"
```

To stop a machine, use the `docker-machine stop` command.

```
docker-machine stop qa
```

HOT TIP The docker-machine start and stop commands literally start and stop VirtualBox VMs. If you have the VirtualBox Manager open, you can watch the state of the VM change as you run the commands.

ABOUT THE AUTHOR

Christopher M. Judd is the CTO and a partner at Manifest Solutions (http://www.manifestcorp.com), an international speaker, an open source evangelist, the Central Ohio Java Users Group (http://www.cojug.org) and Columbus iPhone Developer User Group leader, and the co-author of Beginning Groovy and Grails (Apress, 2008), Enterprise Java Development on a Budget (Apress, 2003), and Pro Eclipse JST (Apress, 2005), as well as the author of the children's book "Bearable Moments." He has spent 20 years architecting and developing software for Fortune 500 companies in various industries, including insurance, health care, retail, government, manufacturing, service, and transportation. His current focus is on consulting, mentoring, and training with Java, Java EE, Groovy, Grails, Cloud Computing, and mobile platforms like iPhone, Android, Java ME, and mobile web.

CREDITS:
Editor: **G. Ryan Spain** | Designer: **Yassee Mohebbi** | Production: **Chris Smith** | Sponsor Relations: **Chris Brumfield** | Marketing: **Chelsea Bosworth**

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513
888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 **$7.95**