Before we start, make sure to install:

- Basic knowledge in Linux and networking
- Laptop
- Docker (version 17+, Docker Compose 1.16+)
- Terminal or Text Editor.
- Git
- Gnu make
- Go 1.8

WIFI: Wizeline Academy
Password: academyGDL

Clone our workshop repo:

$ git clone

https://github.com/wizelineacademy/docker-intro-app.git

DevOpstober Fest
Docker Workshop

WIZELINE®

Objective of this Workshop

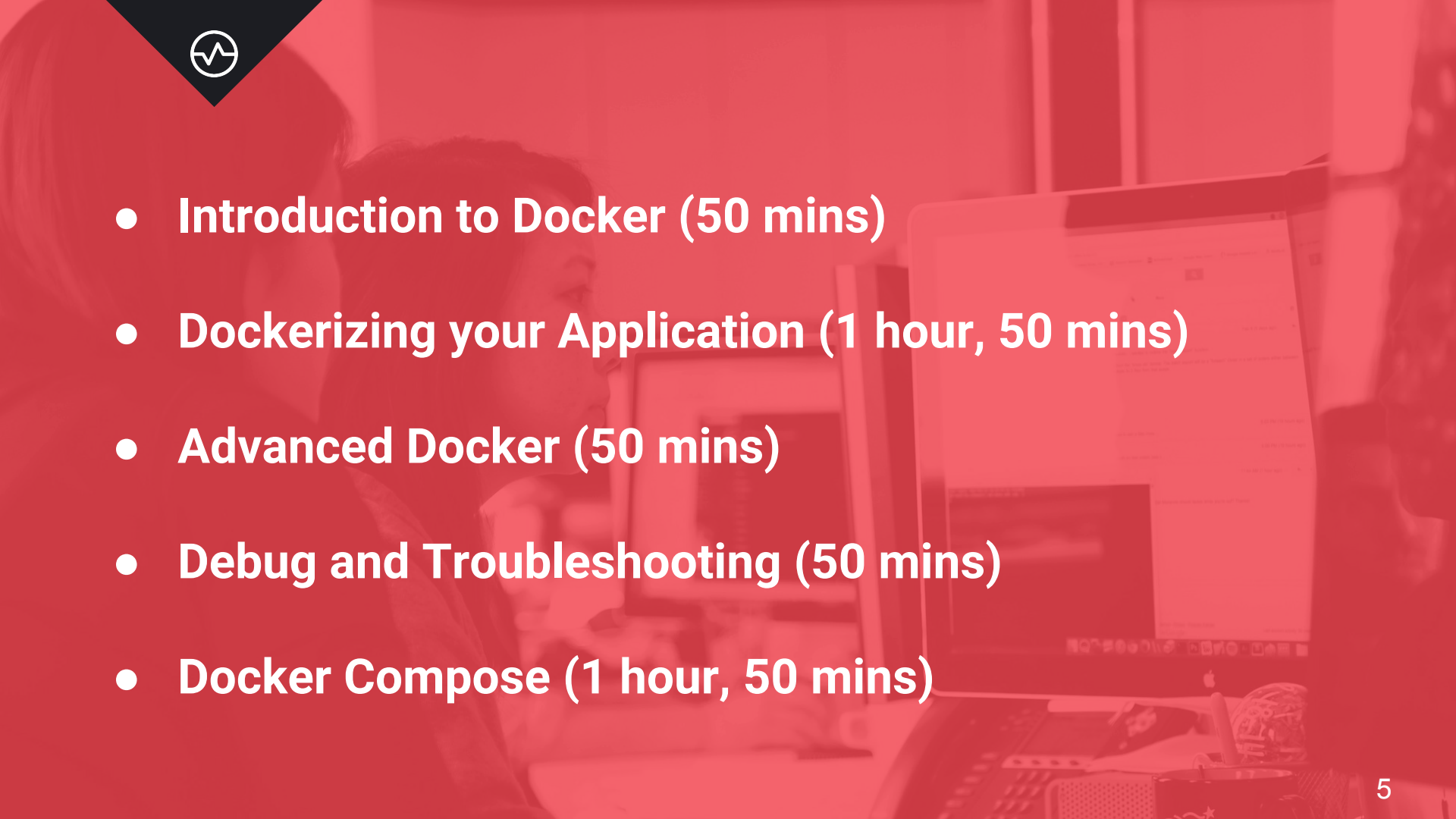
Today's we will have an introduction to the containers technology, using a very popular container platform called Docker.

Containers is a technology that has provided a way to manage our resources, applications and services, it helps to minimize the effort needed to set up environments for development and operations areas, provides a fast / easy way to manage our deployment.

Why Docker?

Docker is a highly used container solution, it has a lot of tools built around it to help us to manage our apps. It's an open source solution (moby) and it has the backup of a big community that is constantly providing documentation, updates and enhancements for the benefit of all.

TODAY' S AGENDA



- Introduction to Docker (50 mins)
- Dockerizing your Application (1 hour, 50 mins)
- Advanced Docker (50 mins)
- Debug and Troubleshooting (50 mins)
- Docker Compose (1 hour, 50 mins)

Pre-Steps

Make sure you have installed in your system:

- Git
- Docker 17.05+
- Go 1.8+

Clone our workshop repo:
$ git clone https://github.com/wizelineacademy/docker-intro-app.git

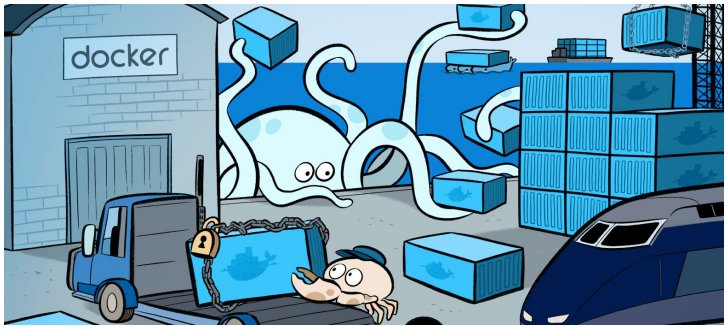
Introduction to Docker

Introduction to Docker

- What's a Container / Image
- Installing Docker
- Running a Docker Image

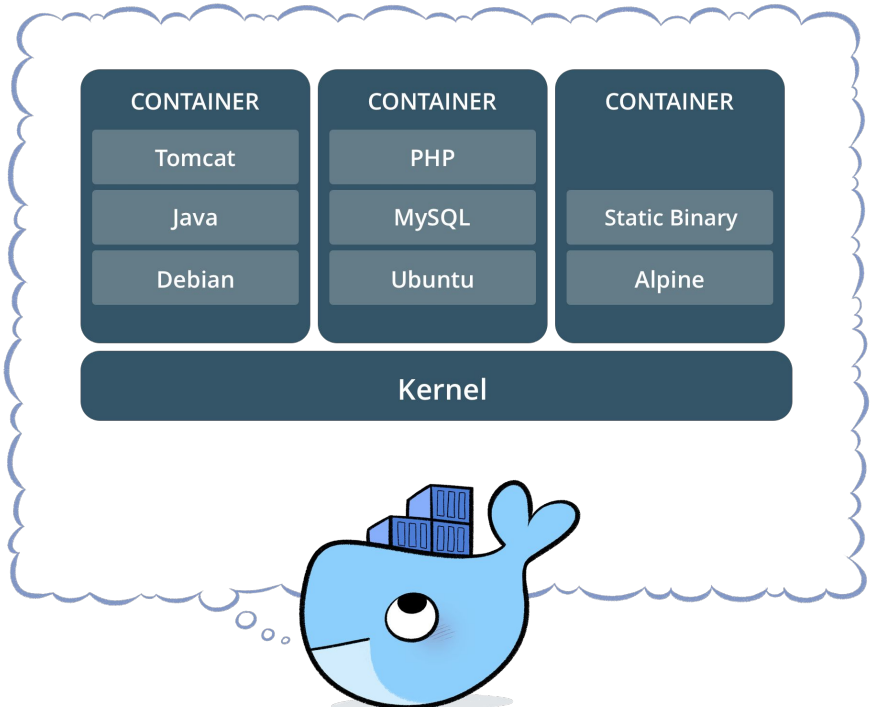
What is Docker?

- Provides containers.
- Provides resource isolation.
- Avoids the overhead of starting and maintaining VMs.
- Eliminates “works on my machine” problems.
- Runs and manages apps side-by-side in isolated containers.

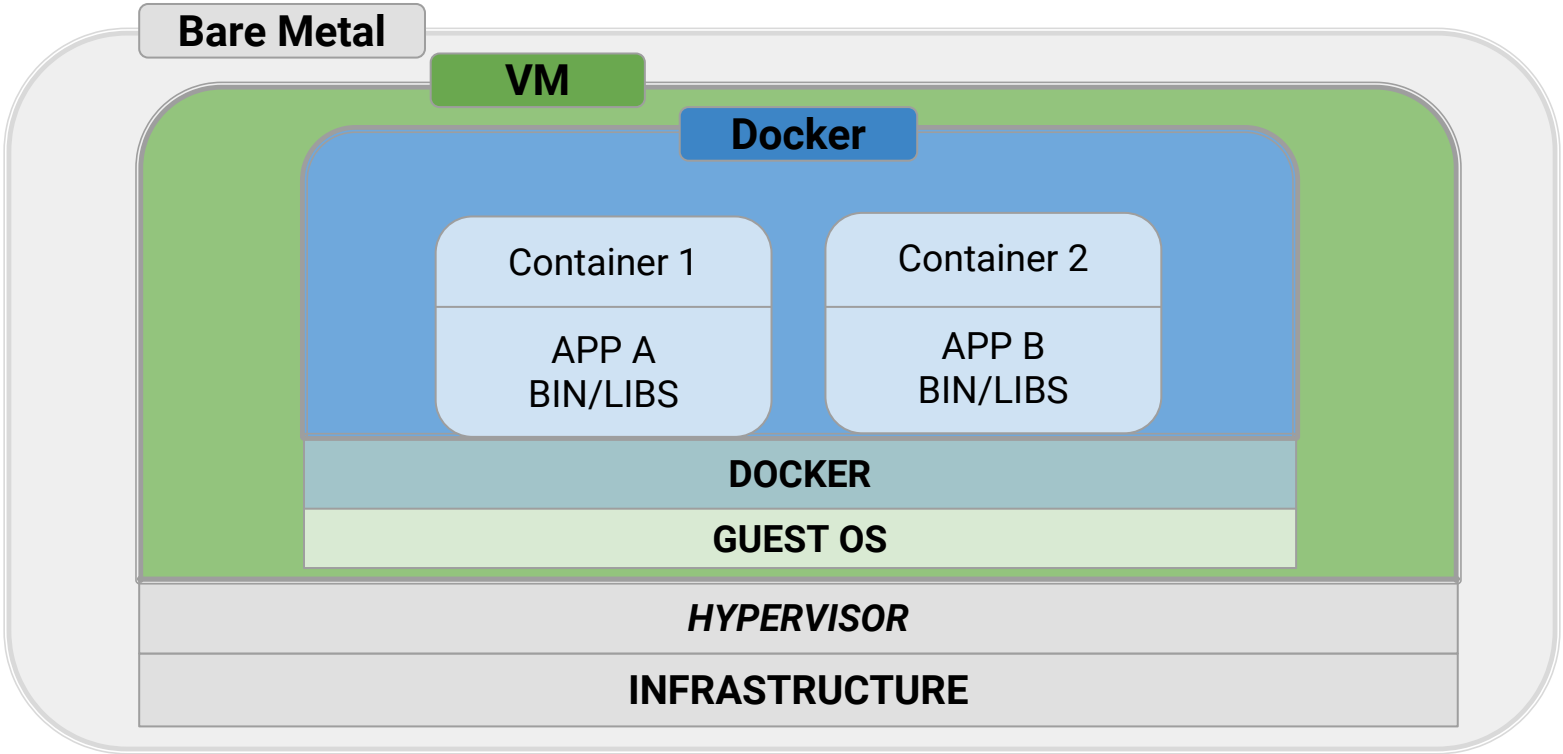


What's a Container / Image

- Packaged software in a standard format
- Can run on any computer with Docker
- Isolate resources
- Launch faster than VMs
- Ideally images are light weight



Bare Metal / VM / Docker



Installing Docker

Linux:

```
# dnf install docker {fedora}
# yum install docker-latest {centOS}
# apt-get install docker.io {ubuntu}
# pacman -S docker {Arch}
```

Mac:

```
https://docs.docker.com/docker-for-mac/install/#download-docker-for-mac
```

Windows:

```
https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows
```

Starting Docker

Linux:

```
# systemctl start docker {systemd}
# service docker start
# systemctl enable docker {systemd}
# chkconfig docker on
# usermod -aG docker your_user
```

Mac:

```
Start docker application
```

Running a Docker Image

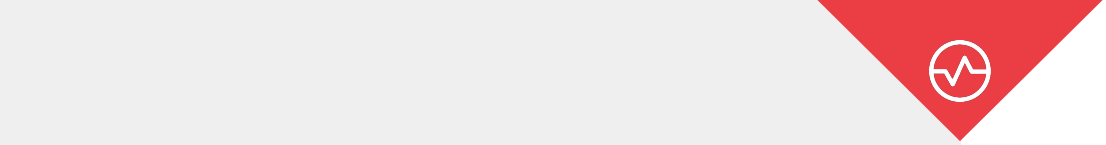
Let’s do some hands on work to understand better how to run a docker image, open your terminals.

```
$ docker rmi nginx
```

Common options

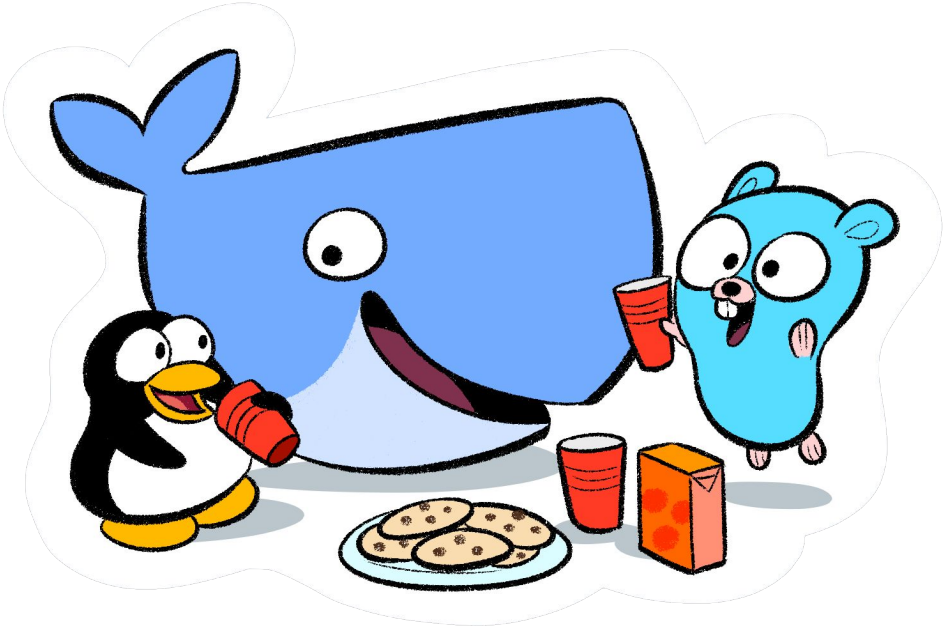
- `pull`
- `images`
- `run`
- `exec`
- `logs`
- `ps`
- `stop`
- `rm`
- `rmi`

Dockerizing your Application



Overview

- Dockerfile
- Building the app
- Running the app
- Docker hub



Dockerfile

What is a Dockerfile?

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

Using docker build users can create an automated build that executes several command-line instructions in succession.

Dockerfile:

```
FROM busybox
ENTRYPOINT ["/bin/echo"]
CMD ["Hello World"]
```

Execute:

```
# Build the image
$ docker build -t example .

# Run our example
$ docker run --rm example

# Run our example with another cmd
$ docker run --rm example Hi
```

Build heavy image for later

```
$ cd docker-intro-app/Backend/
$ git checkout dockerize-your-app
```

big-backend:

```
# Move into backend dir
$ cd docker-intro-app/Backend/
# Build the image, and tag it as big-backend
$ docker build -t big-backend .
```

Most used commands

```
ARG OS_VERSION=stretch                  # var passed at build-time
FROM node:${OS_VERSION}                 # sets the base image
LABEL version="1.0.2"                   # adds metadata to an image
ENV APP_DIR=/opt/my_app                 # sets env var in build-time and
                                        # in run-time
WORKDIR ${APP_DIR}                      # sets (and creates) work dir
ADD my_app.tar.xz ${APP_DIR}            # extracts into image's fs
COPY package.json package.json          # copies into image's fs
```

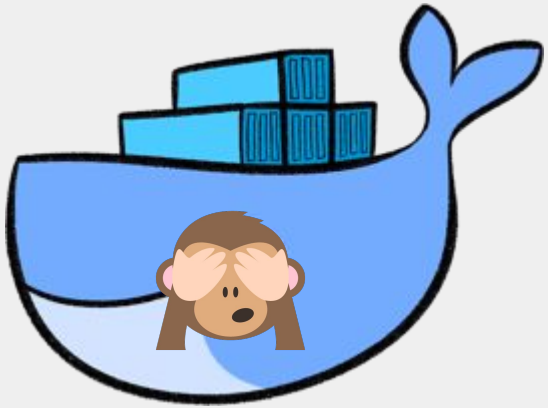
Most used commands

```
RUN yarn install          # executes commands and commits
                          # results
USER node                 # sets user for following commands
ENTRYPOINT node app.js    # allows container to run as executable
CMD my_apps_args          # can be executable or params
EXPOSE 3000               # container listens on this port in run-time
```

Best Practices

.dockerignore

The directories and files added will not be copied.



Dockerfile:

```
COPY . .
```

Execute:

```
.git
.gitignore
node_modules
*.log
coverage
.env
Dockerfile
README.md
```

FROM

Sets the base image taken from an either public- or private repository or from the local system.

Dockerfile:

```
FROM <image>:<tag> AS <name>
```

Examples:

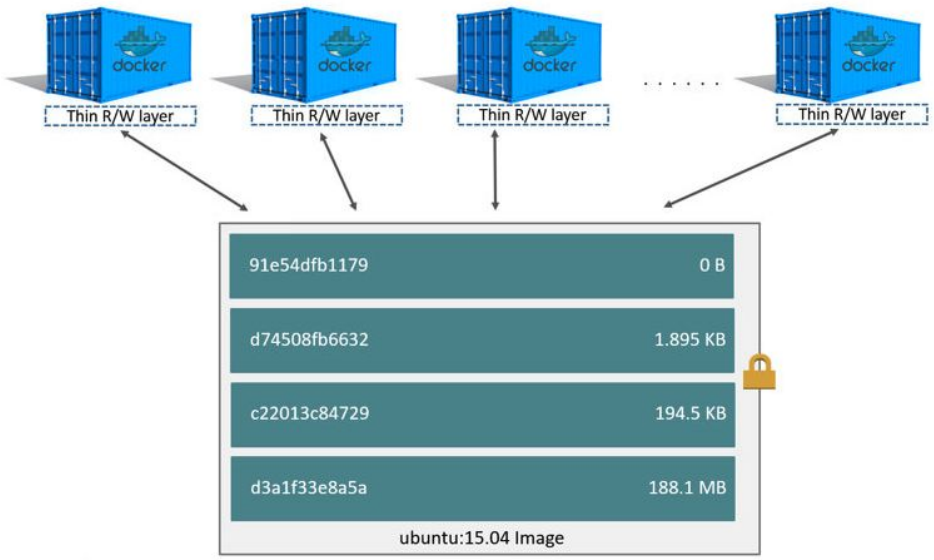
```
1: FROM golang

2: FROM golang:latest

3: ARG OS_VERSION=”1.9.1”
   FROM golang:${OS_VERSION}

4: FROM golang:1.9.1 AS builder
```

Image Layers

A Docker image is built up from a series of layers.

Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only.

In Docker 1.10+, only RUN, COPY, and ADD create image layers.



Docker History Command

big-backend layers:

```
$ docker history big-backend
IMAGE               CREATED             CREATED BY                                      SIZE
cb33107b777a        4 seconds ago       /bin/sh -c #(nop)  ENTRYPOINT ["./bin/api"]     0B
2af2d3c58ff2        5 seconds ago       /bin/sh -c make build                           7.92MB
3cdd91765b59        8 seconds ago       /bin/sh -c make dep                             50.9MB
4337e44d1aa8        29 seconds ago      /bin/sh -c go get github.com/Pepegasca/goop     38MB
...
<missing>           46 hours ago        /bin/sh -c set -eux;   dpkgArch="$(dpkg --...   298MB
<missing>           46 hours ago        /bin/sh -c #(nop)  ENV GOLANG_VERSION=1.9.2     0B
...
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:a71e077a42995a6...   100MB
```

Image Cache

Dockerfile:

```
ARG OS_VERSION=”15.04”
FROM ubuntu:${OS_VERSION}
RUN apt-get update
RUN apt-get install -y python3
COPY app.py app.py
ENTRYPOINT python3 app.py
```

Optimized Dockerfile:

```
ARG OS_VERSION=”15.04”
FROM ubuntu:${OS_VERSION}
RUN apt-get update \
    && apt-get install -y python3 \
    && apt-get clean \
    && apt-get autoclean
COPY app.py app.py
ENTRYPOINT python3 app.py
```

Multi-stage Build

In Docker 17.05+, you can do multi-stage builds and copy only artifacts into the final image.

Dockerfile:

```
FROM golang:1.9.2 as builder
ENV CGO_ENABLED=0 GOOS=linux
RUN go build -a \
             -installsuffix cgo \
             -o app .

FROM alpine:3.6
WORKDIR /root/
COPY --from=builder app .
CMD ["./app"]
```

How would you improve it?

Backend/Dockerfile:

```
FROM golang
RUN mkdir /app
WORKDIR /app
ADD bin bin
ADD config config
ADD src src
ADD api_app.go api_app.go
ADD Makefile Makefile
RUN make dep
RUN make build
ENTRYPOINT ./bin/api
EXPOSE 8000
```

How would you improve it?

Optimized Dockerfile:

```
ARG OS_VERSION=”1.9.2”
FROM golang:${OS_VERSION}
WORKDIR /app
COPY . .
ENV CGO_ENABLED=0 GOOS=linux
RUN make dep && make build
ENTRYPOINT python3 app.py
EXPOSE 8000
```

.dockerignore:

```
.git
api_app.go
Goopfile
src
Dockerfile*
Makefile
build-docker.sh
run-docker.sh
README.md
```

How would you improve it?

Multi-stage Dockerfile:

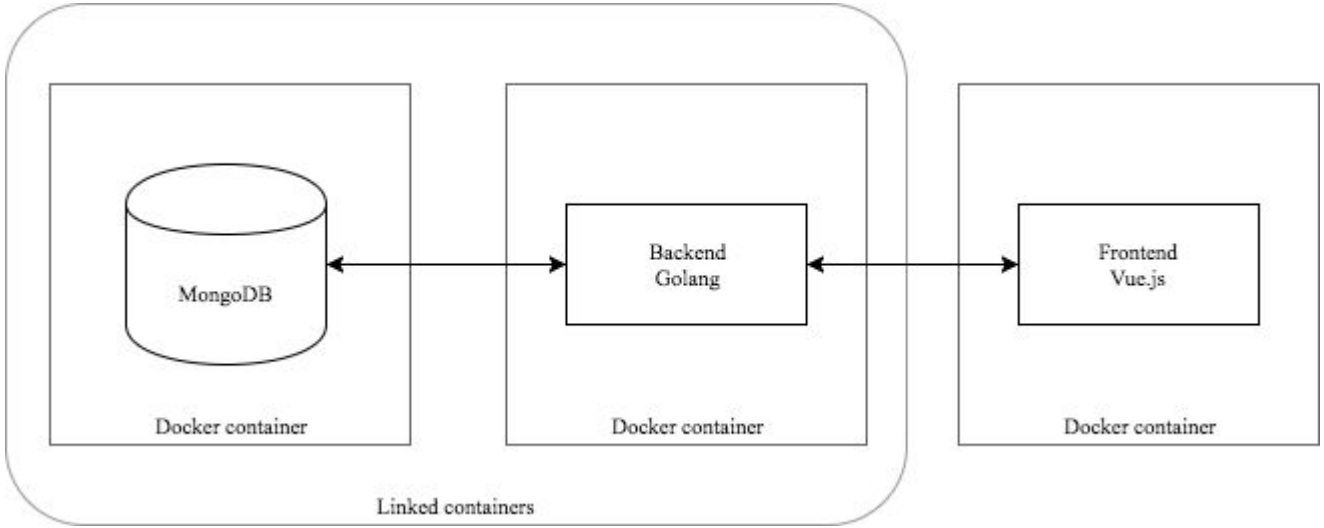
```
ARG OS_VERSION=”1.9.2”
FROM golang:${OS_VERSION} AS builder
WORKDIR /app
COPY . .
ENV CGO_ENABLED=0 GOOS=linux
RUN make dep && make build

ARG OS_VERSION=”3.6”
FROM alpine:${OS_VERSION}
WORKDIR /app
COPY --from=builder . .
EXPOSE 8000
CMD ./bin/api
```

.dockerignore:

```
.git
api_app.go
Goopfile
src
Dockerfile*
Makefile
build-docker.sh
run-docker.sh
README.md
```

Example Application Overview

TO-DO list application



Building an Application

$ docker build

- Creates a Docker image taking as input a Dockerfile and a context.
- By default, the Dockerfile is looked in the current working directory.

```
$ docker build .

$ docker build -t my-awesome-image .

$ docker build -t my-awesome-image -f Dockerfile.awesome .

$ docker build -t my-awesome-image -f Dockerfile.awesome  /path/to/context
```

Backend

```
$ docker build -t big-backend .
```

Frontend

```
$ docker build -t big-frontend .
```

Backend

```
$ docker build -t smaller-backend
-f Dockerfile.clean .
```

Frontend

```
$ docker build -t smaller-frontend
-f Dockerfile.clean .
```

Backend

```
$ cd Backend
$ docker build -t todo-backend -f
Dockerfile.build .
```

Frontend

```
$ cd Frontend
$ docker build -t todo-frontend -f
Dockerfile.build .
```

$ docker images

Shows the Docker images stored in the local system.

```
REPOSITORY      TAG       IMAGE ID       CREATED              SIZE
todo-backend    latest    1b244ebfc07a   50 seconds ago       792MB
<none>          <none>    6e66fca3aa1c   13 minutes ago       361MB
<none>          <none>    d3149416710a   19 minutes ago       24.7MB
<none>          <none>    45dd9a5213d3   19 minutes ago       797MB
mongo           latest    a28fdc58a538   22 hours ago         361MB
```

$ docker tag

- Tags a local Docker image.
- It is normally used to set the version or as an identifier.
- If not set, by default the images are tagged as `latest`.

```
$ docker tag <image> <repository/name:tag>

$ docker tag 1b244ebfc07a wizelineacademy/todo-backend:1.0.0
$ docker tag 9d8d557dc540 wizelineacademy/todo-frontend:1.0.0

$ docker images
REPOSITORY                       TAG     IMAGE ID      CREATED          SIZE
wizelineacademy/todo-backend     1.0.0   1b244ebfc07a  2 minutes ago    24.7MB
wizelineacademy/todo-frontend    1.0.0   9d8d557dc540  3 minutes ago    17MB
```

$ docker rmi

Deletes Docker images from the local system storage.

```
$ docker rmi <image>

$ docker rmi 1b244ebfc07a
Untagged: wizelineacademy/todo-backend:1.0.0
Deleted: sha256:1b244ebfc07a103f25d2b5e824233a64e95f7127dcfdccbaef9c141bdb68381e
Deleted: sha256:f1fd51236c4914aaf4ebbbcd53765616cc57edc339ec905cfd1cfc3388c34663
Deleted: sha256:0011989a8e723b51be94d57440df74d9956b33925dd25e1689e96b5f5fc6b676
Deleted: sha256:6a48a266f109296d42dd382fe9e724f55333fc868e55f523ece531b753615309
```

Running an Application

$ docker run

- Spins up new containers from already built images.
- If the image is not in the local system, it tries to pull it from the Docker Hub.

```
$ docker run busybox
$ docker run --name mybusybox busybox
$ docker run -ti --name mybusybox busybox
$ docker run --rm -ti --name mybusybox busybox
$ docker run --rm -ti --name mybusybox -p 80:80 busybox
$ docker run -d --name mybusybox -p 80:80 busybox
$ docker run --rm -ti --name mybusybox busybox bash
```

Run the application

Mongo

```
$ docker run -d -p 27017:27017 --name sample-mongo mongo
```

Backend

```
$ docker run -it --rm --name todo-backend -p 8000:8000
    --link sample-mongo:mongodb wizelineacademy/todo-backend:1.0.0
```

Frontend

```
$ docker run -it --rm --name todo-frontend -p 80:80
wizelineacademy/todo-frontend:1.0.0
```

$ docker ps

Shows information about the containers running on the system.

```
CONTAINER ID  IMAGE                                 COMMAND                  PORTS                     NAMES
ba49614819e0  wizelineacademy/todo-frontend:1.0.0  "nginx -g 'daemon ..."   0.0.0.0:80->80/tcp        todo-frontend
8a083a4d50ee  wizelineacademy/todo-backend:1.0.0   "./bin/api"              0.0.0.0:8000->8000/tcp    todo-backend
38e8ac4d76b6  mongo                                 "docker-entrypoint..."   0.0.0.0:27017->27017/tcp  sample-mongo
```

$ docker stop

Stops running containers on the system.

```
$ docker stop <container-id>

$ docker stop <container-name>


$ docker stop 38e8ac4d76b6

$ docker stop sample-mongo
```

$ docker rm

- Deletes a stopped container from the local system.
- Only the container, not the image.

$ docker rm <container-id>

$docker rm <container-name>

$ docker rm 38e8ac4d76b6

$ docker rm sample-mongo

Docker Registries

$ docker pull

- Pulls an image from a remote registry into our local system.
- By default, it pulls them from the Docker Hub.

```
$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
Digest:
sha256:3e8fa85ddfef1af9ca85a5cfb714148956984e02f00bec3f7f49d3925a91e0e7
Status: Downloaded newer image for busybox:latest
```

$ docker login

When working with private registries, you need to login to be able to pull and push Docker images.

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you
don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: 9chito9
Password:
Login Succeeded
```

$ docker push

- Once you built an image, you can push it into a public/private registry.
- By default, it’s pushed into the Docker Hub.

```
$ docker push wizelineacademy/todo-backend:1.0.0
The push refers to a repository [docker.io/wizelineacademy/todo-backend]
bf3fb9724ab8: Layer already exists
5bef08742407: Layer already exists
1.0.0: digest: sha256:ecffc60939b33f9019885f819...b065d0c130d94 size: 740
```

$ docker logout

- Log outs from the registry.
- Removes your registry account from your local system.

```
$ docker logout
Removing login credentials for https://index.docker.io/v1/
```

Advanced Docker

Advanced Docker

- Docker Volumes
- Docker Networks
- Docker Events
- Q&A

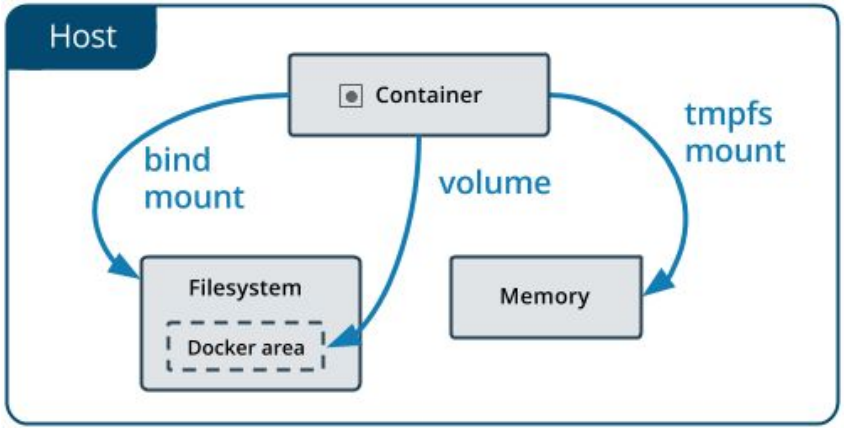
Docker Volumes

If you ever have wondered how you can persist data in a container, or how to share data among containers, the answer you are looking for is: volumes

There are three kind of volumes

- Volume
- Bind mount
- tmpfs

Storage drivers are out of the scope of this workshop



All kinds of volumes

You can mount them either in read-only or in read-write modes

Volumes can be shared among several containers at a time

Bind Mount

It shared a directory from the host

You can bind mount single files

Always obscures the data of the mount point

Volumes

Empty volumes propagates the data of the mount point

Non empty volumes obscures the data of the mount point

tmpfs

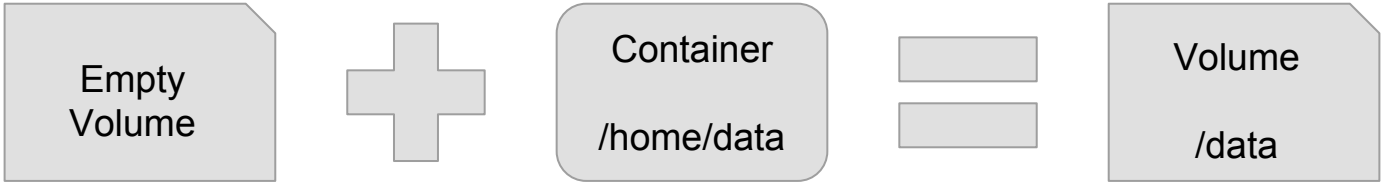
This kind of volume is non persistent

Only supported on Linux containers

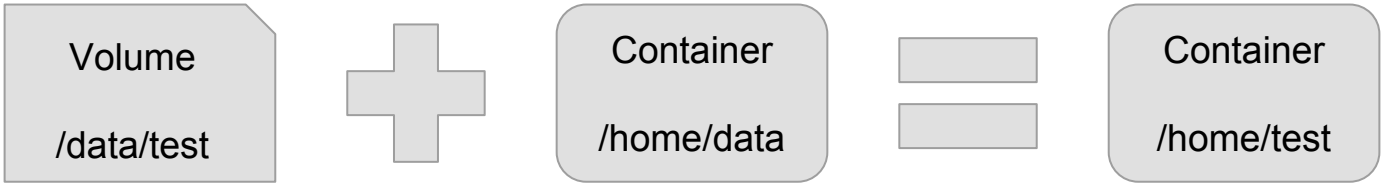
Always obscures the data of the mount point

Propagation and Obscuring

```
$ docker run -d --mount type=volume,target=/home image:latest
```




```
$ docker run -d --mount source=/data,destination=/home image:latest
```



Docker Volumes Commands

List volumes

```
$ docker volume ls
DRIVER              VOLUME NAME
local               ec881e0daf725b9c5a28d10cec305fee4225d249519941aa6a78969b6e5cf231
local               my-vol
```

Create a named volume

```
$ docker volume create my-vol
my-vol
```

Create an anonymous volume on container start

```
$ docker run -d --mount type=volume,target=/etc/nginx nginx
$ docker run -d -v /etc/nginx nginx
```

Create an tmpfs volume with size of 100MB and uid set to 1000

```
$ docker volume create --opt type=tmpfs --opt device=tmpfs --opt o=size=100m,uid=1000
my-tmpfs-vol
```

Docker Volumes Commands

Attach a bind mount volume

```
$ docker run -d --mount src=/home/myuser/nginx,dst=/usr/share/nginx/html nginx
$ docker run -d -v /home/myuser/nginx:/usr/share/nginx/html nginx
```

Attach a read-only volume

```
$ docker run -d --mount src=my-vol,dst=/usr/share/nginx/html,readonly nginx
```

Attach a read-only file as a bind mount volume

```
$ docker run -d --mount src=/data/nginx.conf,dst=/etc/nginx/nginx.conf,readonly nginx
$ docker run -d -v /data/nginx.conf:/etc/nginx/nginx.conf:ro nginx
```

Remove all unused volumes (destructive)

```
$ docker volume prune
```

Docker Networks

There are three default networks

- bridge (default)
  - Every container in this network can communicate with each other by IP *
- none
  - Full network isolation
- host
  - The container is on the host's network stack

* The default bridge connection does not provides automatic service discovery

You can create user defined bridge networks.

- It provide DNS resolution of container names
- All containers in the network can communicate
- The network is isolated from external networks
- You can publish ports to make them available
- You can't link containers within the network

Docker Networks Commands

List the available networks

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
93563b3da1c5        bridge              bridge              local
253d586adc33        host                host                local
2480ffb2426d        none                null                local
```

Create a new user-defined bridge network

```
$ docker network create --driver bridge isolated_nw
```

Attach a container to the user-defined network

```
$ docker run -d --network=isolated_nw nginx
```

Other advanced networking topics in docker *(out of the scope of this workshop)*

- docker_gwbridge
- overlay
- Custom network plugins
- Proxy servers (Edge)

Docker Events

Docker has a way to expose real-time events from docker's server.

It is not just for monitoring the new and current events. You can also query past events (--since).

You can limit the time frame of the events as well (--until).

You can define filters to narrow down the events output:

- Action (create, destroy, die, pull)
- Type (container, image, network)
- Type's name (container=frontend, image=nginx)

You can also change the output format:

- Select fields (Type, Action)
- Output as JSON lines

Docker Events Commands

Real-time, since and until

Return new/real-time events

```
$ docker events
```

Return new events and all events since the specified time

```
$ docker events --since 2017-10-28
```

Return past events between the specified time range

```
$ docker events --since  2017-10-28T10:00:00-05:00 --until 2017-10-28T12:00:00-05:00
```

Docker Events Commands

Filter and format

Return all the create events (container, volume, network)

```
$ docker events --filter 'event=create'
```

Return all the events of the image object (pull, delete, tag)

```
$ docker events --filter 'type=image'
```

Return all the create events of the container object

```
$ docker events --filter 'type=container' --filter 'event=destroy'
```

Return just the fields Type and Action

```
$ docker events --format 'Type={{.Type}} Action={{.Action}}'
```

Return the events output in JSON lines format

```
$ docker events --format '{{json .}}'
```

Docker Events Sample

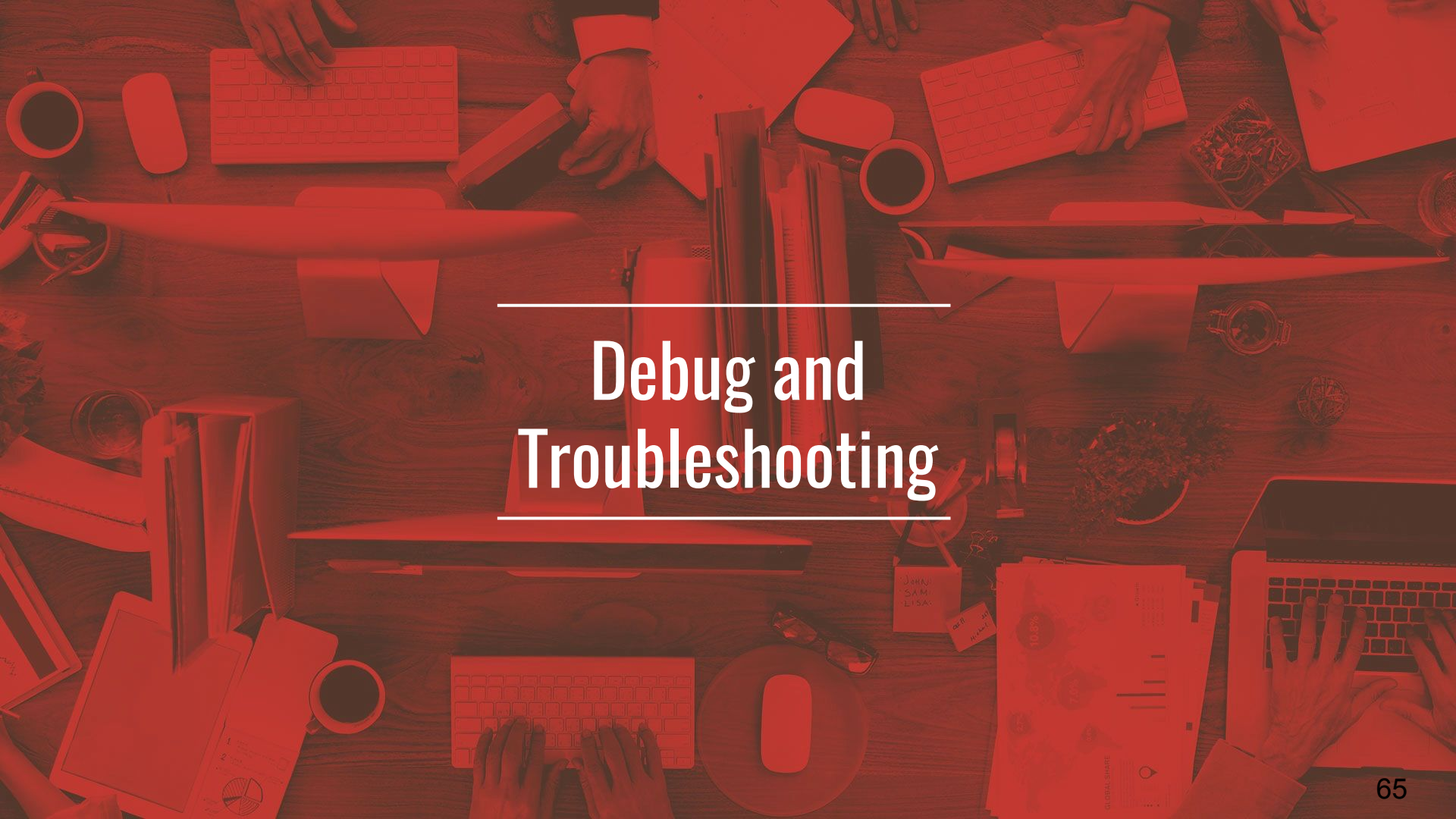
Terminal 1

```
$ docker events


2017-10-28T12:00:00.000000000-05:00 image
pull alpine:latest (name=alpine)


2017-10-28T12:01:00.000000000-05:00 image
untag sha256:76da55c8019d7a47c347c0dce...
...

2017-10-28T12:01:01.000000000-05:00 image
delete sha256:76da55c8019d7a47c347c0dce...
...
```

Terminal 2

```
$ docker pull alpine

Using default tag: latest
latest: Pulling from library/alpine
...
Status: Downloaded newer image for
alpine:latest


$ docker rmi alpine

Untagged: alpine:latest
Untagged: alpine@sha256:f006ecbb824d87947...
Deleted: sha256:76da55c8019d7a47c347c0dce...
Deleted: sha256:5bef08742407efd622d243692...
```

Q&A



Debug and Troubleshooting

Debug & Troubleshooting

- **Checklist**
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (OS X)
- Cleanup

Troubleshoot checklist

Check docker.d and docker client version
docker version

Check image architecture compatibility
Images have to be built in their intender architecture e.g. ARM (raspberrypi) images will not run in x86

For daemons, check the main process (CMD) is not exiting
Container process is being terminated thus detached container is not behaving as a daemon.

Debug & Troubleshooting

- **Checklist**
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (OS X)
- Cleanup

Troubleshoot checklist

Check where the image was built (Cross OS Images)
Images built in linux, may not work in windows and the other way around.

OS X filesystem is not case sensitive by default, it may break your image.

Check output is directed to stdout and stderr
Only necessary output to stdout and stderr so docker logs can show it.

Check that container is persisting data in volumes
Do not persist data inside the container (stateless), data may be lost, instead use a volume.

Debug & Troubleshooting

- **Checklist**
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (OS X)
- Cleanup

Troubleshoot checklist

Volumes in different file systems must be created before docker starts

Check you have the correct proxy configuration
You may not be able to download images or have connectivity problems if proxy configuration is incorrect

Check you are logged to the docker registry
You will not be able to pull private images if not logged in to the registry

Most frequent stackoverflow docker questions

Debug & Troubleshooting

- Checklist
- **Most common pitfalls**
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (os x, opcional)
- Cleanup

Most common pitfalls

docker: command not found

- Make sure docker is installed

docker daemon not running? - Check

- Docker daemon is running
- Your permissions

docker container can not be accessed through the published ports? - Check

- You have published the port
- Your listening ports are correct
- Access through the correct IP address.
- Firewall rules are in place.

Debug & Troubleshooting

- Checklist
- **Most common pitfalls**
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (os x, opcional)
- Cleanup

Most common pitfalls

docker pull takes too much - Check

- Size of the image
- You have a reliable internet connection

docker containers can not start - Check

- Your logs
- Your disk space
- Your docker values

docker filesystem is full, containers won't run or will die.

- *Have a dedicated filesystem for docker (/var/lib/docker)*
- *Avoid sending too much text to stdout, stderr*

Debug & Troubleshooting

- Checklist
- Most common pitfalls
- **Logs**
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (OS X)
- Cleanup

Docker Log

Console logs can be accessed with the command

```
docker logs <container>
docker logs -f <container>
docker logs -f --tail 10 <container>
```

Additionally logs can be accessed in json format in

```
/var/lib/docker/containers/<id>/<id>-json.log
```

Copy Logs from container to your host machine

```
docker cp <id>:/path/to/file /destination
```

Debug & Troubleshooting

- Checklist
- Most common pitfalls
- Logs
- **Log into the container**
  - **Attach**
  - **Exec**
- Inspect and Stats
- Accessing moby image (OS X)
- Cleanup

Log Into the Container

Attaching

```
docker attach <container>
```

To avoid sending unwanted signals to the container use the following flag

```
docker attach --sig-proxy=false <container>
```

To detach CTRL-p CTRL-q
To exit, just type exit, logout, CTRL-d

Executing

```
docker exec -ti <container> /path/to/shell
```

Main difference between attach an exec is that exec spawns a new tty, while attach reuses the current tty

Debug & Troubleshooting

- Checklist
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- **Inspect and Stats**
- Accessing moby image (OS X)
- Cleanup

Docker Inspect

Very useful to identify the properties and settings of a container

```
docker inspect <container>
```

- Entrypoint
- Image
- Network Info / Driver
- Volumes
- Container

Docker Stat

```
docker stats <container>
```

- CPU
- Memory
- Net I/O
- Block I/O
- Number of Processes

Debug & Troubleshooting

- Checklist
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- **Accessing moby image (OS X)**
- Cleanup

This won't work with tmux

```
screen
~/Library/Containers/com.docker.docker/Da
ta/com.docker.driver.amd64-linux/tty
```

```
docker run -it --privileged --pid=host
debian nsenter --target 1 --mount --uts
--net --ipc sh
```

Debug & Troubleshooting

- Checklist
- Most common pitfalls
- Logs
- Log into the container
  - Attach
  - Exec
- Inspect and Stats
- Accessing moby image (OS X)
- **Cleanup**

Docker Remove

Remove Containers and Images

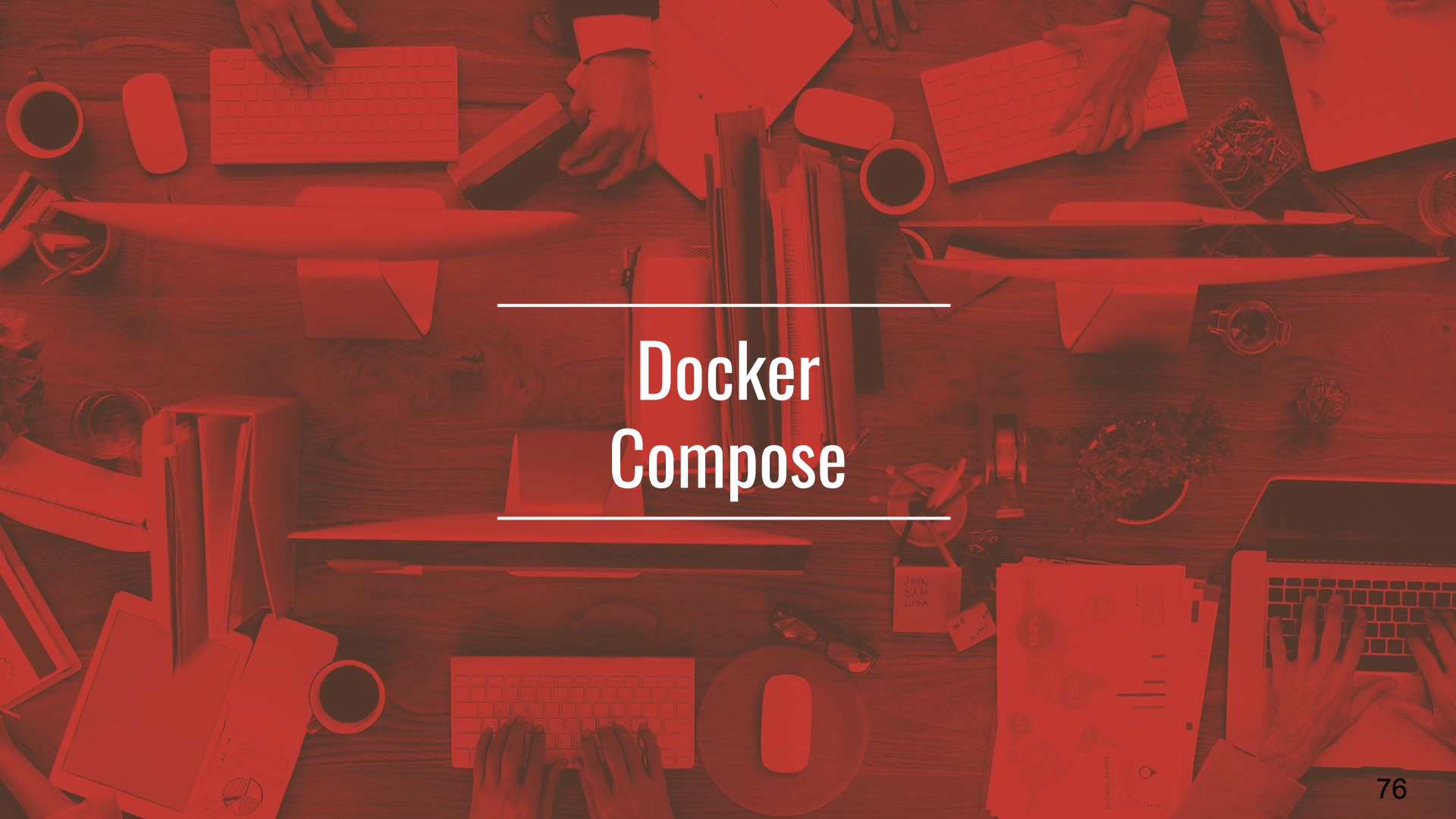
```
docker rm <container>
docker rmi <image>
docker rmi -qa -f dangling=true
docker rmi -f $(docker images -qa)

docker <object> prune

docker ps -qa
docker ps -qa -f dangling=true
```

Stop all containers:

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

Remove all containers:

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



Docker Compose

Docker Compose

- Overview
- Installing Docker Compose
- Getting Started
- Understanding Docker Compose Use Cases
- Tips and tricks

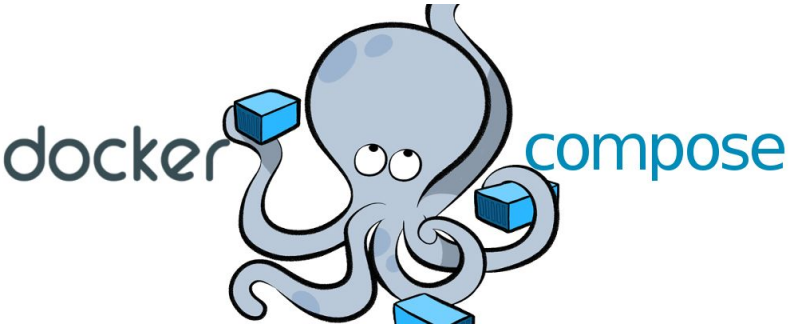
Overview

Overview

History

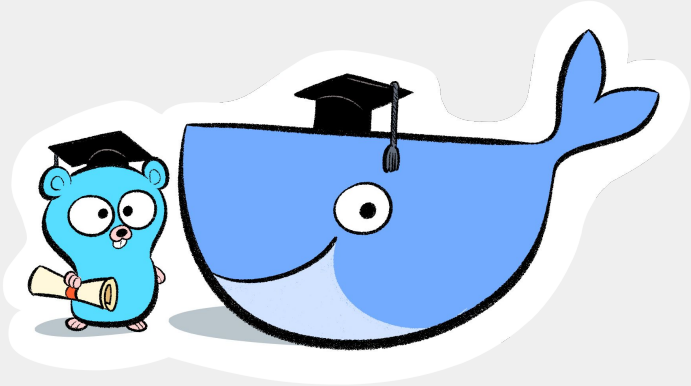
- Originally started as fig
- Was so popular that got merged into docker in 2015

Fig



Overview

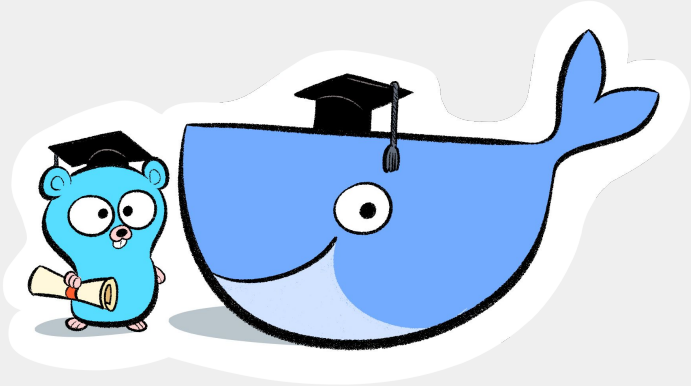
What's a Service?



- Different pieces/components of an app are called "services."
- A service is a "container."
- A service only runs one image.

Overview

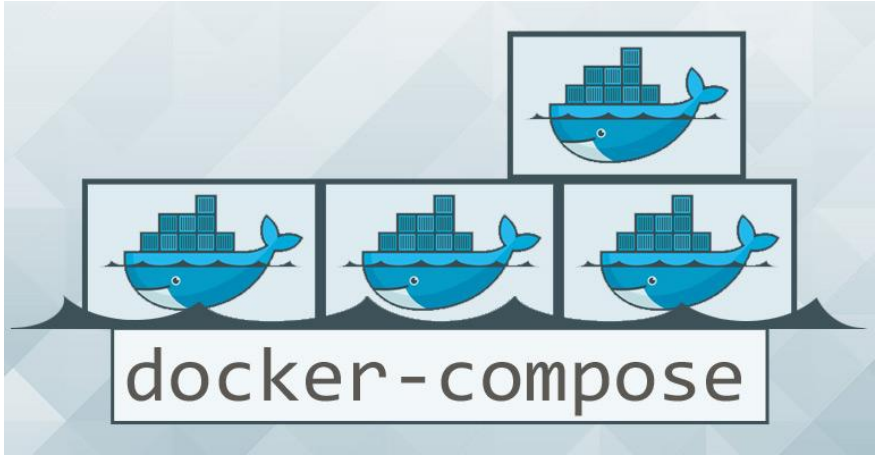
What's a Service?



- A service defines the way an image runs:
  - Ports
  - Volumes
  - Replicas
    - Capacity
  - Etc.

Overview

- Compose allows you to define and run multiple containers, all by using a single command.
- You use YAML files to configure all needed dependencies/services.
- You can start/stop services through configuration files.



Overview

It is possible to use

multi-environment configuration

- Development, Staging,

  Production, testing, etc.

Installing Docker Compose

Installing Docker Compose

Requirements

- Docker Engine
  - Already included for desktop installs

MAC:

https://docs.docker.com/docker-for-mac/install/

WINDOWS:

https://docs.docker.com/docker-for-windows/install/

Installing Docker Compose

Requirements

- **Docker Engine**
  - **Already included for desktop installs**

LINUX:

```
$ sudo curl -L
https://github.com/docker/compose/releases/download/1.16.1/docker-c
ompose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose
```

Installing Docker Compose

Verify installation

```
$ docker-compose --version
```

```
docker-compose version 1.16.1,

build 6d1ac21
```

Getting Started

Getting Started

Docker compose can manage the whole lifecycle of your app

- Run commands
- See the output of logs
- View status of current apps
- Start, stop, rebuild services

Getting Started

Useful commands you should know

```
$ docker-compose --help
```

Getting Started

Build

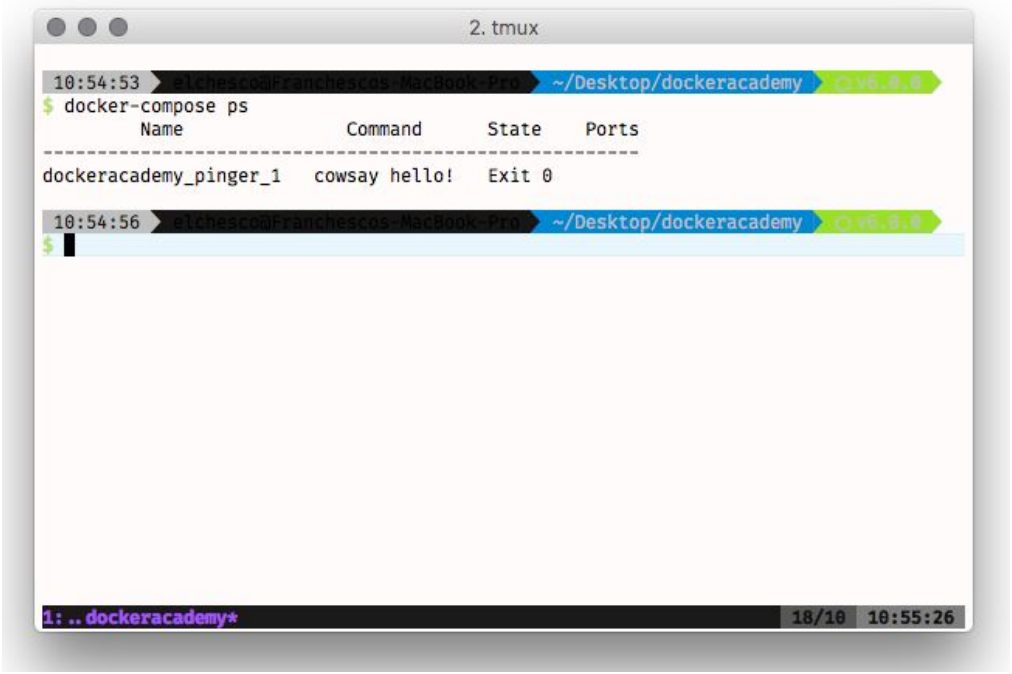
```
$ docker-compose build
```

```
Usage: build [options] [--build-arg key=val...]
[SERVICE...]
```

Getting Started

List containers

```
$ docker-compose ps
```

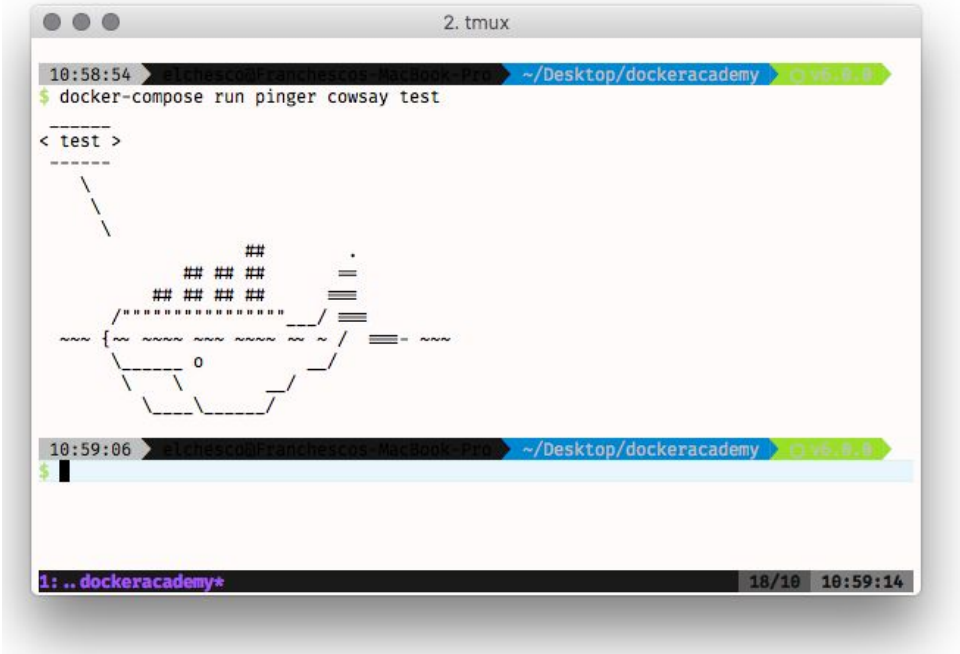



```
Usage: ps [options] [SERVICE...]
```

Getting Started

Run command(s) against container

```
$ docker-compose run pinger cowsay test
```




```
Usage: run [options] [-v VOLUME...] [-p PORT...] [-e
KEY=VAL...] SERVICE [COMMAND] [ARGS...]
```

Getting Started

See logs

```
$ docker-compose logs -f
```


```
Usage: logs [options] [SERVICE...]
```

Getting Started

3-Step process

1. Define a `DockerFile`
2. Define services for your app in `docker-compose.yml`
3. Run `docker-compose up`

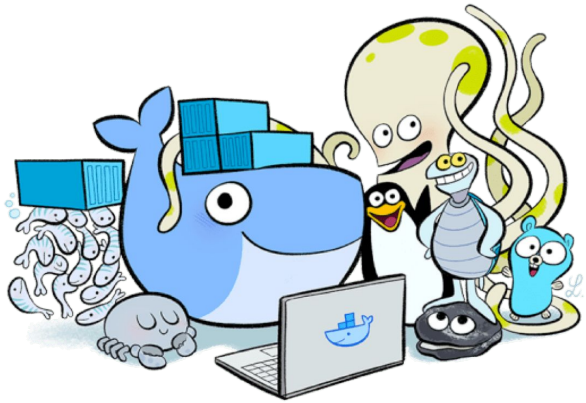
Getting Started

Dockerfile

```
FROM docker/whalesay:latest


RUN apt-get -y update && apt-get install -y fortunes


CMD /usr/games/fortune -a | cowsay
```

Getting Started

Define services



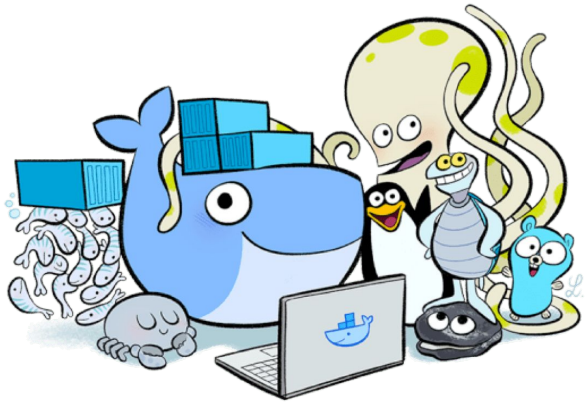
```
$ cat docker-compose.yml
version: "3"

services:
 pinger:
    image: docker/whalesay
    command: ["cowsay", "hello!"]
```

Getting Started

Running containers

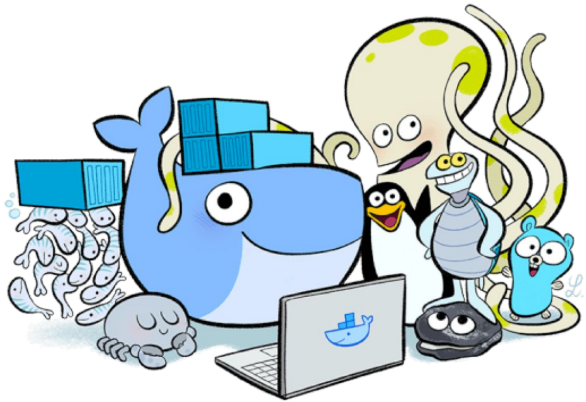
```
$ docker-compose up
```

- Aggregates the output of
  - $docker-compose logs -f



Getting Started

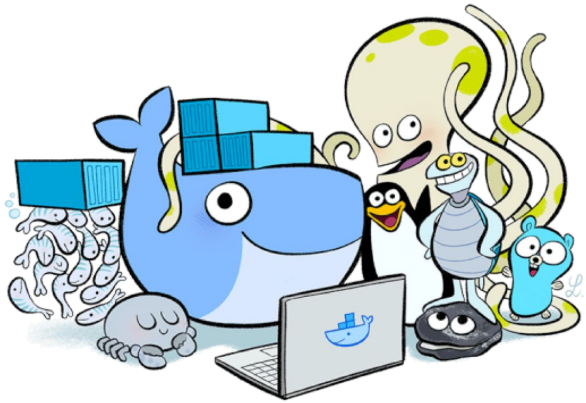
Compose up



```
$ docker-compose up
Creating network "dockeracademy_default" with the
default driver
Creating dockeracademy_pinger_1 ...
Creating dockeracademy_pinger_1 ... done
Attaching to dockeracademy_pinger_1
pinger_1  |  ________
pinger_1  | < hello! >
pinger_1  |  --------
pinger_1  |     \
pinger_1  |      \
pinger_1  |       \
pinger_1  |                    ##         .
pinger_1  |              ## ## ##        ==
pinger_1  |           ## ## ## ##       ===
pinger_1  |       /""""""""""""""""___/ ===
pinger_1  |  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===-
~~~
pinger_1  |       \_____ o           __/
pinger_1  |        \    \         __/
pinger_1  |          \____\_____/
dockeracademy_pinger_1 exited with code 0
```

Getting Started

Pass environment variables



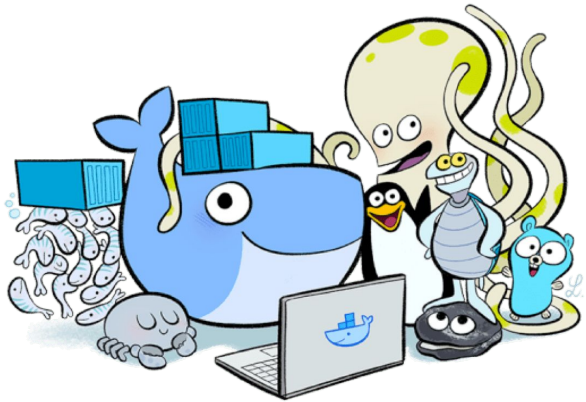
```
$ cat docker-compose.yml
version: "3"

services:
  pinger:
    image: "docker/whalesay:${TAG}"
    environment:
    - DEBUG=true
    - USER
    command: ["cowsay", "hello!"]
```

Getting Started

Run on background

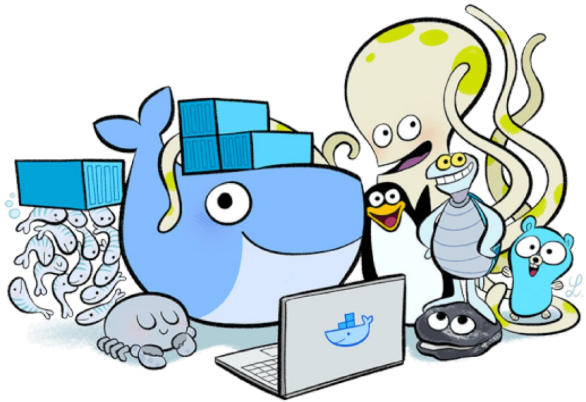
```
$ docker-compose up -d
Starting dockeracademy_pinger_1 ...
Starting dockeracademy_pinger_1 ... done
```

- Detached mode



Getting Started

Stop vs down vs pause



`$ docker-compose down`

- Stops containers
  - Removes :
    - Containers
    - Networks
    - Volumes
    - images

`$ docker-compose stop`

- Stops containers
  - Without removing any of them

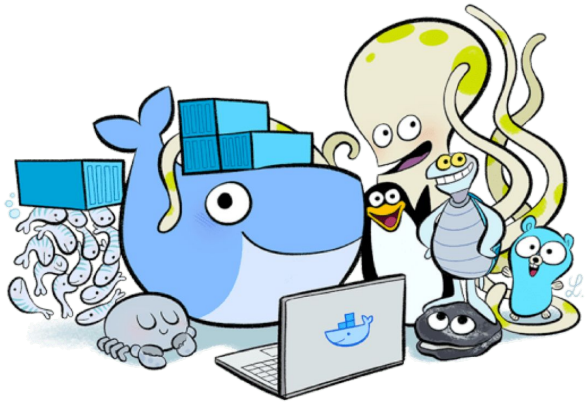
`$ docker-compose pause`

- Pauses running containers of a service

Getting Started

What if we need to re-deploy code?

```
$ docker-compose build web
$ docker-compose up --no-deps -d web
```



- --no-deps
  - Don’t start linked services

Getting Started

Converting our app to be docker compose ready.

Let's do a step by step conversion.

Conversion

01 First version
02 Basic build
03 Adding ports
04 Commands + dependencies
05 Adding container name
06 Adding environment variables
07 Adding configuration
08 Healthcheck + Logging
09 Adding more settings + restart policy
10 Putting everything together

01

First version

- `git checkout docker-compose`

01

First version

- **Helpful files**
  - **cat run-docker-compose.sh**
  - **ls docker-compose-backend.yml**
    - docker-compose-backend.xx.yml
  - **ls docker-compose-frontend.yml**
    - docker-compose-frontend.xx.yml

01

First version

- `Helpful files`
  - `xx-backend.diff`
  - `xx-frontend.diff`

01

First version

- Use latest version
  - No version = v1 = legacy
- Use docker-compose config
  - Validate and view Compose file

01

First version

```
$ cat docker-compose-backend.01.yml
version: "3.2"


$ cat docker-compose-frontend.01.yml
version: "3.2"


$ sh run-docker-compose.01.sh
********************
services: {}
version: '3.2'
```

02
Basic build

- **Build**
  - **Supports contexts**
    - **Using a string path**
  - **Context:**
    - **What if the build is on a different machine than *dockerd***
  - **build:**
    context: ./dir

02
Basic build

- **Build**
  - **Combining build and image is possible**
    - **Ends up build on dir using name = image**
  - **build: ./dir**
    image: webapp:tag

02
Basic build

- **Build**
  - **It is possible to use a different Dockerfile**
  - **build:**

```
context: .
dockerfile: Dockerfile-alternate
```

02

Basic build, *diff: 01*

```
$ cat docker-compose-backend.02.yml
version: "3.2"
services:

  docker-academy-backend:
    build:
    context: ./Backend

$ cat docker-compose-frontend.02.yml
version: "3.2"
services:

  docker-academy-frontend:
    build:
    context: ./Frontend
```

02

Run step

```
$ sh run-docker-compose.02.sh
```

03

Adding ports

- Ports
  - Always use strings: "xx:xx"
    - Yaml uses base 60
  - It is possible to specify
    - Target
      - Port inside container
    - Published
      - Publicly exposed port
    - Protocol
      - Tcp or udp

03

Adding ports, *diff: 02*

```
$ cat
docker-compose-frontend.
03.yml
version: "3.2"
services:


docker-academy-frontend:
    ...
    ports:
      - "80:80"
```

```
$ cat
docker-compose-backend.03.yml
version: "3.2"
services:

  docker-academy-backend:
    ...
    ports:
      - "8000:8000"
```

03

Run steps

```
$ sh run-docker-compose.03.sh
```

04

Command + deps

- Command
  - Overrides the default command
- Depends on
  - Notes dependencies between services
    - Will create and start services as needed
    - Will not wait by default for the *internal service* to fully come up

04

Command + deps, *diff: 03*

```
$ cat
docker-compose-backend.04.
yml
version: "3.2"
services:

  docker-academy-backend:
    ...
    ports:
      - "8000:8000"
    command: ["./bin/api"]
```

```
$ cat
docker-compose-frontend.04.yml
version: "3.2"
services:

  docker-academy-frontend:
    ...
    ports:
      - "80:80"
    depends_on:
      - "docker-academy-backend"
```

04

Run step

```
$ sh run-docker-compose.04.sh
```

05

Adding container name

- Container name
  - Specifies a custom container name
    - Overrides default
  - Names must be unique

05

Adding container name, *diff: 04*

```
$ cat docker-compose-frontend.05.yml
…
container_name: docker-academy-frontend
build: .
…
```

```
$ cat docker-compose-backend.05.yml

…
container_name: docker-academy-backend
build: .
…
```

05

Run step

```
$ sh run-docker-compose.05.sh
```

06

Adding environment variables

- `Environment`
  - `Adds environment variables`
  - `YAML format specific`

06

Adding environment variables, *diff: 05*

```
$ cat docker-compose-backend.06.yml
…
    - "8000:8000"
environment:
    - CGO_ENABLED=0
    - GOOS=linux
…
```

06

Run step

```
$ sh run-docker-compose.06.sh
```

07

Adding config

- DNS
  - Adds custom dns servers

07

Adding config,, *diff: 06*

```
$ cat docker-compose-frontend.07.yml
…
  - "8000:8000"
dns:
  - 8.8.8.8
  - 9.9.9.9
…
```

```
$ cat docker-compose-backend.07.yml
…
  - "8000:8000"
dns:
  - 8.8.8.8
  - 9.9.9.9
…
```

07

Run step

```
$ sh run-docker-compose.07.sh
```

08

Healthcheck + Logging

- **Healthcheck**
  - **Determines whether or not a service is healthy**
- **Logging**
  - **Logs using specific driver**

08

Healthcheck + Logging, *diff: 07*

```
$ cat docker-compose-backend.08.yml
…
    - GOOS=linux
healthcheck:
    test: ["CMD-SHELL", "curl -f http://0.0.0.0:8000 || exit
1"]
    interval: 1m30s
    timeout: 10s
    retries: 3
logging:
    driver: "json-file"
    options:
      max-size: "200k"
      max-size: "200k"
command: ["./bin/api"]
```

08

Healthcheck + Logging, *diff: 07*

```
$ cat docker-compose-frontend.08.yml
…
    - 9.9.9.9
healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost || exit
1"]
    interval: 1m30s
    timeout: 10s
    retries: 3
logging:
    driver: "json-file"
    options:
      max-size: "200k"
      max-size: "200k"
depends_on:
```

08

Healthcheck + Logging

```
$ cat Backend/Dockerfile.prod
…
FROM alpine:latest
RUN apk -U add curl && \
  rm -rf /var/cache/apk/*
COPY --from=docker-academy-backend-builder app app
…
```

08

Healthcheck + Logging

```
$ cat Frontend/Dockerfile.prod
…
FROM nginx:alpine
RUN apk -U add curl && \
  rm -rf /var/cache/apk/*
COPY --from=docker-academy-frontend-builder /app/dist
/usr/share/nginx/html
…
```

08

Run step

```
$ sh run-docker-compose.08.sh
```

09

Adding config

- ulimit & sysctl limit system-wide resources
  - Avoids unresponsiveness
- sysctl
  - Kernel params
- ulimit
  - Override default ulimits

09

Adding more settings + restart policy, *diff: 08*

```
$ cat docker-compose-backend.09.yml
…
    max-file:"10"
sysctls:
  net.core.somaxconn: 1024
ulimits:
  nproc: 65535
  nofile:
    soft: 20000
    hard: 40000
restart: on-failure
command: ["./bin/api"]
…
```

09

Adding more settings + restart policy, *diff: 08*

```
$ cat docker-compose-frontend.09.yml
…
    max-file:"10"
sysctls:
  net.core.somaxconn: 1024
ulimits:
  nproc: 65535
  nofile:
    soft: 20000
    hard: 40000
restart: on-failure
depends_on:
…
```

09

Run step

```
$ sh run-docker-compose.09.sh
```

10

Putting everything together

- Restart
  - Configures conditions on when to restart containers
    - On failure
- Build
  - Cache from
  - labels

10

Putting everything together

```
$ cat docker-compose.10.yml
...
 dockerfile: Dockerfile.prod
 cache_from:
   - nginx:alpine
   - mhart/alpine-node:8
 labels:
   com.example.description: "Frontend app"
   com.example.department: "DevOps Wizeline"
...
restart: on-failure
...
depends_on:
```

10

Run step

```
$ sh run-docker-compose.10.sh
```

Access your App

Open your browser to http://localhost

Features

Features

Backwards compatibility

- V1 = legacy
- No version = v1
- V2 still widely accepted
- Best practice: use the latest!

Docker Compose API version	Docker Daemon version
3.0+	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1+

Features

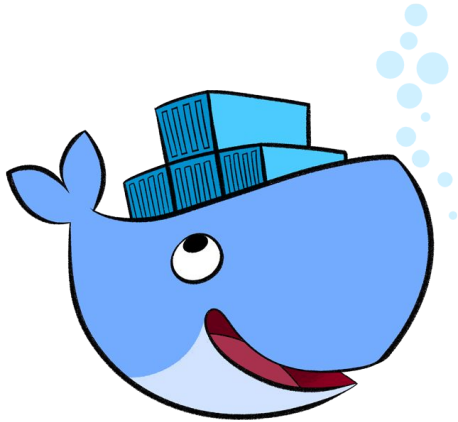
Preserve volume data

- When docker-compose up runs, it checks for any previous containers and copies them into new ones.
- Data from volumes is never lost.

Features

Container creation based on changes only

- Compose caches containers.
  - (when no changes are made)
- Re-using containers on a restart.



Features

Composition between environments

Docker reads these files by default:

- **docker-compose.yml**
  - **Base config**
- **docker-compose.override.yml**
  - **Config override**

Features

Limit usage

https://docs.docker.com/engine/admin/resourc

e_constraints/

```
deploy:
  resources:
    limits:
      cpus: '0.001'
      memory: 50M
    reservations:
      cpus: '0.0001'
      memory: 20M
```

Use Cases

Use Cases

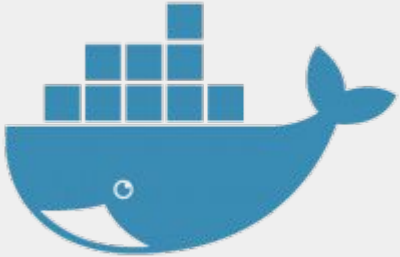
- Isolated test environments
- Dev test environments
- Integrate to Infrastructure as Code pipelines

```
$ docker-compose up -d

$ ./run_tests

$ docker-compose down
```

Tips and Tricks

Tips and Tricks

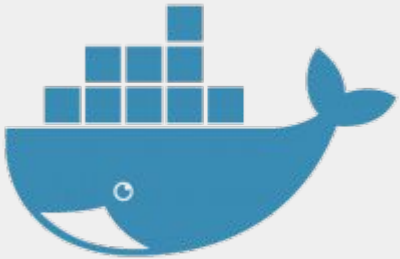
Use .json files for describing your services.



```
$ docker-compose -f
docker-compose.json up
```

Tips and Tricks

Use .json files for describing your services.

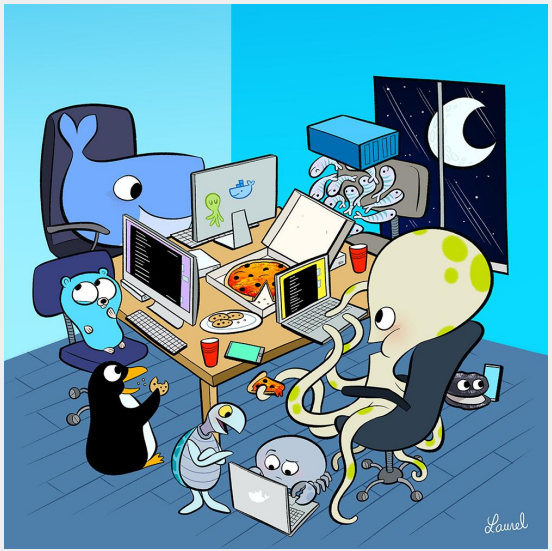


```
{
  "version": "2.1",
  "services": {
    "api": {
      "build": ".",
      "ports": [
        "8080:8080"
      ],
      "environment": [
        "DEBUG=$DEBUG"
      ],
      "command": "start api"
    }
  }
}
```

Tips and Tricks

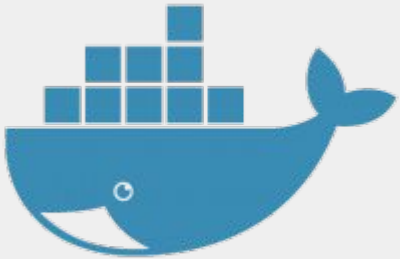
Initialize only one specific service.

```
$ docker-compose up redis
$ docker-compose up web
```



Tips and Tricks

Push your image by providing a tag and docker hub username.



```
$ cat docker-compose.yml

version: '3'
services:
  redis:
    image: 'redis:3.2-alpine'
  web:
    #build '.'
    image: 'elchesco/web:1.0'

...
```

Tips and Tricks

Declare default environment variables in file.

```
$ cat docker-compose.yml

version: '3'
services:
  redis:
    image: 'redis:3.2-alpine'
  web:
    build '.'
    depends_on:
    - 'redis'
    env_file:
    - '.env'
    - '.env.production'

...
```

Tips and Tricks

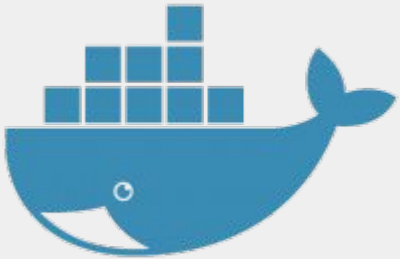
Declare default environment variables in file.

COMPOSE_PROJECT_NAME prefixes all your images with this value.

```
$ cat .env

COMPOSE_PROJECT_NAME=myproject
```

Tips and Tricks

Declare default environment variables in file.



- Declare each line in VAR=VAL format
- #Comments are ignored

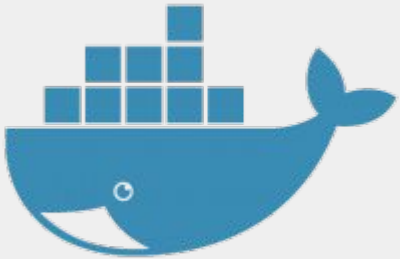
Tips and Tricks

Control Startup Order

```
$ cat docker-compose.yml
version: "2"
services:
  web:
    build: .
    ports:
      - "80:8000"
    depends_on:
      - "db"
    command: ["./wait-for-it.sh", "db:5432", "--", "python", "app.py"]
  db:
    image: postgres
```

Tips and Tricks

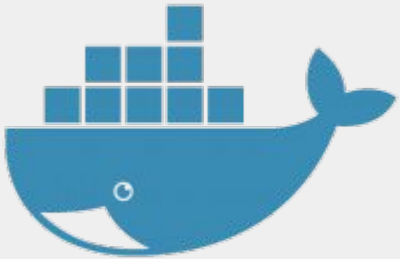
Deploy to prod



- Remove volume bindings
    - Code can’t be changed from the outside
- Bind to different ports
- Set env vars accordingly
- Restart policy to avoid downtime
- Enable extra services

Tips and Tricks

Deploy to prod



```
$ export APP_VERSION=1.2.3

$ docker-compose -f
docker-compose.yml -f
production.yml up -d
```


THANK
YOU
SURVEY:
bit.ly/dockeracademy
WIZELINE®
sales@wizeline.com

WIZELINE ACADEMY

Grow your career:
Free courses in Artificial Intelligence,
Software Development, User Experience and More

UPCOMING COURSES

SECRETS TO GETTING VENTURE CAPITAL: INVESTOR INSIGHTS FOR ENTREPRENEURS
Nov 1 @GDL

SECURITY VULNERABILITIES: HOW TO DEFEND AGAINST THEM
Nov 14 @GDL Nov 16 @CDMX

REACT FIBER: UPDATING YOUR CODEBASE
Nov 25 @GDL

@WizelineAcademy

academy.wizeline.com

/WizelineAcademy

Get notified about courses:
tinyurl.com/WL-academy