



An interactive Intro to Docker



What is Docker?



Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere.





Nerdy Explanation

Docker is a containerization platform built on top of LXC.

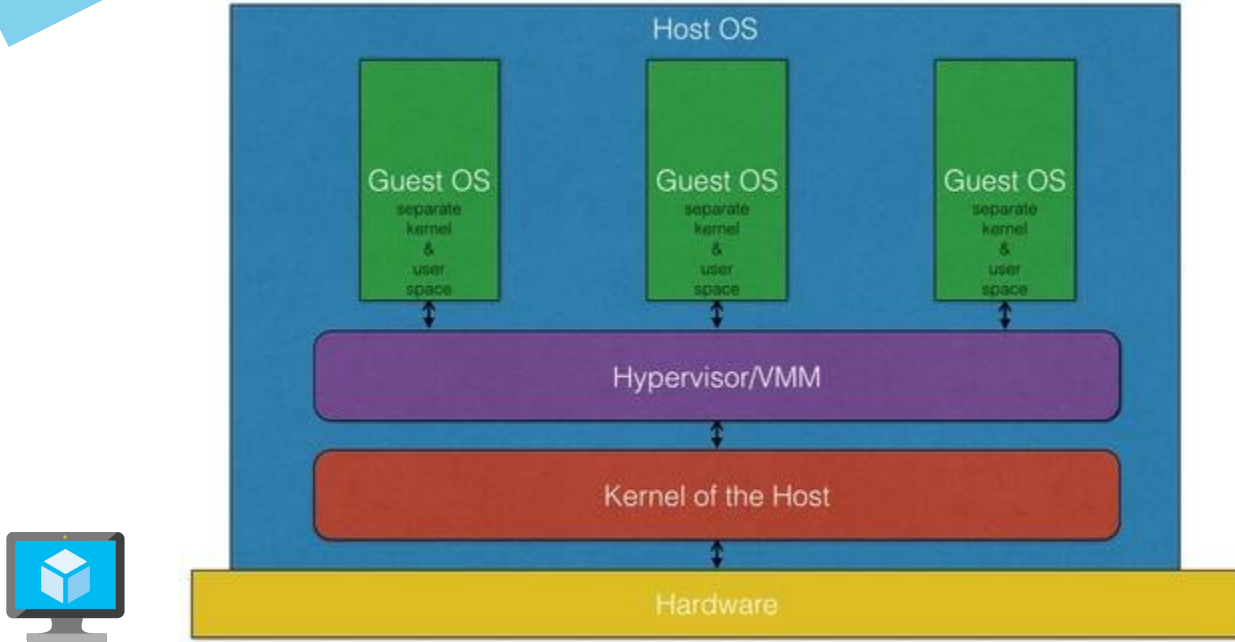
Containers are like VMs in how you would use them. The difference is, the VM hypervisor abstracts an entire machine.

Containers just abstract the OS kernel.

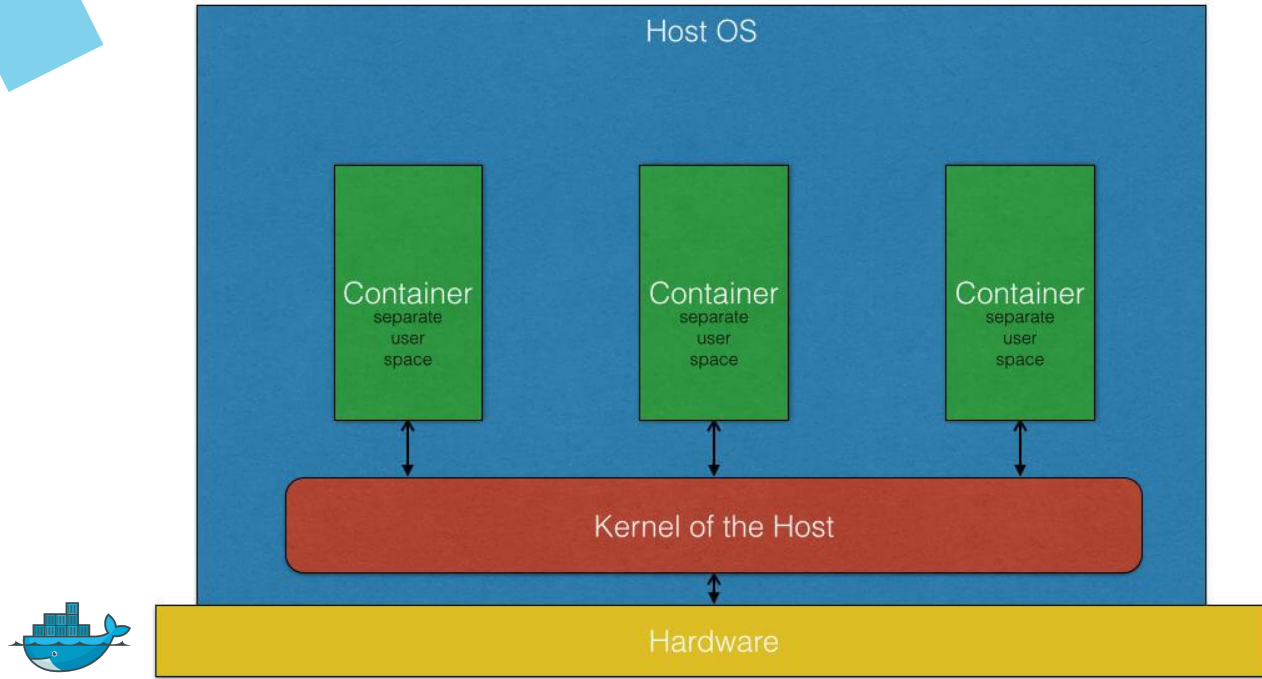




Like Virtual Machines?



Hypervisor based Virtualization

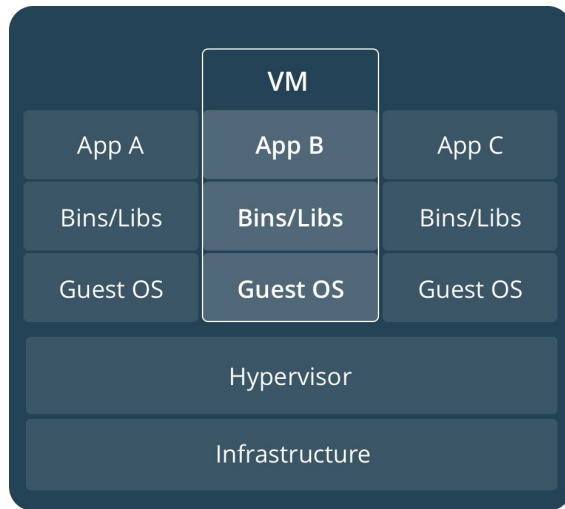


Container Virtualization

Isolation



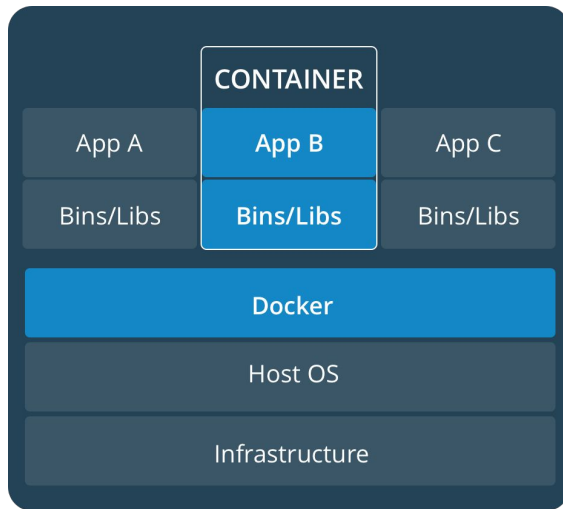
Processes running within a VM cannot see or affect any processes running in another VM, or in the host system.



Isolation



Processes running within a container cannot see or affect any processes running in another container, or in the host system.



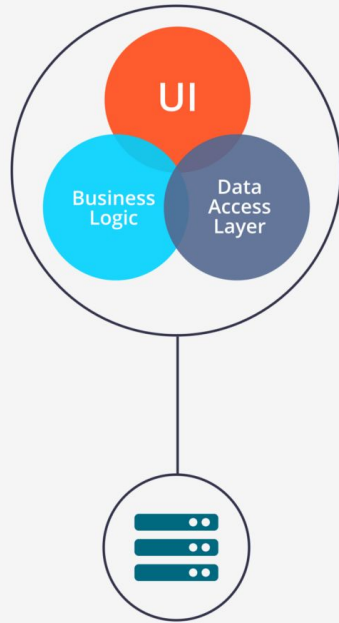


**When and Why would I
ever use Docker?**

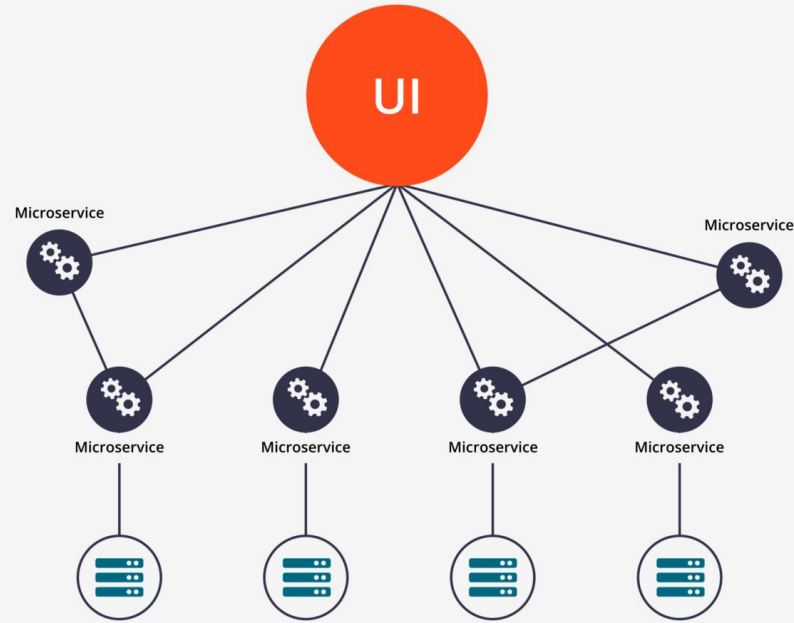


Web services are moving away from monolithic apps on giant servers to loosely coupled microservice architecture

Web services are moving away from monolithic apps on giant servers to loosely coupled microservice architecture



Monolithic Architecture



Microservice Architecture



Why Docker and MSA?


- » It's easier to maintain and update smaller applications and codebases
- » Faster release cycles
- » Microservices are deployed across many small servers
- » Apps should be:
 - ◇ Independently scalable
 - ◇ Stateless
 - ◇ Highly Available

<https://12factor.net/>





CI/CD


- » Deploying via Docker makes it much easier to automatically deploy or rollback a version of your app.
 - » Once a version is tested in QA environment and approved, you can push that exact version to production with confidence that it is consistent.
- 



Portability

It shouldn't matter what the host platform is, if it's running Docker, it will run predictably.

If a stack runs on your machine, then it should run the same way on any other dev environment, or production server.






Other real world use cases?



Basic Deployment Concepts



Basic Deployment Concepts

1. Operating System
 - a. Docker Hosts can be Linux, Win, OSX, or K8s
 - b. Docker Containers can run Linux and Win bins
 2. Application to Deploy
 - a. Web Service, Daemon, Worker
 3. Side Effects
 - a. Network Access
 - b. Jobs
 - c. Artifacts
- 

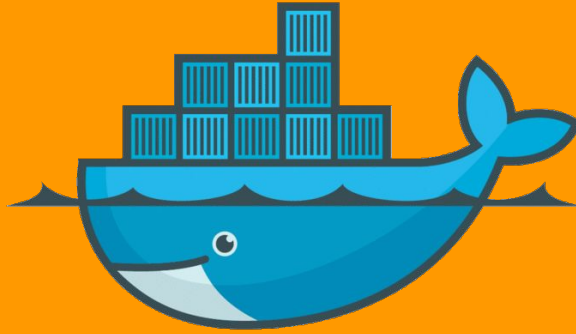


Basic Docker Concepts



Basic Docker Concepts


1. CLI
 2. Docker Daemon
 3. Images
 4. Container Image Registries
 5. Containers
 6. Ports
 7. Volumes
 8. ENVs
 9. Dockerfiles
- 



Additional Docker Concepts



Additional Docker Concepts


- » We will not be covering the following...
 1. Networking
 2. Docker Compose
 3. Docker Swarm
 - » Dive deeper into these concepts on your own
 - » Hack with us at SeattleJSHackers for support
- 



Let's dive in...




Workshop approach

- » You will type in lots of commands
 - » Ask for help quickly
 - » Intentionally encounter problems
 - » Learn to debug to gain understanding
 - » Experiment!
- 

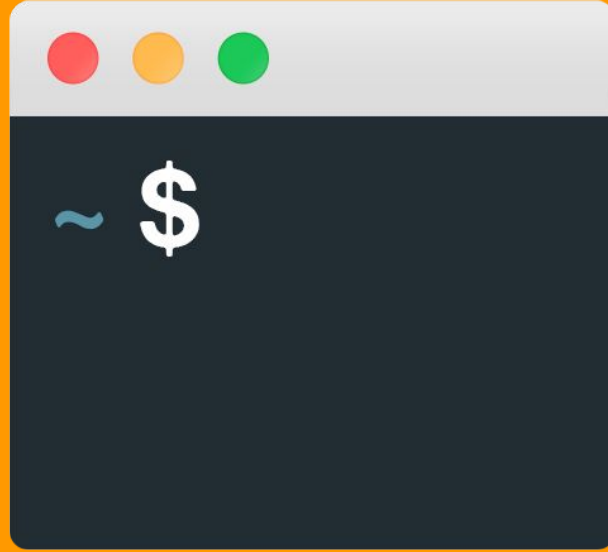


Workshop requirements

- » A Laptop
 - » Docker needs to be installed
 - ◇ [install it now](#)
 - » A Code Editor
 - » Patience
- 

1. CLI

Command Line Interface





1. CLI

- » A Terminal Emulator is an application that runs a shell - **Launch one now**
- » A Shell is an interactive REPL that allows you to interact with your OS
- » We'll use the CLI to run Docker commands.
- » Ensure Docker is installed

```
$ docker
```





2. Docker Daemon

This needs to be running to do anything with Docker

2. Docker Daemon

- » Check if Docker Daemon is running

```
$ docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|---------|---------|--------|-------|-------|
|--------------|-------|---------|---------|--------|-------|-------|

- » If the Docker Daemon is not running, you'll get an error like this

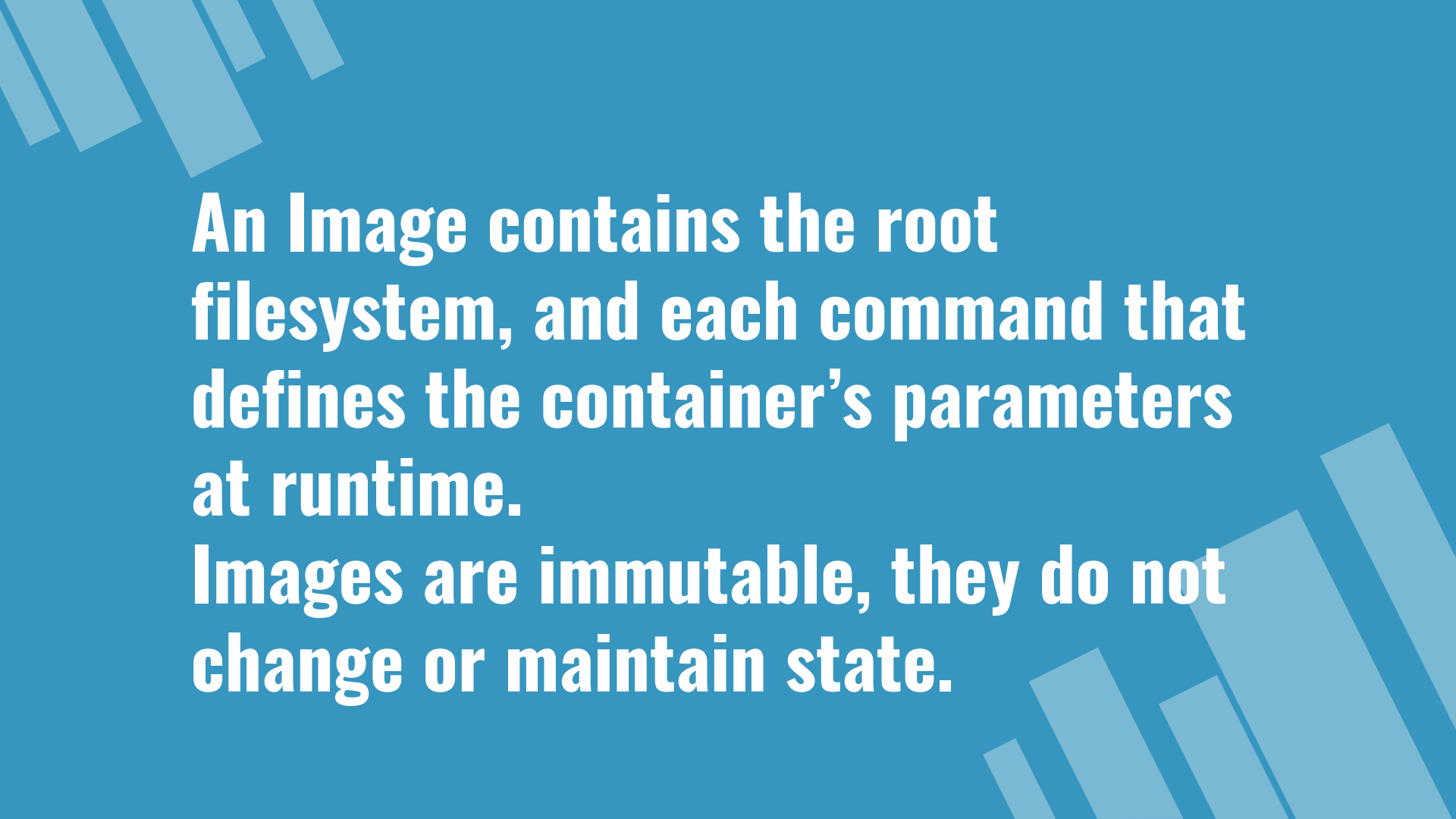
```
error during connect: Get  
http://%2Fvar%2Frun%2Fdocker.sock/v1.37/containers/json: EOF
```

- » Then turn it on like **this**



3. Docker Images

The base of any running container



An Image contains the root filesystem, and each command that defines the container's parameters at runtime.

Images are immutable, they do not change or maintain state.



3. Docker Images

- » Pull a few images


```
$ docker pull busybox
```

- » *Note the tag and full image name that was pulled*

- » Inspect your images

```
$ docker images
```

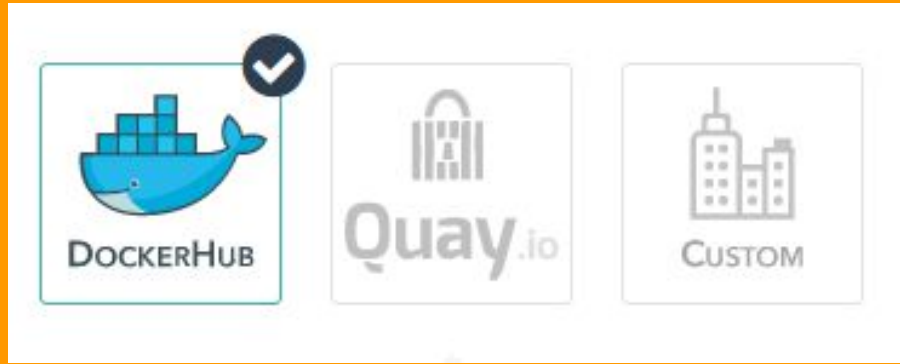
```
$ docker pull node:alpine
```





Where did we “pull” these images from?





4. Container Image Registries

A service that stores container images



**Registries can be hosted by self or
3rd party.**

**Images can be public or private.
You can push and pull images.**





Docker Hub - hub.docker.com

Quay.io

AWS ECR (EC2 Container Registry)

Google Container Registry

GitLab





5. Containers

A runtime instance of an Image



5.a Running Containers

- » Let's run an instance of Busybox


```
$ docker run busybox
```

- » *What happened?*
- » Inspect your running containers

```
$ docker ps
```

- » Inspect all containers on your host

```
$ docker ps -a
```



5.b Running a Container Interactively

- » Let's run a shell in a **Busybox** container

```
$ docker run -it busybox
```

- » What does the *-it* flag do?
- » You are now running a shell inside a container!

```
$ ls
```

- » Note: *There is no **docker** command in **busybox***



**Always be aware of which shell you
are using!**

Host or Container?



5.c Container Instances

- » Let's run multiple instances

```
$ touch HELLO_DOCKER  
$ ls  
$ exit
```

```
$ docker run -it busybox
```

```
$ ls
```

- » Where is the HELLO_DOCKER file?

```
$ exit
```

5.d Understand “CMD”

- » What is happening when we run `busybox`?
 - ◇ Omitting the `COMMAND` arg will run the Image's default `CMD` parameter
- » Let's run a different command in `busybox`

```
$ docker run -it busybox ls -lh
```

- » Let's run a different shell, `/bin/bash`

```
$ docker run -it busybox bash
```

- » What happened?




5.e Managing Containers

- » Let's look at our containers

```
$ docker ps  
$ docker ps -a
```

- » Clean up unused containers

```
$ docker rm [Container Name Here]
```



5.f Named Containers

- » Docker auto assigns container names
- » Let's name our containers

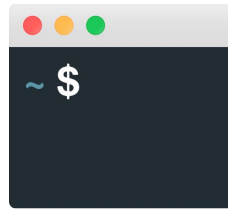
```
$ docker run -it --name bb busybox
```

- » Open another terminal

```
$ docker ps
```

- » Kill that running container

```
$ docker rm -f bb
```





5.g Temporary Containers

- » Let's run a temporary instance of `busybox`


```
$ docker run -it --rm busybox
```

- » *What does the `--rm` flag do?*

```
$ exit
```

- » Inspect your containers

```
$ docker ps  
$ docker ps -a
```





6. Exposing Ports

Enable network access to your container

6.a Expose a container port

- » Let's run a container that listens on port :80 exposed as port :8080

```
$ docker run --name web -p 8080:80 nginx
```

- » Visit <http://localhost:8080>
- » This is running “interactively” consuming one of your terminals

6.b Run a webserver daemonized

- » Kill that running container by pressing `ctrl+c`
- » Run a webserver daemonized

```
$ docker run --name web -d -p 8080:80 nginx
```

- » Resolve errors!

```
$ docker ps  
$ # ??
```

```
$ docker run --name web -d -p 8080:80 nginx
```

- » Visit <http://localhost:8080>

6.c Enter a running container

- » You can run any legal command inside of any running container using `docker exec`

```
$ docker exec -it web bash
```

```
root@w00t$ cat /usr/share/nginx/html/index.html
```

- » You have access to the OS. `Install a code editor.`

```
root@w00t$ apt update
root@w00t$ apt install nano
root@w00t$ nano /usr/share/nginx/html/index.html
root@w00t$ exit
```

6.d Kill a daemonized container

- » How would you kill a daemonized container?

```
$ docker rm web
```

- » What is the error?
- » You could either stop the container, then kill it

```
$ docker stop web  
$ docker rm web
```

- » Or, you could force kill the container

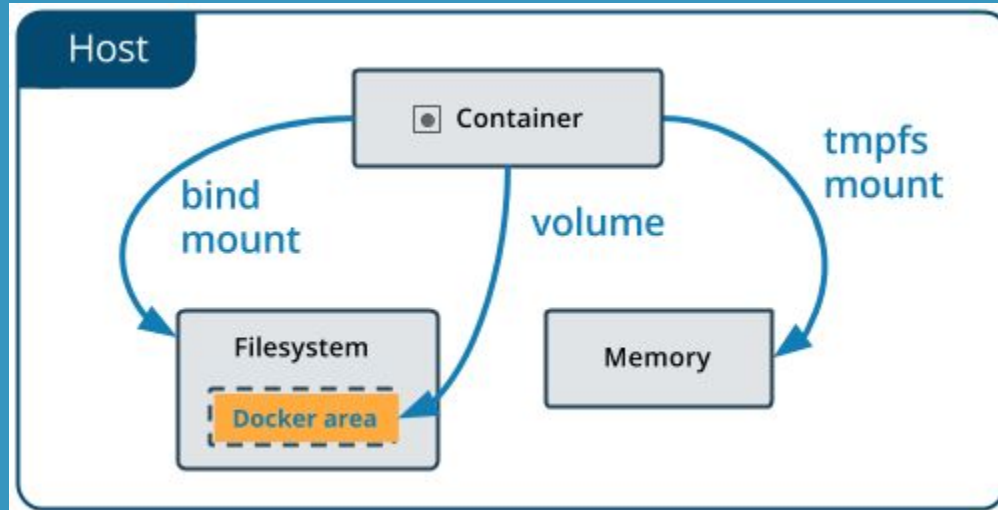
```
$ docker rm -f web
```



`-v /tmp:/tmp:ro`

7. Container Volumes

Persistent and Mounted Volumes



7.a Mount a local directory

- » Create a practice static web site

```
$ mkdir -p ~/Learn-Docker/static-website  
$ cd ~/Learn-Docker/static-website  
$ $EDITOR index.html
```



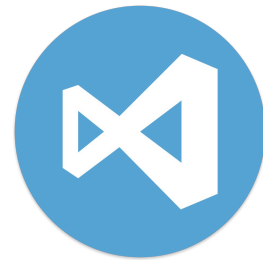
7.b index.html

```
<h1>Hello Docker</h1>
```

7.c Mount a local directory

- » Create a practice static web site

```
$ mkdir -p ~/Learn-Docker/static-website  
$ cd ~/Learn-Docker/static-website  
$ $EDITOR index.html
```



- » Run **nginx** daemonized, mounting this directory

```
$ docker run --name web -d -p 8080:80 \  
-v $PWD:/usr/share/nginx/html/ nginx
```

- » Visit <http://localhost:8080>
- » Edit your website with your favorite editor

7.d Run a node script

- » Create a practice node script

```
$ mkdir -p ~/Learn-Docker/node-script  
$ cd ~/Learn-Docker/node-script  
$ $EDITOR script.js
```



7.e script.js

```
console.log('Hello Docker');
```

7.f Run a node script

- » Create a practice node script

```
$ mkdir -p ~/Learn-Docker/node-script  
$ cd ~/Learn-Docker/node-script  
$ $EDITOR script.js
```



- » Run `node:alpine`, mounting this directory

```
$ docker run --name node-script \  
-v $PWD:/srv node:alpine node /srv/script.js
```

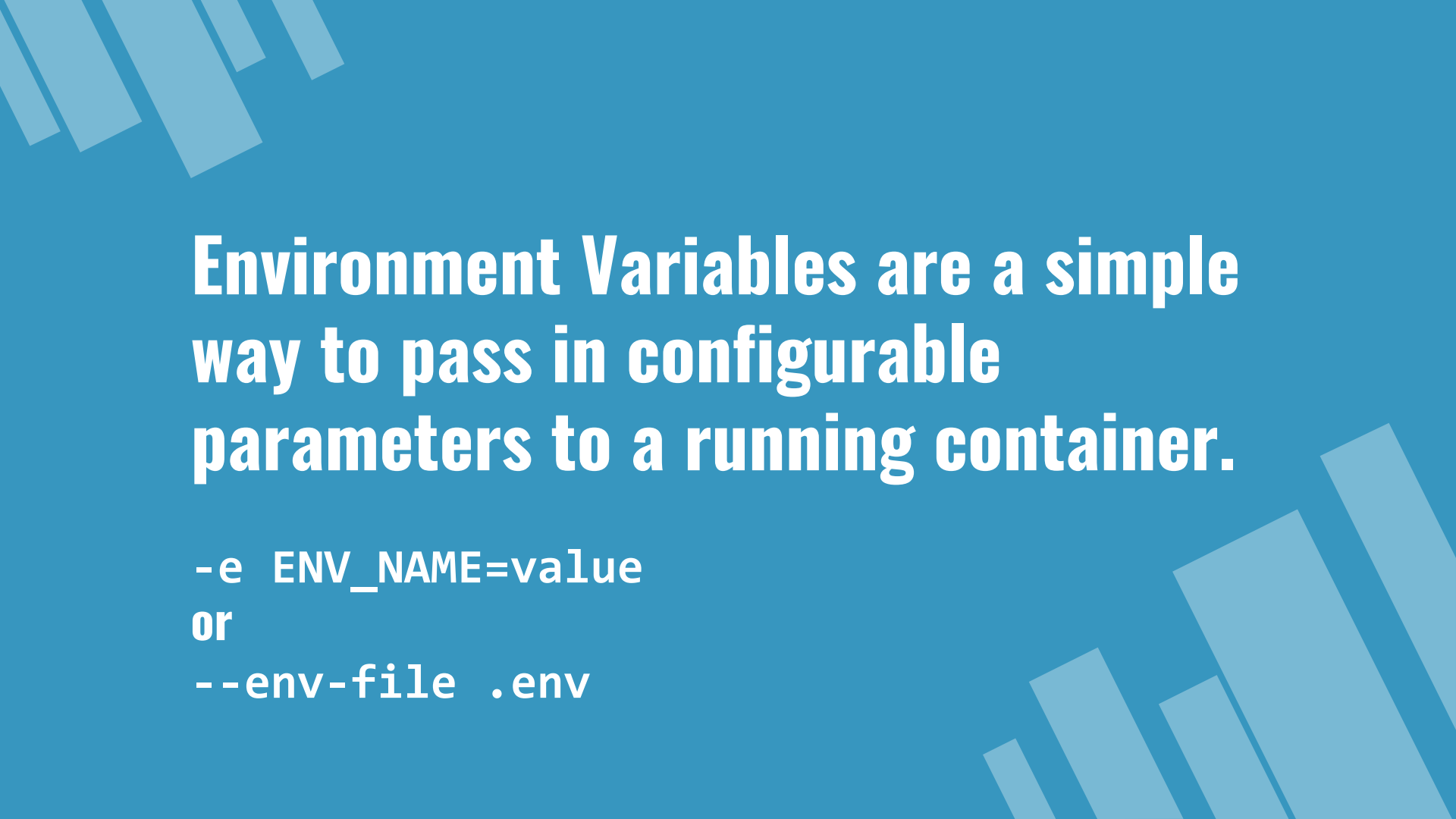
- » Run the same command again
- » What's happening, and how would you fix it?



`-e PORT=3000`

8. ENV Variables

Runtime configuration



Environment Variables are a simple way to pass in configurable parameters to a running container.

`-e ENV_NAME=value`
or
`--env-file .env`

8.a Run a node server

- » Create a practice node web server

```
$ mkdir -p ~/Learn-Docker/node-api  
$ cd ~/Learn-Docker/node-api  
$ npm init  
$ npm i -S fastify  
$ $EDITOR index.js
```

- » Source : <https://github.com/theRemix/node-api>



8.b index.js

```
const fastify = require('fastify')({ logger: true })
const API_NAME = process.env.API_NAME || 'default'

fastify.get('/', async (request, reply) => {
  reply.type('application/json').code(200)
  return { API_NAME }
})

const start = async () => {
  try {
    await fastify.listen(3000, '0.0.0.0')
    fastify.log.info(`server listening on ${fastify.server.address().port}`)
  } catch (err) {
    fastify.log.error(err)
    process.exit(1)
  }
}
start()
```

» Source : <https://github.com/theRemix/node-api>

8.c Run a node server

- » Pass the `API_NAME` env to the container

```
$ docker run -d --name node-api -p 3000:3000 \
-v $PWD:/srv -e API_NAME=yourname node:alpine node /srv
```

- » Visit <http://localhost:8080>
- » Inspect the logs with `docker logs`

```
$ docker logs node-api
$ docker logs -f node-api
```

- » Inspect the container with `docker inspect`

```
$ docker inspect node-api
```

8.d Run multiple node servers

- » Let's run 1 more

```
$ docker run -d --name node-api-2 -p 3000:3000 \  
-v $PWD:/srv -e API_NAME=yourname-2 node:alpine node /srv
```

- » Resolve the error
- » Run 2 more instances of the api

```
$ docker run -d --name node-api-2 -p 3002:3000 \  
-v $PWD:/srv -e API_NAME=yourname-2 node:alpine node /srv  
$ docker run -d --name node-api-3 -p 3003:3000 \  
-v $PWD:/srv -e API_NAME=yourname-3 node:alpine node /srv
```

- » Visit each api <http://localhost:3002>



Dockerfile

9. Dockerfiles

A text file containing all the commands needed to build a Docker image.



We'll package up your node app for deployment.

9.a Create a Dockerfile

```
$ cd ~/Learn-Docker/node-api  
$ $EDITOR Dockerfile
```

- » Dockerfiles have a simple format to follow
- » It *could* require a significant understanding of linux
- » RTFM



9.b Dockerfile

```
FROM node:alpine
EXPOSE 3000
ENV API_NAME default
WORKDIR /srv

COPY package*.json /srv/
COPY index.js /srv/

RUN npm install

CMD ["node", "."]
```

9.c Build an Image from Dockerfile

- » Build with the `docker build` command
- » Use your own namespace

```
$ docker build -t theremix/node-api:0.0.1 .
```

- » Run a container from your image...
 - ◇ Expose port `:3005` to `:3000`
 - ◇ No volumes needed
 - ◇ No command needed

```
$ docker run -d --name node-api-5 -p 3005:3000 \  
-e API_NAME=yourname-5 theremix/node-api:0.0.1
```

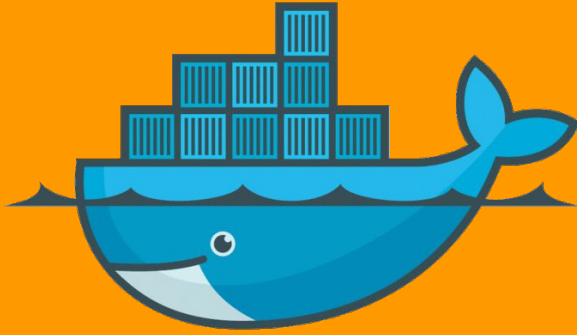


**Images can be pushed to a registry
for sharing and deployment.**



Conclusion.

You should now have some basic fundamentals in your toolbox to deploy amazing applications!



Q&A