Introduction to Containers

or: How I Learned to Stop Worrying and Love the Container

Who are you people?

Mario Loria

@marioploria

marioploria@gmail.com

Jorge Castro

@castrojo

jorge@heptio.com

Today's TLDR;

You will learn:

Getting Started with Docker

Container Concepts

Pro-tips

Workshopping

Out of Scope for this session:

Orchestration (Kubernetes etc.)

CI/CD Pipelines

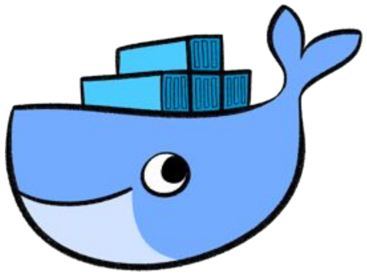
Advanced networking and storage

Application-level stuff

Before we Begin

Install Docker / Docker Compose

- **OSX:**
  https://store.docker.com/editions/community/docker-ce-desktop-mac
- **Windows:**
  https://store.docker.com/editions/community/docker-ce-desktop-windows
- **Linux:**
  See distribution repo



Containerization Concepts

Old Method

- Mutable systems
- Controlled via config management
- Static networking
- Imperative changes
- Probably monolithic

New Method

- Immutable artifacts
- Orchestrated by a system
- Dynamic networking
- Declarative configuration
- Encourages decoupled microservices

Don’t overcomplicate it. It’s basically just a zip file.

(There are technical details here, but think of it as an output.)

What is a Container?

- **Isolated**
  - Kernel handles application namespace separation
- **Fast**
  - Containers boot in milliseconds
- **Immutable**
  - Bundles an application, its dependencies, and other run-time requirements into an immutable, redistributable image.
  - Small compared to full fledged VM.
  - Because it’s repeatable, this makes an ideal distribution method also.

What a Container is Not

- **A VM**
  - Containers are not a VM. Docker is not a hypervisor.
  - If you want a container that is more like a VM, check out LXC.
- **Persistent**
  - Containers are ephemeral!
- **Secure by Default**
  - A container is not a security panacea
  - They **DO** have the capability of being more secure, but effort is required

"I like putting apps into containers because then I can pretend they're not my problem."

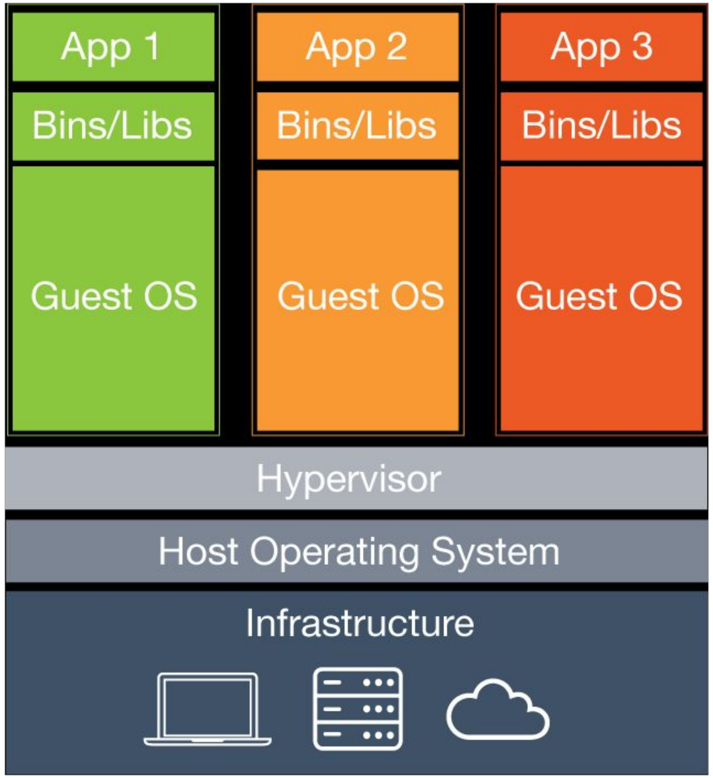
@sadoperator

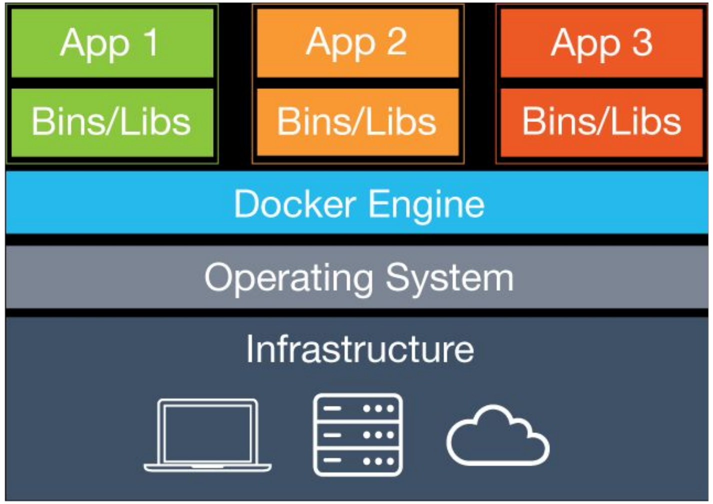
A Brief History

- **1982** - Unix/BSD chroot
- **2000** - BSD Jails
- **2005** - Solaris Zones
- **2008** - LXC
- **2013** - LMCTFY (Google)
- **2013** - Docker (formerly dotCloud)

What is a Container Under the Hood?

- Cgroups
  - Linux Kernel Feature
  - Manages groups of processes and resources
- Namespaces
  - ‘Contains’ aspects of host
  - PID, Net, IPC, and MNT
- Filesystem (image)
  - Hierarchical layered filesystem
  - Tar file





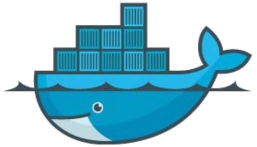
Source: docker.com

Container vs VM

What is Docker?

- Docker is **BOTH** a Company (Docker Inc.) and an Open Source Containerizer project.
- Available on multiple platforms and architectures.
- “Standard” when people think of container.

Docker Core Components

Engine	Image	Registry
		
● The Docker Daemon running on a host. ● Manages building, storing, and running images on a specific host.	● The item that is executed by the engine. ● Images consists of a manifest and collection of read-only layers generated by a Dockerfile.	● The service that acts as an image repository. ● Example: Docker hub

Docker Workflow

Build	**Ship**	**Run**
● Build, develop and join the components of your application together into an image or set of images.	● Push your image to a registry making it widely available.	● Deploy, manage and use the same containers your application was developed on in a production environment.

Before we start

Get in the ephemeral mindset:

- None of these commands can break your computer
  - Within reason. :D
- Learn to be comfortable throwing containers away
- You can always start over

Lets Run a Container!

```
$ docker run alpine echo hello from alpine!
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
ff3a5c916c92: Pull complete
Digest:
sha256:7df6db5aa61ae9480f52f0b3a06a140ab98d427f86d8d5de0bedab9b8df6b1c0
Status: Downloaded newer image for alpine:latest
hello from alpine!
```

Image is not found local
Pulls from docker hub

latest tag is default

Image is verified

Instance of image is run executing command:
`echo hello from alpine!`

What happened to our Container?

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              3fd9065eaf02        4 months ago        4.15MB
```

Image is pulled and stored locally

What happened to our Container?

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
27ef389a0bb9        alpine              "echo hello from alp…"   21 minutes ago



STATUS                         PORTS               NAMES
Exited (0) 22 minutes ago                          herp_derp
```

Daemonizing a Container

```
$ docker run -d -p 8081:80 nginx
```

Runs in background

Map port from container to host:
`-p <host port>:<container port>`

```
$ docker ps
STATUS                   PORTS                       NAMES
Up 3 minutes             0.0.0.0:8081->80/tcp        herp_derp
```

Let's get Interactive

```
$ docker run -it alpine /bin/sh
/ # hostname
cb7d3fa44cf0
/ #
```

-i - interactive session

-t - attaches a tty

```
$ docker exec -it herp_derp /bin/sh
/ # hostname
cb7d3fa44cf0
/ #
```

exec - ‘*executes*’ a command within a running container.

Viewing the Logs

```
$ docker logs herp_derp
```

Outputs stdout/stderr of the specified container

```
$ docker logs -f herp_derp
```

-f - follows the log output

```
$ docker logs -t herp_derp
```

-t - tails log output

Stop / Start Container

```
$ docker stop herp_derp
```

Send **SIGHUP** to container. If it does not stop within 10 seconds, send **SIGKILL**.

```
$ docker stop -t 15 herp_derp
```

-t - Wait x seconds before sending **SIGKILL**.

```
$ docker start herp_derp
```

Starts a stopped container.

Everyday Docker Usage

- Tell me everything this running container

```
$ docker inspect herp_derp
```

- Show the running processes

```
$ docker top
```

- Stats on everything running on your host

```
$ docker stats
```

Lets Destroy them ALL!

Let's Destroy them ALL

```
$ docker rm herp_derp
```

Removes specified **stopped** container. YOUR DATA GOES WITH IT.

```
$ docker container prune
87dd6d549366
d92351a3a3f7
f97bf0d5aa56
```

← remove all stopped containers

```
$ docker system prune
87dd6d549366
d92351a3a3f7
f97bf0d5aa56
```

← clean up the world!

Volumes

```
$ docker run --name mysql -v /home/jorge/mysql-data:/var/lib/mysql
mysql:latest
```

Map a local directory to where MySQL puts it’s data. Now I don’t lose data when the container goes away.

Lots o’ Volumes

```
$ docker run --name mysql -v /home/jorge/mysql-data:/var/lib/mysql -v
/home/jorge/mysql-conf:/etc/mysql/conf.d mysql:latest
```

Map a local directory to where MySQL puts it’s data.

And also make another volume for keeping MySQL config.

I have now decoupled the data and config from the container image itself.

(Still lots of things to do, but we’re getting there.)

Environment Variables

```
$ docker run --name mysql -v /home/jorge/mysql-data:/var/lib/mysql -e
MYSQL_USER=jorge -e MYSQL_PASSWORD=swordfish mysql:latest
```

Pass an environment variable to the container.

Credentials and other dynamic data NEVER, EVER, gets baked into the container.

Building declarative Images with Dockerfile's

```
FROM alpine:3.7

RUN apk --update --no-cache add \
      unrar \
    && rm -rf /var/cache/apk/*

COPY echo-server /echo-server
COPY httpstat-bin /bin/httpstat
COPY run /
RUN chmod +x /run /bin/httpstat /echo-server/echo-server
WORKDIR /echo-server
ENV PORT 80
ENV SSLPORT 443
ENTRYPOINT ["/run"]
CMD ["/echo-server/echo-server"]
```

CMD vs ENTRYPOINT

Entrypoint - The executable to launch, taking further configuration from your Command. `/bin/sh -c` by default (PID 1)

Command - Parameters to configure the entrypoint or simpler way to launch executable.

Differences and Gotchas: https://www.ctl.io/developers/blog/post/dockerfile-entrypoint-vs-cmd/

Building an Image

```
$ docker build -t engage ./Dockerfile
...
Step 6/7 : EXPOSE 80
 ---> Running in f2e73d6a7948
Removing intermediate container f2e73d6a7948
 ---> 61d57c406652
Step 7/7 : CMD ["gunicorn", "-b", "0.0.0.0:80", "httpbin:app", "-k", "gevent"]
 ---> Running in 31d5dcf6809a
Removing intermediate container 31d5dcf6809a
 ---> c335ff31682f
Successfully built c335ff31682f
Successfully tagged my_httpbin:latest
```

A quick sidebar

On building images if you come from VMs

“Cheap” Operations

- Stopping/killing/deleting containers
- Rebuilding images
- Publishing images
- Keeping old images around

This world is ephemeral and dynamically repeatable.

Stats from Datadog

- Median company is running about **7 containers per host.**
- Containers churn **9x faster than VMs.**
    - 2.5 day lifespan when orchestrated
    - 5.5 day lifespan when not
    - 23 day lifespan for VMs

https://www.datadoghq.com/docker-adoption/

Tagging and Pushing

```
$ docker tag engage reg.example.com/engage:1.2
$ docker images reg.example.com/engage:1.2
REPOSITORY                                      TAG
IMAGE ID                CREATED                   SIZE
reg.example.com/engage   1.2                          b175e7467d66
5 weeks ago             109MB
$ docker push registry.lolcakes.com/engage
```

Enough networking to make you dangerous

Docker provides a Bridge interface with NAT.

- Run the container on a specific port and ip.

```
$ docker run -d -p 3306:3306 mysql
```

- Just run everything on the host’s network namespace:

```
$ docker run -d --net=host mysql
```

Docker Compose

Docker-compose gives you a declarative method for well defined container runtime configuration and execution.

Also life is too short for all those CLI flags.

Here are a few examples of docker-compose.yml configurations

Workshop Time!

Levelling Up

For now, concentrate on your laptop / single-node workflow.

You are now ready to graduate to a container orchestration system like Mesos, Swarm, or Kubernetes.

… or just use it on single hosts or for development, that’s fine too!

ITHAKA
ARTSTOR / ITHAKA S+R / JSTOR / PORTICO

LLamasoft®
Supply Chain By Design

Questions?

CLOUD NATIVE
COMPUTING FOUNDATION

fly