INTRODUCTION

What is Docker?

*an open-source project that automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux.*

What are containers?

The industry standard today is to use Virtual Machines (VMs) to run software applications. VMs run applications inside a guest Operating System, which runs on virtual hardware powered by the server's host OS.

VMs are great at providing full process isolation for applications: there are very few ways a problem in the host operating system can affect the software running in the guest operating system, and vice-versa. But this isolation comes at great cost — the computational overhead spent virtualizing hardware for a guest OS to use is substantial.

Containers take a different approach: by leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a fraction of the computing power.

Why use containers?

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop. This gives

developers the ability to create predictable environments that are isolated from the rest of the applications and can be run anywhere.

From an operations standpoint, apart from portability containers also give more granular control over resources giving your infrastructure improved efficiency which can result in better utilization of your compute resources.

Due to these benefits, containers (& Docker) have seen widespread adoption. Companies like Google, Facebook, Netflix and Salesforce leverage containers to make large engineering teams more productive and to improve utilization of compute resources. In fact, Google credited containers for eliminating the need for an entire data center.

This document contains a series of several sections, each of which explains a particular aspect of Docker. In each section, we will be typing commands (or writing code). All the code used in the tutorial is available in the Github repo.

Prerequisites

There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor. This tutorial uses `git clone` to clone the repository locally. If you don't have Git installed on your system, either install it or remember to manually download the zip files from Github. Prior experience in developing web applications will be helpful but is not required. As we proceed further along the tutorial, we'll make use of a few cloud services. If you're interested in following along, please create an account on each of these websites:

- Amazon Web Services
- Docker Hub

Setting up your computer

Getting all the tooling setup on your computer can be a daunting task, but thankfully as Docker has become stable, getting Docker up and running on your favorite OS has become very easy.

Until a few releases ago, running Docker on OSX and Windows was quite a hassle. Lately however, Docker has invested significantly into improving the on-boarding experience for its users on these OSes, thus running Docker now is a cakewalk. The *getting started* guide on Docker has detailed instructions for setting up Docker on Mac, Linux and Windows.

```
$ docker run hello-world

Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

HELLO WORLD

Playing with Busybox

Now that we have everything setup, it's time to get our hands dirty. In this section, we are going to run a Busybox container on our system and get a taste of the `docker run` command.

To get started, let's run the following in our terminal:

```
$ docker pull busybox
```

Note: Depending on how you've installed docker on your system, you might see a `permission denied` error after running the above command. If you're on a Mac, make sure the Docker engine is running. If you're on Linux, then prefix your `docker` commands with `sudo`. Alternatively, you can create a docker group to get rid of this issue.

The `pull` command fetches the busybox **image** from the **Docker registry** and saves it to our system. You can use the `docker images` command to see a list of all images on your system.

```
$ docker images
```

```
REPOSITORY                    TAG                    IMAGE ID                CREATED
VIRTUAL SIZE
```

```
busybox
```

Docker Run

Great! Let's now run a Docker **container** based on this image. To do that we are going to use the almighty `docker run` command.

```
$ docker run busybox
```

```
$
```

```
$ docker run busybox echo "hello from busybox"
```

```
hello from busybox
```

```
$ docker ps
```

```
CONTAINER ID            IMAGE                     COMMAND                  CREATED
STATUS                  PORTS                     NAMES
```

Since no containers are running, we see a blank line. Let's try a more useful variant: `docker ps -a`

```
$ docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES

305297d7a235        busybox             "uptime"                 11 minutes ago
Exited (0) 11 minutes ago                       distracted_goldstine

ff0a5c3750b9        busybox             "sh"                     12 minutes ago
Exited (0) 12 minutes ago                       elated_ramanujan

14e5bd11d164        hello-world         "/hello"                 2 minutes ago
Exited (0)
```

```
$ docker run -it busybox sh

/ # ls

bin   dev   etc   home  proc  root  sys   tmp   usr   var

/ # uptime

 05:45:21 up  5:58,  0 users,  load average: 0.00, 0.01, 0.04
```

```
docker rm 305297d7a235 ff0a5c3750b9

305297d7a235

ff0a5c3750b9
```

On deletion, you should see the IDs echoed back to you. If you have a bunch of containers to delete in one go, copy-pasting IDs can be tedious. In that case, you can simply run -

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

$ docker container prune

WARNING! This will remove all stopped containers.

Are you sure you want to continue? [y/N] y
```

```
Deleted Containers:

4a7f7eebae0f63178aff7eb0aa39f0627a203ab2df258c1a00b456cf20063

f98f9c2aa1eaf727e4ec9c0283bcaa4762fbdba7f26191f26c97f64090360


Total reclaimed space: 212 B
```

WEBAPPS WITH DOCKER

Great! So we have now looked at `docker run`, played with a Docker container and also got a hang of some terminology. Armed with all this knowledge, we are now ready to get to the real-stuff, i.e. deploying web applications with Docker!

Static Sites

Let's start by taking baby-steps. The first thing we're going to look at is how we can run a dead-simple static website. We're going to pull a Docker image from Docker Hub, run the container and see how easy it is to run a webserver.

```
$ docker run --rm prakhar1989/static-site
```

```
$ docker run -d -P --name static-site prakhar1989/static-site

E61d12292d69556eabe2a44c16cbd54486b2527e2ce4f95438e504afb7b02810
```

In the above command, `-d` will detach our terminal, `-P` will publish all exposed ports to random ports and finally `--name` corresponds to a name

we want to give. Now we can see the ports by running the `docker port [CONTAINER]` command

```
$ docker port static-site

80/tcp -> 0.0.0.0:32769

443/tcp -> 0.0.0.0:32768
```

You can open http://localhost:32769 in your browser.

```
$ docker run -p 8888:80 prakhar1989/static-site

Nginx is running…
```

```
$ docker stop static-site

Static-site
```

```
$ docker images

REPOSITORY                                TAG                 IMAGE ID
CREATED              VIRTUAL SIZE

prakhar1989/catnip                 latest              c7ffb5626a50        2
hours ago           697.9 MB

prakhar1989/static-site            latest              b270625a1631        21
hours ago          133.9 MB

python                             3-onbuild           cf4002b2c383        5
days ago            688.8 MB

martin/docker-cleanup-volumes      latest              b42990daaca2        7
weeks ago           22.14 MB
```

```
ubuntu                                  latest              e9ae3c220b23        7
weeks ago           187.9 MB

busybox                                 latest              c51f86c28340        9
weeks ago           1.109 MB

hello-world
```