

docker

Docker fundamentals

Objectives

By the end of this seminar, you will be able to:

- Explain what docker is and what it is good for
- Know the difference between containers and virtual machines
- Understand the architecture of docker containers
- Build, modify, and run docker containers

Agenda

- Virtualization Overview
- Introduction to Containers
- Introduction to Docker
- Installing Docker
- The Docker Architecture
- The Docker Engine
- Creating a Docker container
- Building Docker images
- Storing and retrieving Docker images from Docker Hub
- Building containers from images
- Deploying applications with Docker
- Docker Networking
- Continuous Integration and Deployment process using Docker



Introduction

Introduction to docker

The Challenge

Multiplicity of Stacks



Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2



Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs



User DB

postgresql + pgv8 + v8



Queue

Redis + redis-sentinel



Analytics DB

hadoop + hive + thrift + OpenJDK



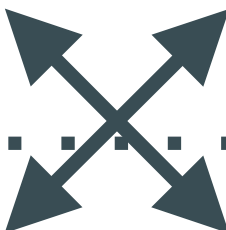
Web frontend

Ruby + Rails + sass + Unicorn



API endpoint

Python 2.7 + Flask + pyredis + celery + pycopg + postgresql-client



Do services and apps interact appropriately?

Multiplicity of hardware environments



Development VM

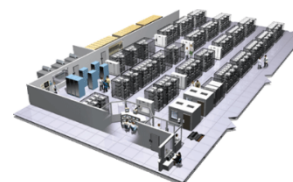


QA server

Customer Data Center



Public Cloud



Production Cluster



Disaster recovery

Production Servers







Contributor's laptop



Can I migrate smoothly and quickly?



The Matrix From Hell



Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



Cargo Transport Pre-1960

Multiplicity of Goods

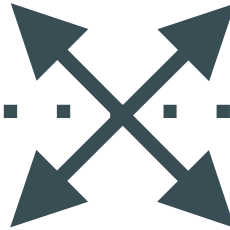


Do I worry about how goods interact (e.g. coffee beans next to spices)








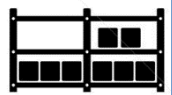





Multiplicity of methods for transporting/storing



Can I transport quickly and smoothly (e.g. from boat to train to truck)



Also a matrix from hell

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Multiplicity of Goods



A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

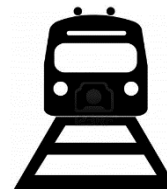
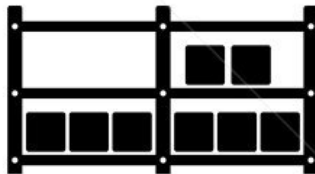


...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

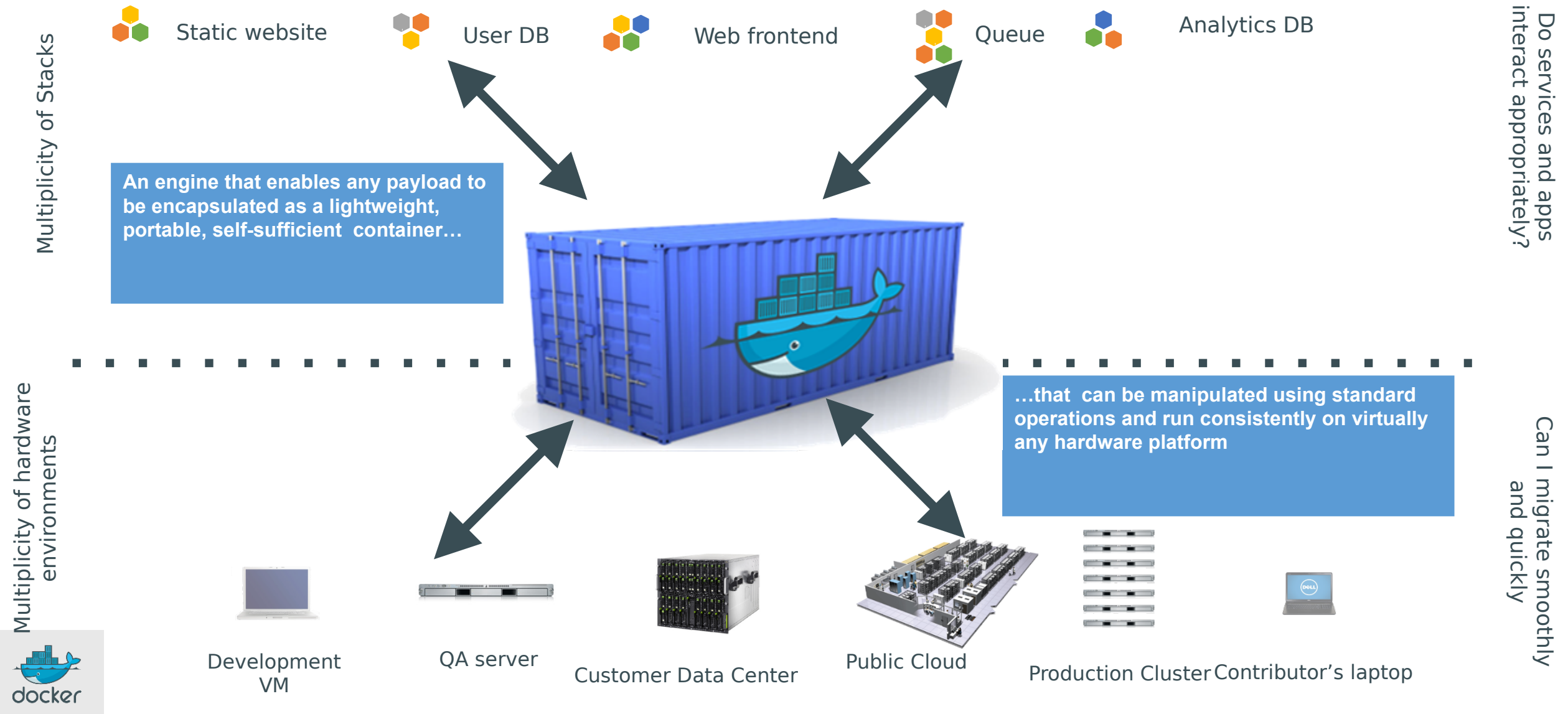
Do I worry about how goods interact (e.g. coffee beans next to spices)

Can I transport quickly and smoothly (e.g. from boat to train to truck)

















































Multiplicity of methods for transporting/storing

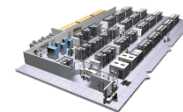


Docker is a shipping container system for code



Docker eliminates the matrix from Hell

	Static website							
	Web frontend							
	Background workers							
	User DB							
	Analytics DB							
	Queue							
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



Why Developers Care

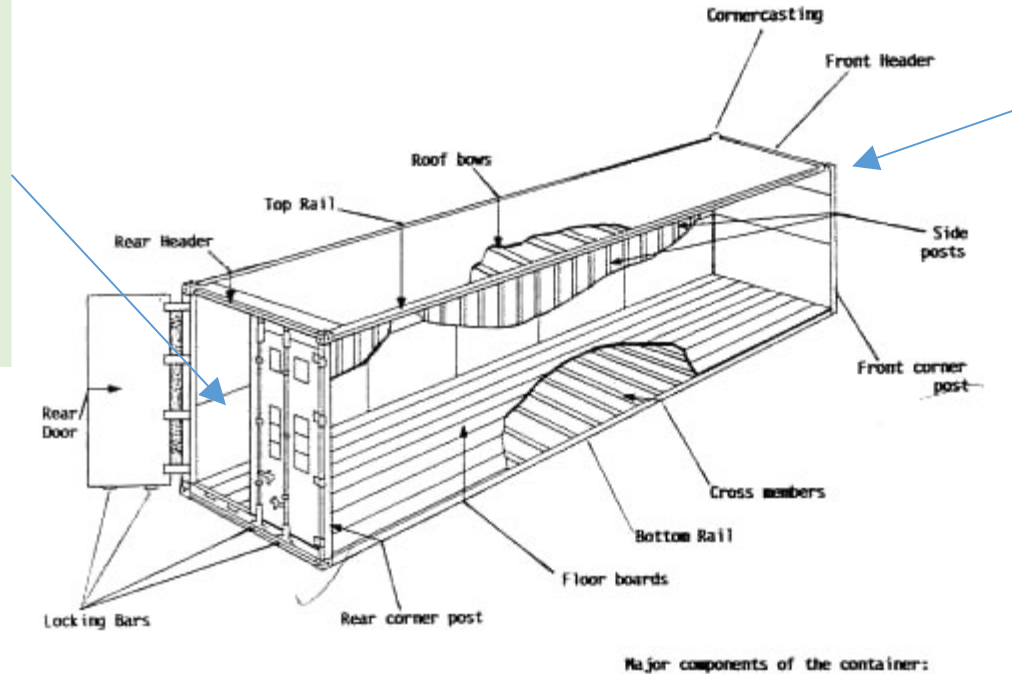
- Build once...(finally) run anywhere*
 - A clean, safe, hygienic and portable runtime environment for your app.
 - No worries about missing dependencies, packages and other pain points during subsequent deployments.
 - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
 - Automate testing, integration, packaging...anything you can script
 - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
 - Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That's the power of Docker

Why Devops Cares?

- Configure once...run anything
 - Make the entire lifecycle more efficient, consistent, and repeatable
 - Increase the quality of code produced by developers.
 - Eliminate inconsistencies between development, test, production, and customer environments
 - Support segregation of duties
 - Significantly improves the speed and reliability of continuous deployment and continuous integration systems
 - Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs

Why it works—separation of concerns

- Dan the Developer
 - Worries about what's "inside" the container
 - His code
 - His Libraries
 - His Package Manager
 - His Apps
 - His Data
 - All Linux servers look the same



- Oscar the Ops Guy
 - Worries about what's "outside" the container
 - Logging
 - Remote access
 - Monitoring
 - Network config
 - All containers start, stop, copy, attach, migrate, etc. the same way

More technical explanation

WHY

- Run everywhere
 - Regardless of kernel version (2.6.32+)
 - Regardless of host distro
 - Physical or virtual, cloud or not
 - Container and host architecture must match*
- Run anything
 - If it can run on the host, it can run in the container
 - i.e. if it can run on a Linux kernel, it can run

WHAT

- High Level—It's a lightweight VM
 - Own process space
 - Own network interface
 - Can run stuff as root
 - Can have its own /sbin/init (different from host)
 - <<machine container>>
- Low Level—It's chroot on steroids
 - Can also *not* have its own /sbin/init
 - Container=isolated processes
 - Share kernel with host
 - No device emulation (neither HVM nor PV) from host)
 - <<application container>>

What is Docker, really?

- An Open-Source (Go) framework to manage “container virtualization”
- Docker isolates multiple user spaces (file systems) inside the same host
- The user space instances are called “Containers”
- They give you the illusion of being inside a different machine
- Think about “execution environments” or “sandboxes”
- No need for an hypervisor (and so very quick to launch)
- Requires x64 Linux and kernel 3.8+

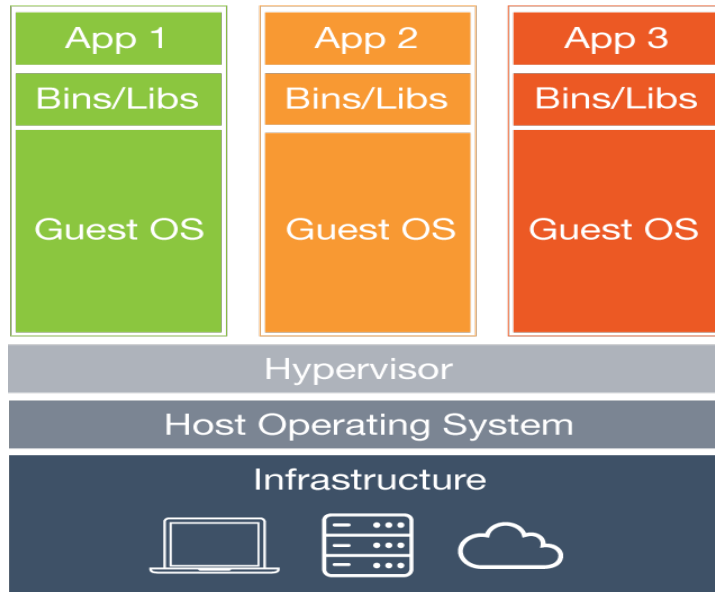


Containers vs virtual machines

- Containers have similar resource isolation and allocation benefits as virtual machines
- A different architectural approach allows them to be much more portable and efficient.

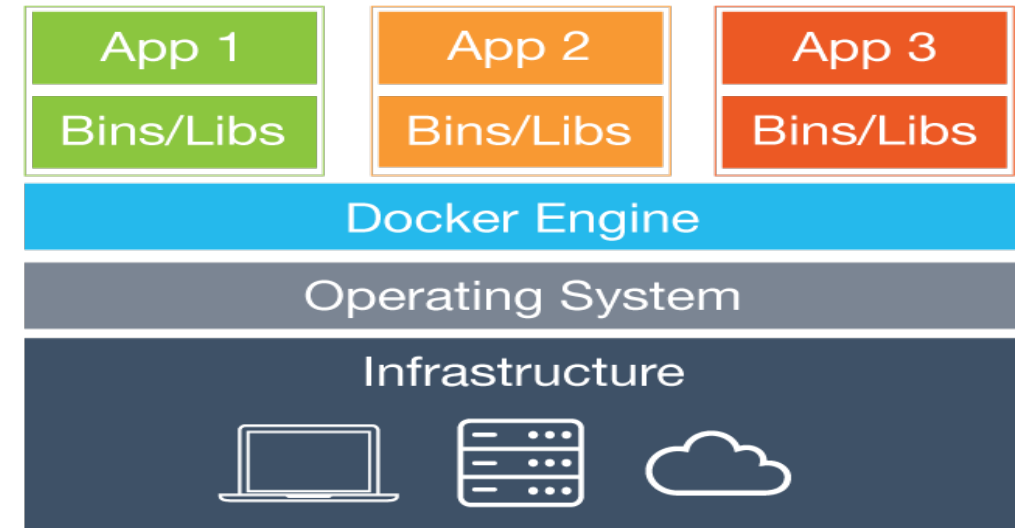
Virtual Machines

Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size.



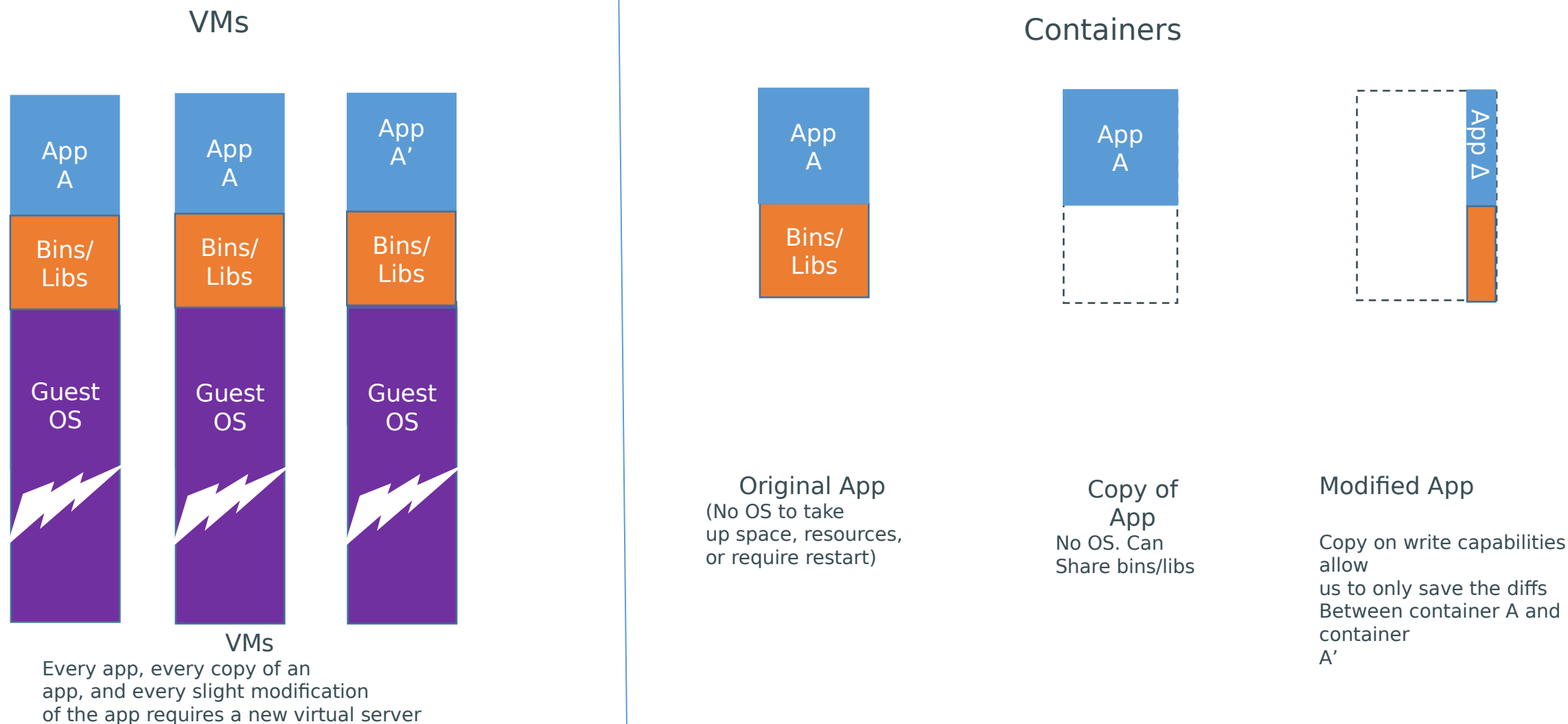
Containers

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure – Docker containers run on any computer, on any infrastructure and in any cloud.

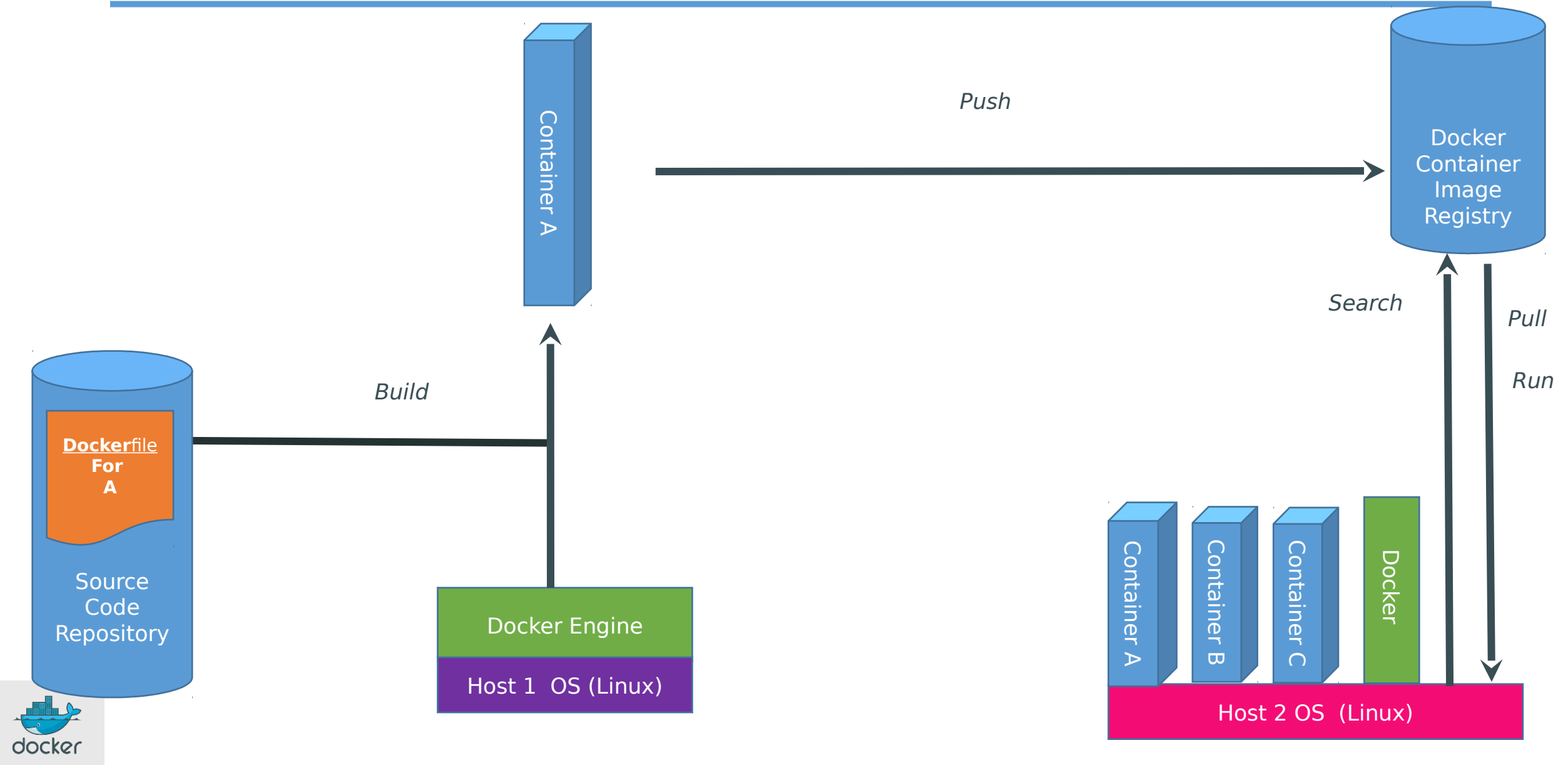


Source: <https://www.docker.com/what-docker>

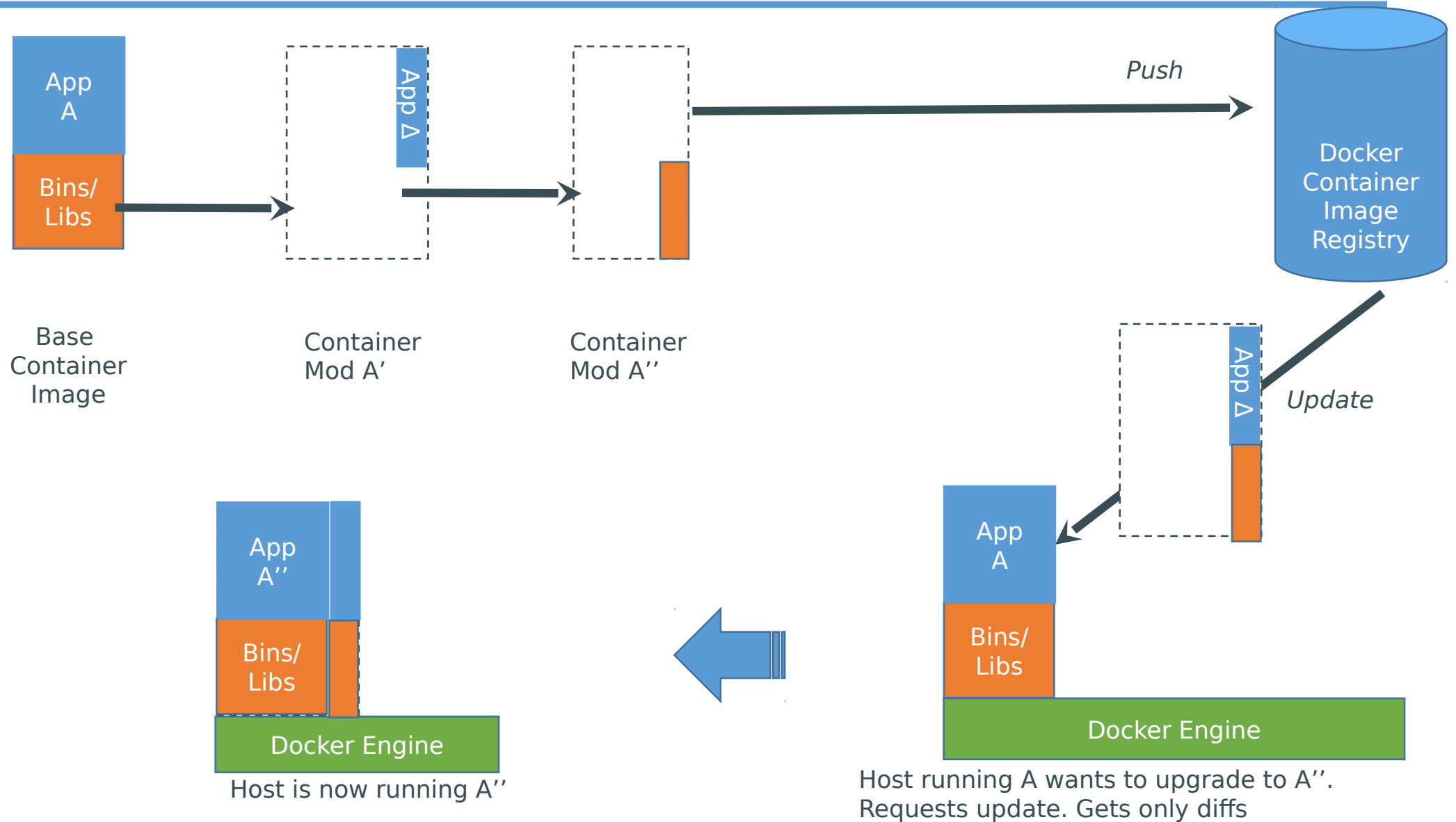
Why are Docker containers lightweight?



What are the basics of the Docker system?



Changes and Updates



Who uses docker?

- Uber, eBay, BBC News, shopify, ING, Groupon, Orbitz
Lyft, Yandex, Spotify, New Relic, Gilt
 - and many others...
- Used by Google Cloud (with the Container Engine + Kubernetes)
- Amazon (Amazon ECS)

Source: <https://www.docker.com/customers>



Who can benefit from Docker?

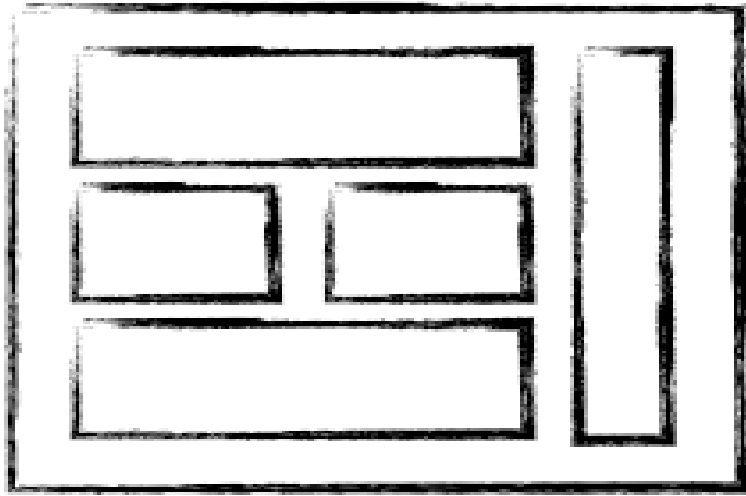
- Developers
 - Yes, even mobile developers
- Sysadmins
- DevOps
- Architects/CTO/COO (want to save some money?)
- You (probably)



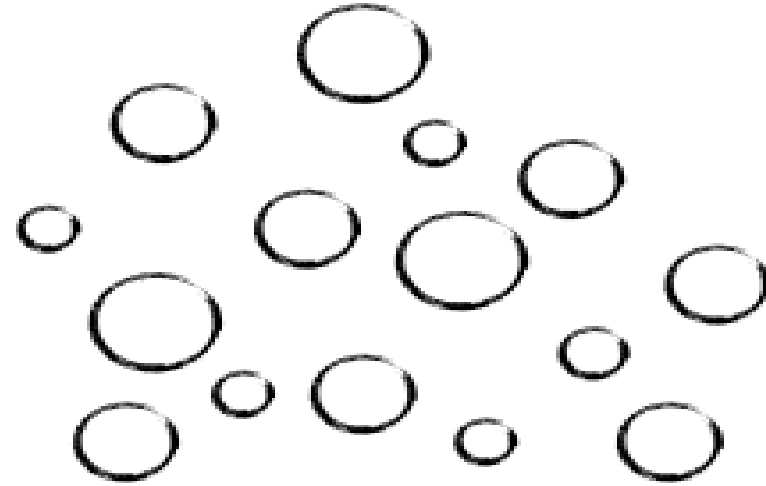
How does it help?

Creates a standard way to ship build artifacts
Ships the exact binary the developer had





MONOLITHIC/LAYERED



MICRO SERVICES

Micro-service architecture benefits

- Eliminates dependencies
- Failure is isolated
- React to change quicker
- Scale is less expensive (APIs scale individually)
- More intuitive learning curve
- Technology stack is not limited to specific skillsets
- Reusable components
- Flexible - will bend rather than break under pressure

Ecosystem Support

- Operating systems
 - Virtually any distribution with a 2.6.32+ kernel
 - Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
 - CoreOS—Small core OS purpose built with Docker
- OpenStack
 - Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana
- Private PaaS
 - OpenShift
 - Solum (Rackspace, OpenStack)
 - Other TBA
- Public PaaS
 - Deis, Voxoz, Cocaine (Yandex), Baidu PaaS
- Public IaaS
 - Native support in Rackspace, Digital Ocean, +++
 - AMI (or equivalent) available for AWS & other
- DevOps Tools
 - Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible +++
- Orchestration tools
 - Mesos, Heat, ++
 - Shipyard & others purpose built for Docker
- Applications
 - 1000's of Dockerized applications available at index.docker.io



Use Cases

- Ted Dziuba on the Use of Docker for Continuous Integration at Ebay Now
 - <https://speakerdeck.com/teddziuba/docker-at-ebay>
 - http://www.youtube.com/watch?feature=player_embedded&v=0Hi0W4gX--4
- Sasha Klizhentas on use of Docker at Mailgun/Rackspace
 - http://www.youtube.com/watch?feature=player_embedded&v=CMC3xdAo9RI
- Sebastien Pahl on use of Docker at CloudFlare
 - http://www.youtube.com/watch?feature=player_embedded&v=-Lj3jt_-3r0
- Cambridge HealthCare
 - <http://blog.howareyou.com/post/62157486858/continuous-delivery-with-docker-and-jenkins-part-i>
- Red Hat Openshift and Docker
 - <https://www.openshift.com/blogs/technical-thoughts-on-openshift-and-docker>



Use Cases—From Our Community

Use Case	Examples	Link
Clusters	Building a MongoDB cluster using docker	http://bit.ly/1acbjZf
	Production Quality MongoDB Setup with Docker	http://bit.ly/15CaiHb
	Wildfly cluster using Docker on Fedora	http://bit.ly/1bCIX0O
Build your own PaaS	OpenSource PaaS built on Docker, Chef, and Heroku Buildpacks	http://deis.io
Web Based Environment for Instruction	JiffyLab – web based environment for the instruction, or lightweight use of, Python and UNIX shell	http://bit.ly/12oaj2K
Easy Application Deployment	Deploy Java Apps With Docker = Awesome	http://bit.ly/11BCvvu
	How to put your development environment on docker	http://bit.ly/1b4XtJ3
	Running Drupal on Docker	http://bit.ly/15MJS6B
	Installing Redis on Docker	http://bit.ly/16EWOKh
Create Secure Sandboxes	Docker makes creating secure sandboxes easier than ever	http://bit.ly/13mZGJH
Create your own SaaS	Memcached as a Service	http://bit.ly/11nL8vh
Automated Application Deployment	Multi-cloud Deployment with Docker	http://bit.ly/1bF3CN6
Continuous Integration and Deployment	Next Generation Continuous Integration & Deployment with dotCloud's Docker and Strider	http://bit.ly/ZwTfoy
	Testing Salt States Rapidly With Docker	http://bit.ly/1eFBtcm
Lightweight Desktop Virtualization	Docker Desktop: Your Desktop Over SSH Running Inside Of A Docker Container	http://bit.ly/14RYL6x



Lets get started!

Docker Terminology

- **Docker**, aka Docker Engine: the daemon managing docker images and containers (using namespaces and cgroups). It runs on the (Linux-based) Host.
- **Docker client**: the binary interacting with the Docker Engine.
- **Docker Image**: a filesystem (read-only template) used to create a Container (think “the binary”)
- **Docker Container**: a running image providing a service (think “the process”)
- **Host**: the computer running the Docker Engine
- **Docker Registry**: a private or public (Docker Hub) collection of Docker Images
- **Docker Machine**: provision hosts and install Docker on them
- **Docker Compose**: create and manage multi-container architectures



Obtaining and installing Docker

First we need to install the docker daemon and client binaries

- Run “curl -fsSL https://get.docker.com/ | sh”

This script will ask you for your sudo password and install all the required binaries

Now lets verify that our installation is complete

- Run “docker run hello-world”
- If the above command fails, you may need to add your user to the docker group before docker will run
- “sudo usermod -a -G docker YOURUSERNAME”

For complete instructions and other Operating systems:

<https://docs.docker.com/linux/>



If everything is worked properly you should see the following message:

Hello from Docker.

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker Engine CLI client contacted the Docker Engine daemon.
2. The Docker Engine daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker Engine daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker Engine daemon streamed that output to the Docker Engine CLI client, which sent it to your terminal.



Docker Architecture

Docker utilizes several feature of the Linux kernel to function

- Cgroups (What you can do)
- Namespaces (What you have)
- Union file systems

Cgroups

- cgroups is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
- This allows us for manage the following per container
- Memory
- Cpu time
- Network bandwidth
- Disk bandwidth

Cgroup features

- Resource limitation: groups can be set to not exceed a configured memory limit, which also includes the file system cache
- Prioritization: some groups may get a larger share of CPU utilization[8] or disk I/O throughput
- Accounting: measures how much resources certain systems use, which may be used, for example, for billing purposes
- Control: freezing the groups of processes, their checkpointing and restarting

Namespaces

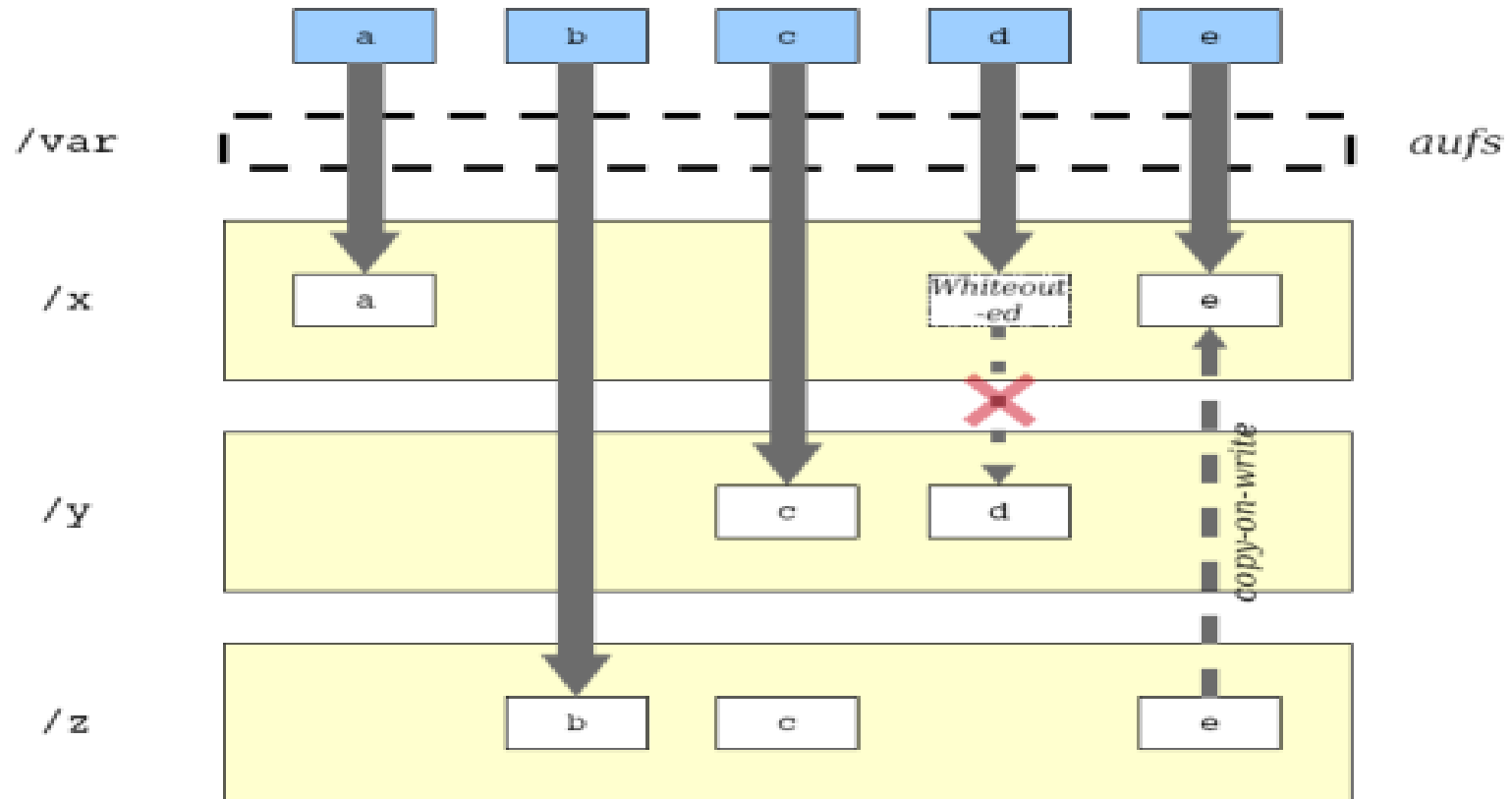
This provides a layer of isolation: each aspect of a container runs in its own namespace and does not have access outside it

- The pid namespace: Used for process isolation (PID: Process ID).
- The net namespace: Used for managing network interfaces (NET: Networking).
- The ipc namespace: Used for managing access to IPC resources (IPC: InterProcess Communication).
- The mnt namespace: Used for managing mount-points (MNT: Mount).
- The uts namespace: Used for isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Union File systems

- Union file systems, or UnionFS, are file systems that operate by creating layers
- Docker uses union file systems to provide the building blocks for containers
- Docker can make use of several union file system variants including: AUFS, btrfs, vfs, and DeviceMapper.
- The most popular is AUFS

AUFS layers



Connecting everything

Docker is the combination of these components

- LXC → LinuX Container = namespaces + cgroups
- AUFS

Lets get our hands dirty!

Exercise 1 - Our first docker container

- Download from the registry
- Execute command inside container Interactive with container

```
$ docker search ubuntu # from lots of release
```

```
$ docker pull ubuntu:latest # may take minutes
```

```
Pulling repository ubuntu
```

```
ad892dd21d60: Pulling dependent layers
```

```
511136ea3c5a: Pulling fs layer
```

```
$ docker images # list local images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntu	latest	ad892dd21d60	6 days ago	275.5 MB

```
$ docker run ubuntu echo "Hello World"
```

```
$ docker run -i -t ubuntu bash
```



docker - the command line tool

Some common commands:

\$ docker search # search hub.docker.com for an image

\$ docker pull # download image

\$ docker images # list all existing local images

\$ docker run # initiates a container from an image

\$ docker ps # list running containers

\$ docker build # build images from Dockerfile

\$ docker start/stop/kill # commands

\$ docker rm/rmi to remove a container or image



Exercise 2 - Add own package and image

- Try to install apache2 inside

- `$ docker run -i -t ubuntu /bin/bash`
 - `# apt-get update && apt-get install -y apache2`
 - `# exit`

- `$ docker ps -l # -l means -latest`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c4bd63cc87f1	ubuntu:latest	bash	2 minutes ago	Exited 2 sec

- `$ docker commit <container id> apache2`

66db661d9ad8681b082bb62b21b6ef5f2ddb4799e3df5dbd8fb23aed16616b1d

- Check and run it again to see the apache is there

- `$ docker images`

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
apache2	latest	66db661d9ad8	28 seconds ago	298.5 MB
ubuntu	latest	ad892dd21d60	6 days ago	275.5 MB

- `$ docker run -i -t apache2 bash # check apache`



How does this work?

Docker images are saved in layers

```
$ docker images --tree
```

Warning: '--tree' is deprecated, it will be removed soon. See usage.

```
└─511136ea3c5a Virtual Size: 0 B
  └─e465fff03bce Virtual Size: 192.5 MB
    └─23f361102fae Virtual Size: 192.7 MB
      └─9db365ecbcb Virtual Size: 192.7 MB
        └─ad892dd21d60 Virtual Size: 275.5 MB Tags: ubuntu:latest
          └─66db661d9ad8 Virtual Size: 298.5 MB Tags: apache2:latest
```

-
- When Docker mounts the rootfs, it starts read-only
 - Uses aufs) to add a read-write file system *over* the read-only file system
 - There are multiple read-only file systems stacked on top of each other
 - each one of these file systems is a **layer**.

Exercise 3 Understanding the layers

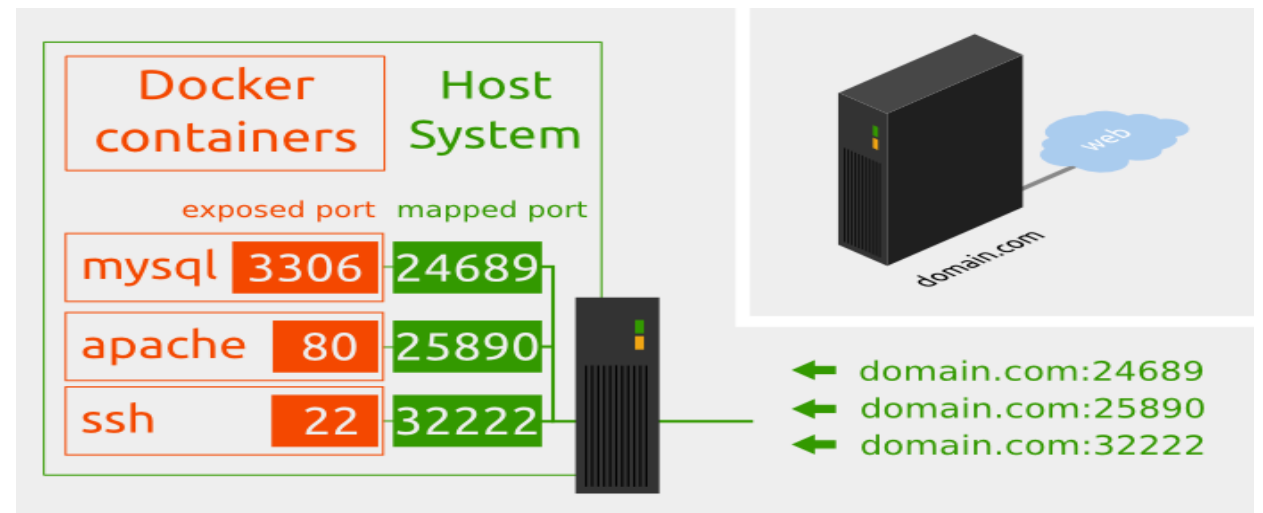
- Data are stored under `/var/lib/docker` (root permission)
- ```
$ ls -al /var/lib/docker
$ ls -l /var/lib/docker/aufs/diff/

..
66db661d9ad8681b082bb62b21b6ef5f2ddb4799e3df5dbd8fb23aed16616b1d/
9db365ecbcbbb20e063eac70842c53e27fcad0e213f9d4ddb78152339cedd3b1/
```
- See what is diff inside `/var/lib/docker/aufs/diff/<image id>`
- ```
$ find /var/lib/docker/aufs/diff/66* | grep apache2
```

Networking in docker

- By default the network in the container is not visible outside – everything is NATed
- The services can be exposed by Port
- Run `-p host:guest` # assign port to host

- `$ docker run -p 25890:80 -i -t apache2 bash`
- `# apache2ctl start`



Docker Daemon mode

- Interactive mode vs. Daemon (Deattach) mode

(`docker run`)

`-d` : run in daemon mode

`-i` : run in interactive mode

(CTRL-P-Q to quit from shell to enter daemon mode, exit will stop the container)

`$ docker attach <container ID>` : connect to existing container

- Other parameters for `docker run`

`--name` : define container name

- Enter into existing docker container (since 1.3 version)

`$ docker exec -it <container ID> bash`



Exercise 4: Expose the service

- Export the port to host as 25890 and start the service manually

- `$ docker run -p 25890:80 -i -t apache2 bash`
- `root@35ac981a49e5:/# service apache2 start`
- `$ docker ps # in another shell`

CONTAINER ID	IMAGE	...	STATUS	PORTS	NAMES
e020aa2c02a5	apache2:latest	Up 14 seconds	0.0.0.0:25890->80/tcp	web

- `$ curl http://localhost:25890`

- Come into the container again to check

- `$ docker exec -it e020aa2c02a5 bash`

- Run contain in daemon mode

- `$ docker run -p 25891:80 -d -t apache2 apache2ctl -D FOREGROUND`

Dockerfile

- Dockerfiles = image representations
- Simple syntax for building images
- Automate and script the images creation

Dockerfile

- Dockerfile Syntax
 - FROM – defines base image
 - RUN – executes arbitrary command
 - ENV – sets environment
 - EXPOSE – expose a port
 - ADD – add local file
 - CMD – default command to execute
 - MAINTAINER – author information
- Used by `docker build`



- TIP: find images with the command: `docker search`



RUN

- Executes any commands on the current image and commit the results
- Usage: RUN <command>
- Example: RUN apt-get install -y memcached

FROM ubuntu

RUN apt-get install -y memcached

- Is equivalent to:

docker run ubuntu apt-get install -y memcached

docker commit XXX

docker build

- Creates an image from a Dockerfile
 - From the current directory = `docker build`
 - From stdin = `docker build - < Dockerfile`
 - From GitHub = `docker build github.com/creack/docker-firefox`

- TIP: Use `-t` to tag your image



Example: Memcached

FROM ubuntu

RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" >
/etc/apt/sources.list

RUN apt-get update

RUN apt-get install -y memcached

- Docker build -t memcached .

Commenting

```
# Memcached
```

```
#
```

```
# VERSION 1.0
```

```
# use the ubuntu base image provided by dotCloud
```

```
FROM ubuntu
```

```
# make sure the package repository is up to date
```

```
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
```

```
RUN apt-get update
```

```
# install memcached
```

```
RUN apt-get install -y memcached
```

ENTRYPOINT

- Triggers a command as soon as the container starts
- Example: ENTRYPOINT echo "Whale You Be My Container?"

```
#  
# VERSION    0.1.2
```

```
# use the ubuntu base image  
FROM ubuntu
```

```
# say hello when the container is launched  
ENTRYPOINT echo "Hello ubuntu"
```

ENTRYPOINT option 2

- Run containers as executables! :)
- `cat /etc/passwd | docker run -i wc`

This is wc

#

VERSION 0.42

use the base image provided by dotCloud
FROM base

MAINTAINER Victor Coisne victor.coisne@dotcloud.com

count lines with wc
ENTRYPOINT ["wc", "-l"]

USER

- Sets the username to use when running the image
- Needs to exist in host machine
- Example: USER daemon

EXPOSE

- Sets ports to be publicly exposed when running the image
- Example: EXPOSE 11211
- Will expose internal port to identical external (on host machine) port

Exercise 5: Dockerfile apache2/wget

- Write the Dockerfile

```
$ mkdir /tmp/docker ; vi /tmp/docker/Dockerfile
$ cd /tmp/docker
$ docker build -t wget .
$ docker images --tree
```

```
FROM apache2
RUN apt-get install -y wget
```

- Start the wget image and verify

Share images in docker repository

- **Docker is also a tool for sharing.** A *repository* is a shareable collection of tagged [images](#) that together create the file systems for containers.
- Public repo. <username>/<repo_name>
- \$ docker search/pull/login/push

Exercise 6: Share your image

- Register in <https://hub.docker.com/>
 - Tag the wget image
- ```
$ docker tag wget omrisiri/wget:latest
```
- Docker push omrisiri/wget



---

# Docker best practices

# Containers should be ephemeral

---

- The container produced by the image your Dockerfile defines should be as ephemeral as possible
- By “ephemeral,” we mean that it can be stopped and destroyed and a new one built and put in place with an absolute minimum of set-up and configuration.

## Use a .dockerignore file

---

- In most cases, it's best to put each Dockerfile in an empty directory.
- Then, add to that directory only the files needed for building the Dockerfile
- To increase the build's performance, exclude files and directories by adding a .dockerignore file to that directory as well.

# Here is an example .dockerignore file:

```
/temp
//temp*
temp?
*.md
!README.md
```

| Rule        | Behavior                                                                                                                                                                                                |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| */temp*     | Exclude files and directories whose names start with temp in any immediate subdirectory of the root. For example, the plain file /somedir/temporary.txt is excluded, as is the directory /somedir/temp. |
| */*/temp*   | Exclude files and directories starting with temp from any subdirectory that is two levels below the root. For example, /somedir/subdir/temporary.txt is excluded.                                       |
| temp?       | Exclude files and directories in the root directory whose names are a one-character extension of temp. For example, /tempa and /tempb are excluded.                                                     |
| !README*.md | Lines starting with ! (exclamation mark) can be used to make exceptions to exclusions                                                                                                                   |



# Avoid installing unnecessary packages

---

- In order to reduce complexity, dependencies, file sizes, and build times, you should avoid installing extra or unnecessary packages
- For example, you don't need to include a text editor in a database image.

# Run only one process per container

---

- In almost all cases, you should only run a single process in a single container.
- Decoupling applications into multiple containers makes it much easier to scale horizontally and reuse containers
- If that service depends on another service, make use of container linking.

# Sort multi-line arguments

---

- Whenever possible, ease later changes by sorting multi-line arguments alphanumerically.

```
RUN apt-get update && apt-get install -y \
bzip \br/>cvs \br/>git \br/>mercurial \br/>subversion
```

# Build cache

---

- During the process of building an image Docker will step through the instructions in your Dockerfile executing each in the order specified.
- As each instruction is examined Docker will look for an existing image in its cache that it can reuse
- If you do not want to use the cache at all you can use the `--no-cache=true` option on the `docker build` command.
- The next slides explain a bit more about build cache



# Build cache

---

- It's important to understand when it will and will not find an image in the cache
- These are the basic rules:
  - Starting with a base image that is already in the cache, the next instruction is compared against all child images derived from that base image to see if one of them was built using the exact same instruction. If not, the cache is invalidated.
  - For the ADD and COPY instructions, the contents of the file(s) in the image are examined and a checksum is calculated for each file. The last-modified and last-accessed times of the file(s) are not considered in these checksums. During the cache lookup, the checksum is compared against the checksum in the existing images. If anything has changed in the file(s), such as the contents and metadata, then the cache is invalidated.
  - Aside from the ADD and COPY commands, cache checking will not look at the files in the container to determine a cache match. For example, when processing a RUN apt-get -y update command the files updated in the container will not be examined to determine if a cache hit exists. In that case just the command string itself will be used to find a match.

**Once the cache is invalidated, all subsequent Dockerfile commands will generate new images and the cache will not be used.**

# The Dockerfile instructions – Best practices

---

- FROM

- Whenever possible, use current Official Repositories as the basis for your image. We recommend the Debian or ubuntu image since it's very tightly controlled and kept minimal (currently under 150 mb), while still being a full distribution.

- RUN

- As always, to make your Dockerfile more readable, understandable, and maintainable, split long or complex RUN statements on multiple lines separated with backslashes.

- Apt-get

- Probably the most common use-case for RUN is an application of apt-get. The RUN apt-get command, because it installs packages, has several gotchas to look out for.
  - You should **avoid** RUN **apt-get upgrade** or **dist-upgrade**
  - If you know there's a particular package, foo, that needs to be updated, use apt-get install -y foo
  - Always combine RUN apt-get update with apt-get install in the same RUN statement
  - Using apt-get update alone in a RUN statement causes caching issues and subsequent apt-get install instructions fail → If you run apt-get update as separate statements it will be cached and you will get old and outdated version

# The Dockerfile instructions – Best practices

---

- **CMD**

- The CMD instruction should be used to run the software contained by your image, along with any arguments.
- CMD should almost always be used in the form of CMD ["executable", "param1", "param2"...]

- **EXPOSE**

- The EXPOSE instruction indicates the ports on which a container will listen for connections.
- Consequently, you should use the common, traditional port for your application

- **ENV**

- order to make new software easier to run, you can use ENV to update the PATH environment variable for the software your container installs.
- For example, ENV PATH /usr/local/nginx/bin:\$PATH will ensure that CMD ["nginx"] just works
- The ENV instruction is also useful for providing required environment variables specific to services you wish to containerize, such as Postgres's PGDATA..
- Lastly, ENV can also be used to set commonly used version numbers so that version bumps are easier to maintain, as seen in the following example:

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC
 /usr/src/postgress && ...
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

# The Dockerfile instructions – Best practices

---

- **ADD or COPY**

- Although ADD and COPY are functionally similar, COPY is preferred.
- it's more transparent than ADD
- COPY only supports the basic copying of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious.
- If you have multiple Dockerfile steps that use different files, COPY them individually, rather than all at once.
- This will ensure that each step's build cache is only invalidated on change

- **ENTRYPOINT**

- The best use for ENTRYPOINT is to set the image's main command, allowing that image to be run as though it was that command (and then use CMD as the default flags)
  - Example:
    - ENTRYPOINT ["s3cmd"]  
CMD ["--help"]
    - Now we can run : `docker run s3cmd ls s3://mybucket`



# The Dockerfile instructions – Best practices

---

- **VOLUME**

- The VOLUME instruction should be used to expose any database storage area, configuration storage or files/folders created by your docker container
- You are **strongly** encouraged to use VOLUME for any mutable and/or user-serviceable parts of your image.

- **USER**

- If a service can run without privileges, use USER to change to a non-root user
- Start by creating the user and group in the Dockerfile with something like:
  - **RUN** groupadd -r postgres && useradd -r -g postgres postgres
- You should avoid installing or using sudo since it has unpredictable TTY and signal-forwarding behavior that can cause more problems than it solves

- **WORKDIR**

- For clarity and reliability, you should always use absolute paths for your WORKDIR
- use WORKDIR instead of proliferating instructions like RUN cd ... && do-something, which are hard to read, troubleshoot, and maintain.

# Advanced topics

---

- Data
  - Today: Externally mounted volumes
    - Share volumes between containers
    - Share volume between a containers and underlying hosts
      - high-performance storage backend for your production database
      - making live development changes available to a container, etc.
    - Optional: specify memory limit for containers, CPU priority
    - Device mapper/ LVM snapshots in 0.7
  - Futures:
    - I/O limits
    - Container resource monitoring (CPU & memory usage)
    - Orchestration (linking & synchronization between containers)
    - Cluster orchestration (multi-host environment)
- Networking
  - Supported today:
    - UDP/TCP port allocation to containers
      - specify *which* public port to redirect. If you don't specify a public port, Docker will revert to allocating a random public port.
      - Docker uses IPtables/netfilter
    - IP allocation to containers
      - Docker uses virtual interfaces, network bridge,
  - Futures:
    - See Pipework (Upstream) : **Software-Defined Networking for Linux Containers** (<https://github.com/jpetazzo/pipework>)
    - Certain pipework concepts will move from upstream to part of core Docker
    - Additional capabilities come with libvirt support in 0.8-0.9 timeframe

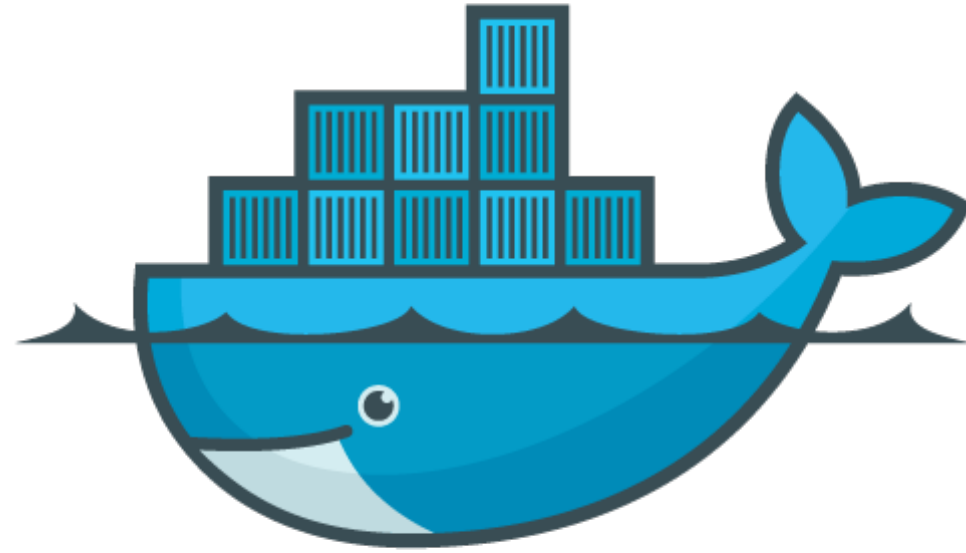


# Want to learn more?

---

- [www.docker.io](http://www.docker.io):
  - Documentation
  - Getting started: interactive tutorial, installation instructions, getting started guide,
  - About: Introductory whitepaper: <http://www.docker.io/the-whole-story/>
- Github: [dotcloud/docker](https://github.com/dotcloud/docker)
- IRC: [#docker](https://freenode.net/#docker)
- Google groups: [groups.google.com/forum/#!forum/docker-user](https://groups.google.com/forum/#!forum/docker-user)
- Twitter: follow @docker
- Meetups: Scheduled for Boston, San Francisco, Austin, London, Paris, Boulder...and Nairobi. <https://www.docker.io/meetups/>





docker  
[www.docker.io](http://www.docker.io)