

Deployment Pattern



Mobina Noori

Fatemeh Aghamohammadi

Prof : Dr. Hasheminejad



Table Of Contents


1. **How was Docker born?**
2. **Why Docker is popular?**
3. **Why use Docker?**
4. **Docker Architecture**
5. **Docker Terminology**
6. **Learn In Practice**

How was Docker born?

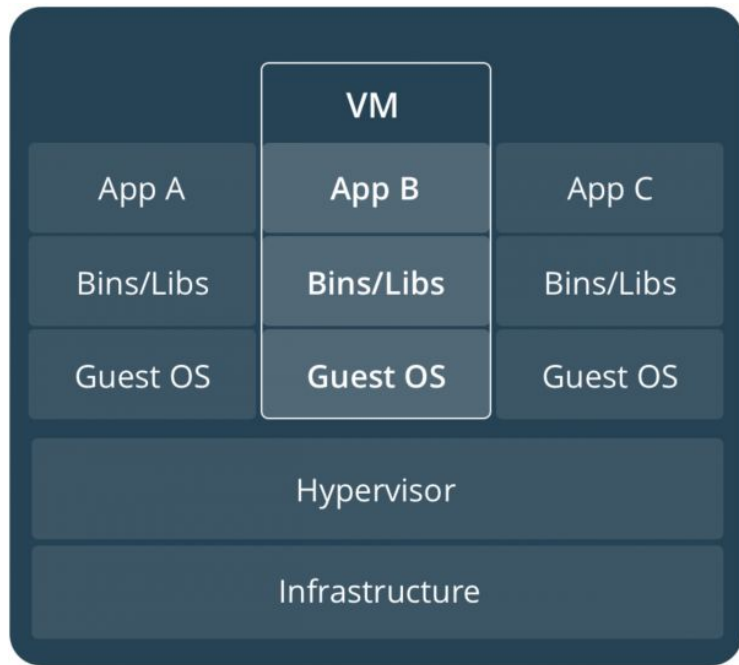



- Docker was originally an in-house project initiated by Docker Inc. (formerly dotCloud).
- It was based on the innovation of dotCloud's multi-year cloud service technology and was open sourced in March 2013 under the Apache 2.0 license.
- Uses Google's Go language for development.
- Is based on the Linux kernel cgroup, namespace, and AUFS like Union FS and other technologies to encapsulate the process of isolation. The isolation environment that Docker creates is called a container.

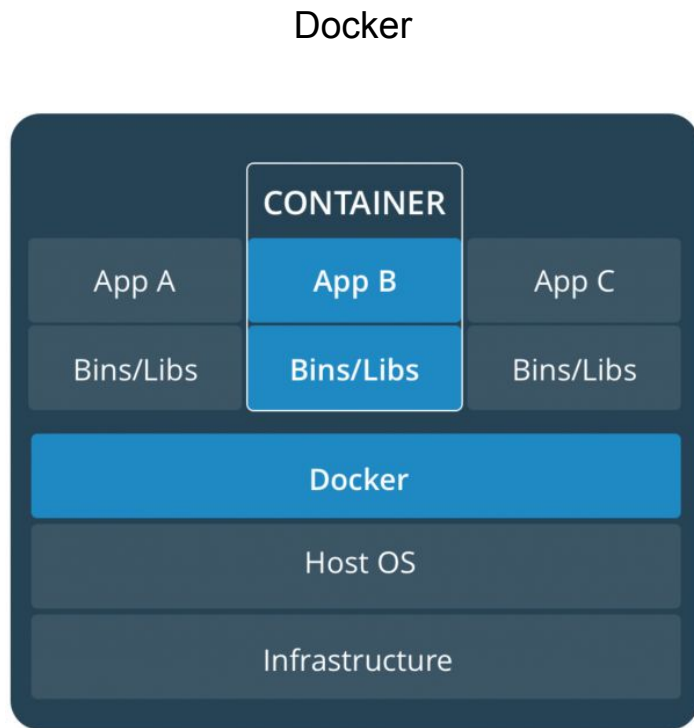
Why Docker is popular?

-  Hypervisors are used to manage Virtual machines
- A full copy of Guest OS is installed on top of the Host machine. These Guest os can be of few Gbs, where the hosted app is just of few Mbs.
- Heavyweight as explained above
- Difficult to monitor and time taking

Virtual Machine



- Docker engine takes care of running the image in an isolated runtime, docker container.

- The application only needs to be packed with its dependencies libraries.
- Much lightweight than Virtual Machines, and powerful at the same time.
- Easy to manage, spinning up a container is just a matter of few commands and a few seconds.

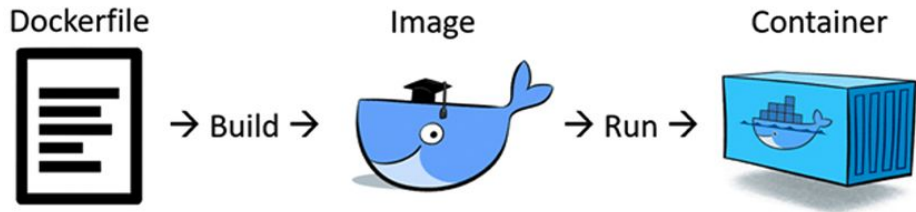


Why use Docker?



Docker is almost an industry standard now, supported in all the popular Operating Systems.

Docker is also available in all the major cloud platform like Google Cloud, AWS, IBM cloud, Alibaba Cloud, etc.



What is Docker?



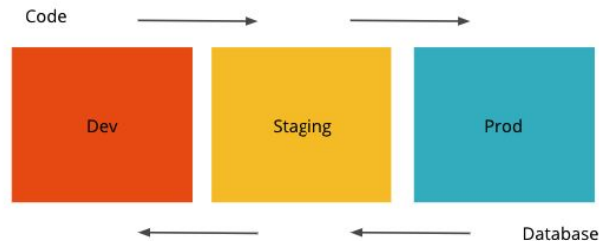
- The easiest way to grasp the idea behind Docker is to compare it to, well... **standard shipping containers**.
- Transportation companies **challenges**:
 1. How to transport **different (incompatible) types** of goods side by side (like food and chemicals, or glass and bricks).
 2. How to handle packages of **various sizes using the same vehicle**.

Develop an application

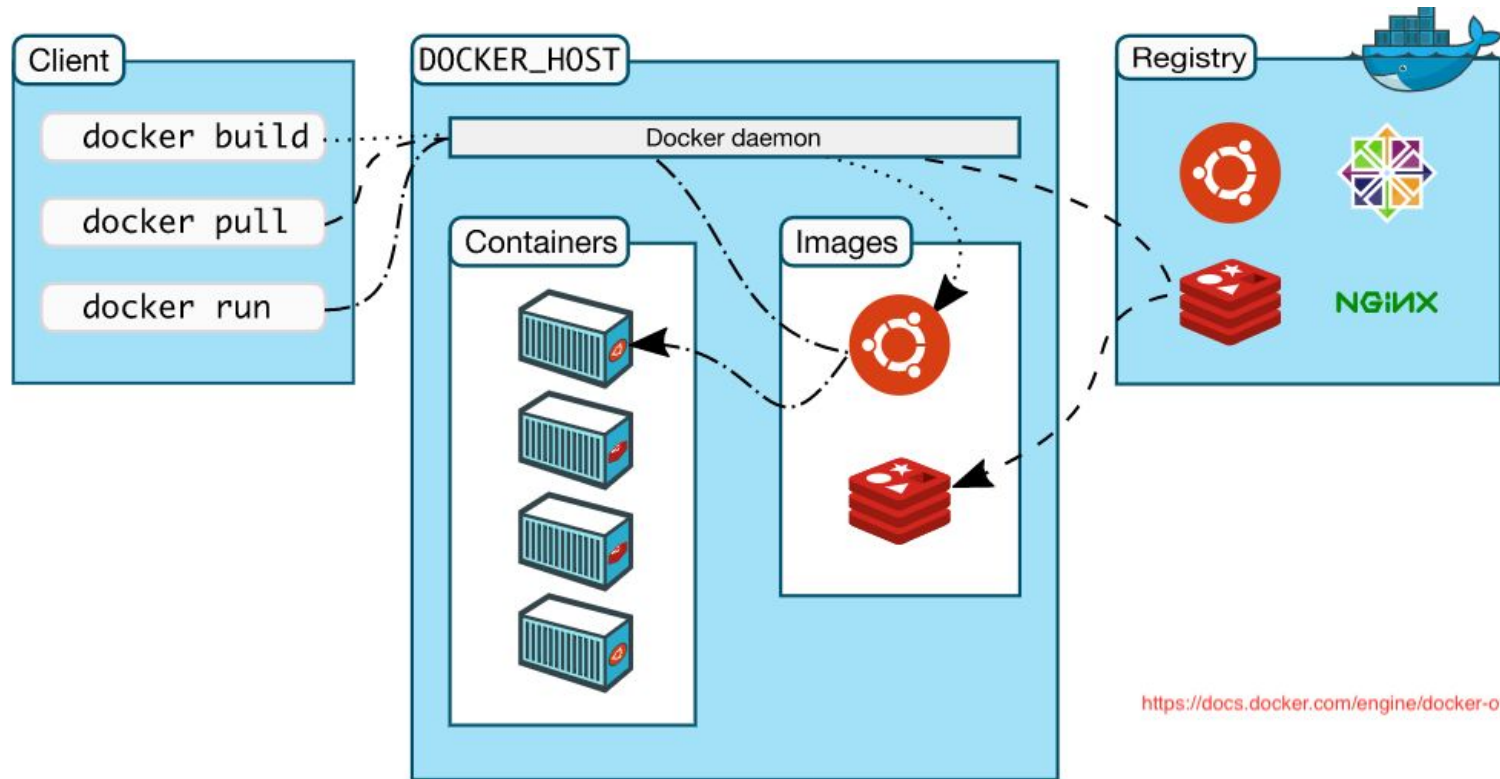
- Provide code along with all possible **dependencies** like libraries, the web server, databases, etc.
- Application is working on your computer, but won't even start on the **staging server**, or the dev or **QA's** machine.
- This challenge can be addressed by **isolating** the app to make it **independent** of the system.

Docker Benefits:

- Faster development process
- Handy application encapsulation
- The same behaviour on local machine / dev / staging / production servers
- Easy and clear monitoring
- Easy to scale



Docker Architecture



<https://docs.docker.com/engine/docker-overview/>

Docker Daemon

- Docker Engine and Docker daemon (dockerd) can be used interchangeably.
- Docker Daemon is a **background process** that manages Docker images, containers, networks, and storage volumes.
- One dockerd can communicate with other daemons to manage docker services.
- Docker daemon exposes REST api using which other daemons and docker cli communicate with it.
- [Choose the one that works for your OS](#)





Docker Client – Docker CLI

```
➔ ~ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1
➔ ~ sudo docker info
[sudo] password for mobina:
Client:
 Context:    default
 Debug Mode: false

Server:
 Containers: 15
  Running: 0
  Paused: 0
  Stopped: 15
 Images: 22
 Server Version: 20.10.14
```


- A user interacts with Docker Engine (dockerd) Using Docker CLI (Docker client).
- The client uses REST API to talk to dockerd. Docker client can communicate with more than one docker daemon.
- Docker client is basically a CLI (terminal or command line) tool as shown opposite.



Docker Registries

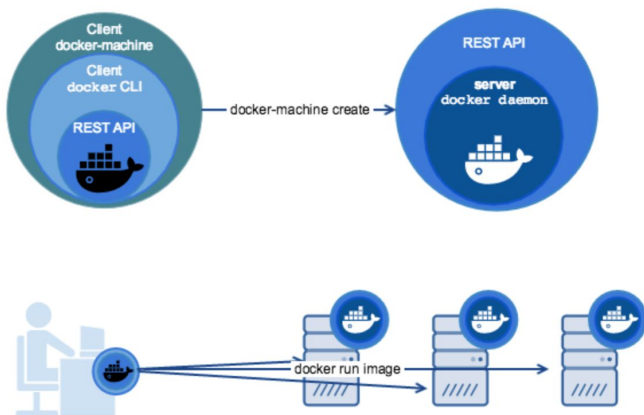
- Docker registry is the central repo where Docker images are stored.
- Docker hub is a public docker repo just like GitHub. When you run `docker pull` or `docker run`, docker images get downloaded from docker registry to your local machine.

Docker Images & Container

- 
- Images are the read-only template with instructions for creating a Docker container (Use `docker images --help` to know the commands to manage images).
 - Docker container is a runnable instance of the Docker image (Use `docker container --help` to know the commands to start, stop, move and delete a container).
 - You can inherit and reuse an image as a base-image to create a new/custom image based on your need. To build your own image, you create a Dockerfile with some instructions, then run `docker build`.
 - By default, a docker container runs in isolation of other containers and host. However, you can configure Dockerfile or pass certain params while running Docker container to open up ports, to make it communicate with other containers, to attach a Volume (storage) and much more.


Docker Compose & Docker Machine

- Docker compose is loved by developers, to define and run multi-container docker applications.
- Usually, use docker-compose.yml to define the containers and use docker-compose --help commands to build, run and manage the containers.



- Docker Machine is a tool for provisioning and managing your Dockerized hosts (hosts with Docker Engine on them).
- Typically, you install Docker Machine on your local system.
- Docker Machine has its own command line client docker-machine --help and the Docker Engine client, docker.
- Use docker-machine commands to start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.

Docker Host & Docker Swarm Mode

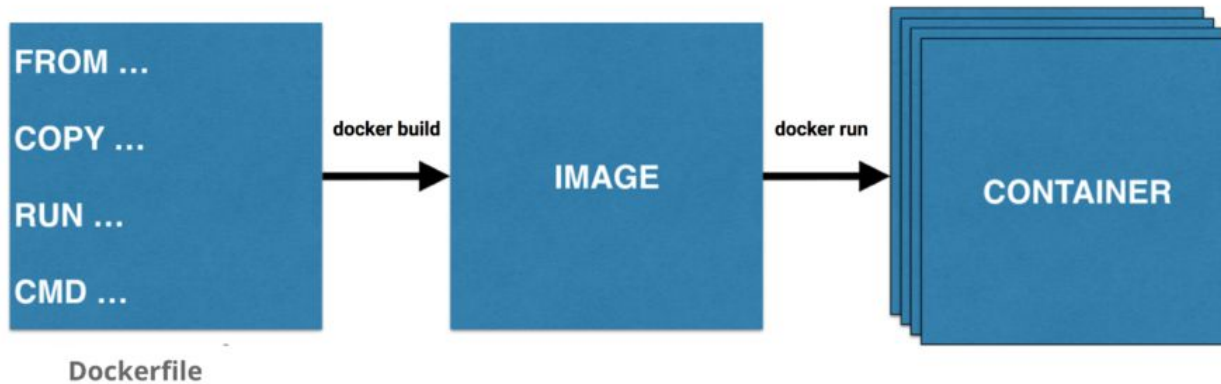
- 
- The Docker host is the machine that has docker installed. It comprises of the Docker daemon, Images, Containers, Networks, and Storage.
 - For a developer, if you using docker on your Laptop, your laptop is the Docker Host. Basically, the system where Docker is installed becomes the host.
 - Docker uses swarm mode to manage clusters of docker-engines (Cluster is also known as Swarm).
 - Use `docker swarm --help` to know the commands to manage a swarm.
 - The popular tool Kubernetes somewhat killed this tool.

Learn in Practice




Assume your project has the following structure:

```
project
├── app.py
└── requirements.txt
```



- app.py is a dummy scripts that only prints out pandas version when executed:



```
requirements.txt x Dockerfile x app.py x
1 import pandas as pd
2
3 print('pandas version:', pd.__version__)
4
5
```

- requirements.txt has all the required dependencies:

```
Plugins supporting requirements.txt files found.
1 |i https://pypi.org/simple
2 numpy==1.18.1
3 pandas==1.0.1
4 python-dateutil==2.8.1
5 pytz==2019.3
6 six==1.14.0
```

Creating a Dockerfile

Create a file and name it Dockerfile (without an extension) and add the following instructions to it:

```
1 FROM python:3.7-slim
2
3 COPY requirements.txt /app/
4 WORKDIR /app
5
6 RUN pip install --upgrade pip \
7     && pip install --requirement requirements.txt
8
9 COPY app.py .
10
11 CMD ["python", "app.py"]
12
```

→ A valid Dockerfile must start with a FROM instruction that sets the parent image.



→ Parent images are hosted on [Docker Hub](#), so you don't need to store them locally.

→ COPY the requirements.txt into the app folder in the parent image (first the folder app will be created as it doesn't exist)

→ Set the app folder as the working directory for any instructions that follow (i.e. we can now simply refer to it as .)

→ RUN 2 pip commands, the first to upgrade pip in the parent image and the second to install all the dependencies found in requirements.txt in the parent image.

→ COPY the app.py to the working directory (i.e. app folder) in the parent image.

→ Define what command gets executed when running the image (i.e. python app.py)



[Supported tags and respective Dockerfile links](#)



[Best practices for writing Dockerfiles](#)

Build Dockerfile as a local Docker image

To build your Docker image, open a terminal and navigating to the project folder, then run the build command:



Docker Build Command

```
→ dockerization sudo docker build --tag start-docker:v.1.0 .
Sending build context to Docker daemon 137.3MB
Step 1/6 : FROM python:3.7-slim
3.7-slim: Pulling from library/python
b85a868b505f: Pull complete
e2be974225ed: Pull complete
a97a867c82b4: Pull complete
ab9320c79a6c: Pull complete
6e8c19ac0f87: Pull complete
Digest: sha256:8c9af1de440d6a036e54d4e7ed3dc8b23e514c002ee9b3f0640f35ad1524160b
Status: Downloaded newer image for python:3.7-slim
--> 17fe3830abb6
Step 2/6 : COPY requirements.txt /app/
--> f87347bf3a0f
Step 3/6 : WORKDIR /app
--> Running in 80d0f3e3b881
Removing intermediate container 80d0f3e3b881
--> fdb0a2b8efa8
Step 4/6 : RUN pip install --upgrade pip && pip install --requirement requirements.txt
--> Running in e711d801e54f
Requirement already satisfied: pip in /usr/local/lib/python3.7/site-packages (2
```

To check that the image was built, you can list all local images and their IDs by running:

```
→ dockerization sudo docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
start-docker	v.1.0	8f8cbe76dc85	2 minutes ago	297MB
python	3.7-slim	17fe3830abb6	2 weeks ago	123MB

```
→ dockerization
```



Run local Docker image

Now you've built a local image, you can run it from the terminal using the run command:


```
→ dockerization sudo docker run start-docker:v.1.0  
pandas version: 1.0.1  
→ dockerization
```



Push local image to a Container Registry 🖱️

- So you've built a local image, and you can run it locally. But how can you share it with others? or even use it yourself from another machine?
- To be able to do that, you need to host it as a repository on a remote Container Registry.
- Docker offers a **Container Registry** at [Docker Hub](#) (similar to GitHub but for hosting Docker images instead).

Now you've built a local image, you can run it from the terminal using the run command:




```
→ dockerization sudo docker tag start-docker:v.1.0 mobina99/start-docker:v.1.0
→ dockerization
```

```
→ dockerization sudo docker image ls
```

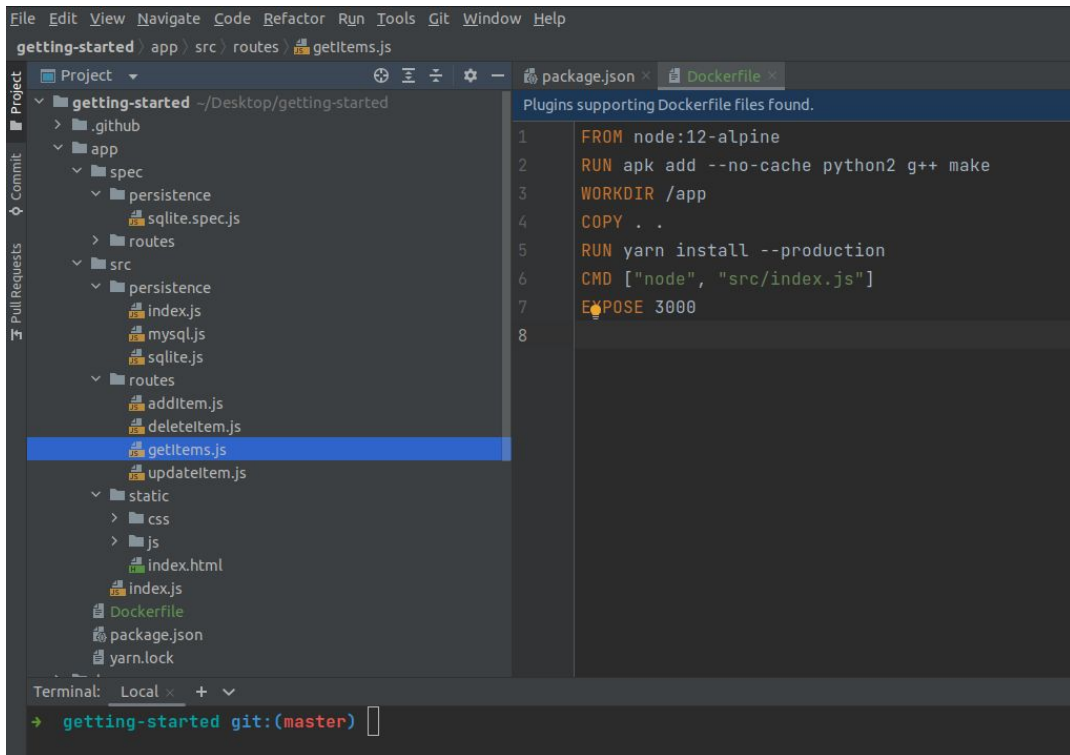
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
start-docker	v.1.0	8f8cbe76dc85	32 minutes ago	297MB
mobina99/start-docker	v.1.0	8f8cbe76dc85	32 minutes ago	297MB
python	3.7-slim	17fe3830abb6	2 weeks ago	123MB

```
→ dockerization
```

```
→ dockerization sudo docker push mobina99/start-docker:v.1.0
The push refers to repository [docker.io/mobina99/start-docker]
8a4a2c0c7557: Preparing
1d5b727116c9: Preparing
b90bbbfdc066: Preparing
05667c31f8f0: Preparing
26bfcfe65477: Preparing
0213c516f6ac: Waiting
d442021c7190: Waiting
08249ce7456a: Waiting
denied: requested access to the resource is denied
→ dockerization
```

Next Sample



The screenshot shows a code editor with a project structure on the left and a Dockerfile on the right. The project structure is as follows:

- getting-started
 - .github
 - app
 - spec
 - persistence
 - sqlite.spec.js
 - routes
 - src
 - persistence
 - index.js
 - mysql.js
 - sqlite.js
 - routes
 - addItem.js
 - deleteItem.js
 - getItem.js
 - updateItem.js
 - static
 - css
 - js
 - index.html
 - index.js
 - Dockerfile
 - package.json
 - yarn.lock

The Dockerfile on the right contains the following code:




```
1 FROM node:12-alpine
2 RUN apk add --no-cache python2 g++ make
3 WORKDIR /app
4 COPY . .
5 RUN yarn install --production
6 CMD ["node", "src/index.js"]
7 EXPOSE 3000
8
```

The terminal at the bottom shows the command prompt: `getting-started git:(master)`.

```
→ app git:(master) X sudo docker build -t getting-started .
Sending build context to Docker daemon 4.654MB
Step 1/7 : FROM node:12-alpine
---> dc1848067319
Step 2/7 : RUN apk add --no-cache python2 g++ make
---> Running in d0992b71f4ee
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/main/x86_64/APKINDEX.gn
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/community/x86_64/APKINDEX.gn
(1/21) Installing binutils (2.33.1-r1)
(2/21) Installing gmp (6.1.2-r1)
(3/21) Installing isl (0.18-r0)
(4/21) Installing libgomp (9.3.0-r0)
(5/21) Installing libatomic (9.3.0-r0)
(6/21) Installing mpfr4 (4.0.2-r1)
(7/21) Installing mpc1 (1.1.0-r1)
(8/21) Installing gcc (9.3.0-r0)
(9/21) Installing musl-dev (1.1.24-r3)
(10/21) Installing libc-dev (0.7.2-r0)
```

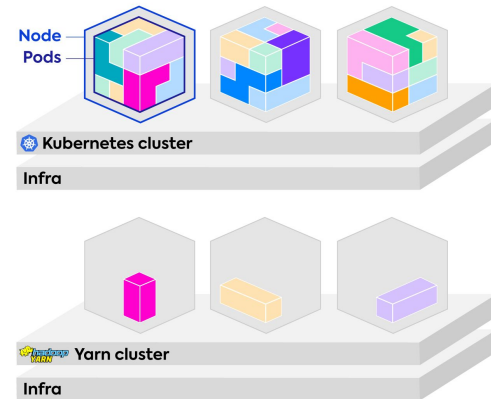
```
→ app git:(master) X sudo docker run -dp 3000:3000 getting-started
5714927a08cfff18f0d9e98641782d1b305558ec2218d98308ec1b5a17a2ef215
→ app git:(master) X
```

Add Item

- ☒ ~~March~~ 
- ☒ ~~April~~ 
- ☐ May 
- ☐ June 
- ☐ July 

Container Schedulers

The development of multiple cluster schedulers by different companies is because there is not an unique solution to solve every problems with cluster computing. Google (main contributor of Omega and Kubernetes) wants an architecture that gives control to the developers, assuming that they respect the rules of the cluster, while Yahoo! (main contributor for YARN) enforces capacity, fairness and deadlines.

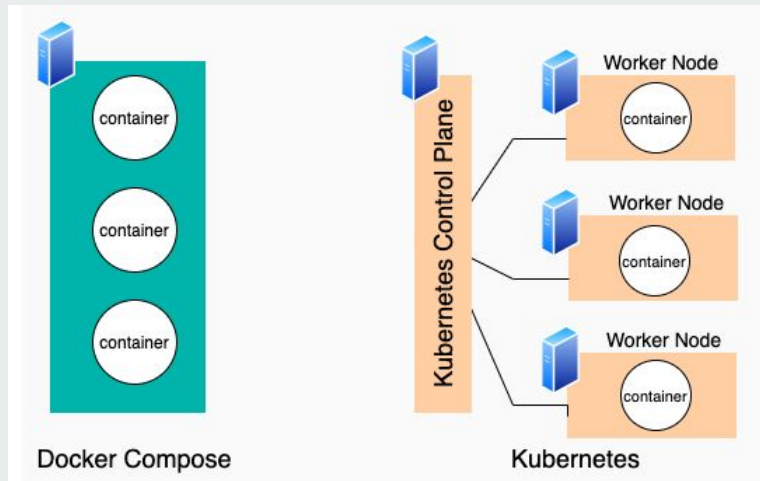


Docker

Kubernetes



Kubernetes vs Docker



- The difference between the two is that Docker is about **packaging containerized applications on a single node** and Kubernetes is meant to **run them across a cluster**.
- Since these packages accomplish different things, they are often used in tandem. Of course, Docker and Kubernetes can be used independently.

References

- <https://djangostars.com/blog/what-is-docker-and-how-to-use-it-with-python/>
- <https://medium.com/@jinnerbichler/10-advanced-tricks-with-docker-2ff191475aae>
- <https://medium.com/pythoneers/docker-zero-to-hero-with-linux-example-e410d89e789c>
- <https://jstobigdata.com/docker/advanced-docker-tutorial/>
- <https://docs.docker.com/engine/reference/commandline/cli/>
- <https://medium.com/swlh/python-how-starting-with-docker-d2be73d9ae92>

Thanks for your attention.
