Introduction to Docker

Agenda

- Motivation: Shift from Monolithic to Microservices Architectures
- The problem solved by Docker
- How Docker is different from Virtual Machines
- Docker workflow: Build, Ship and Run
- Docker commands
- Hands-on exercise

Applications have changed dramatically

Monolithic

Slowly-changing

Big (bare metal)
server

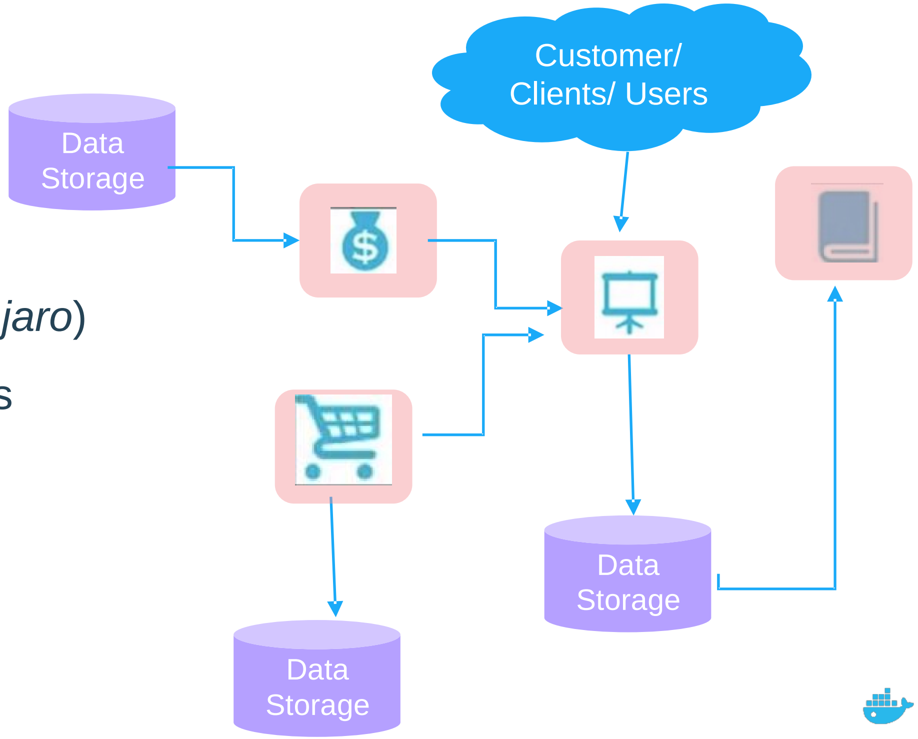
A Decade ago (and still valid)

- Apps were monolithic
- Built on a single stack such as .NET or Java
- Long Lived
- Deployed to a single server

Applications have changed dramatically

Today

- Apps are constantly developed
- Newer version are deployed often (*Manjaro*)
- Built from **loosely coupled** components
- Deployed to a multitude of servers



Customer/ Clients/ Users

Data Storage

Data Storage

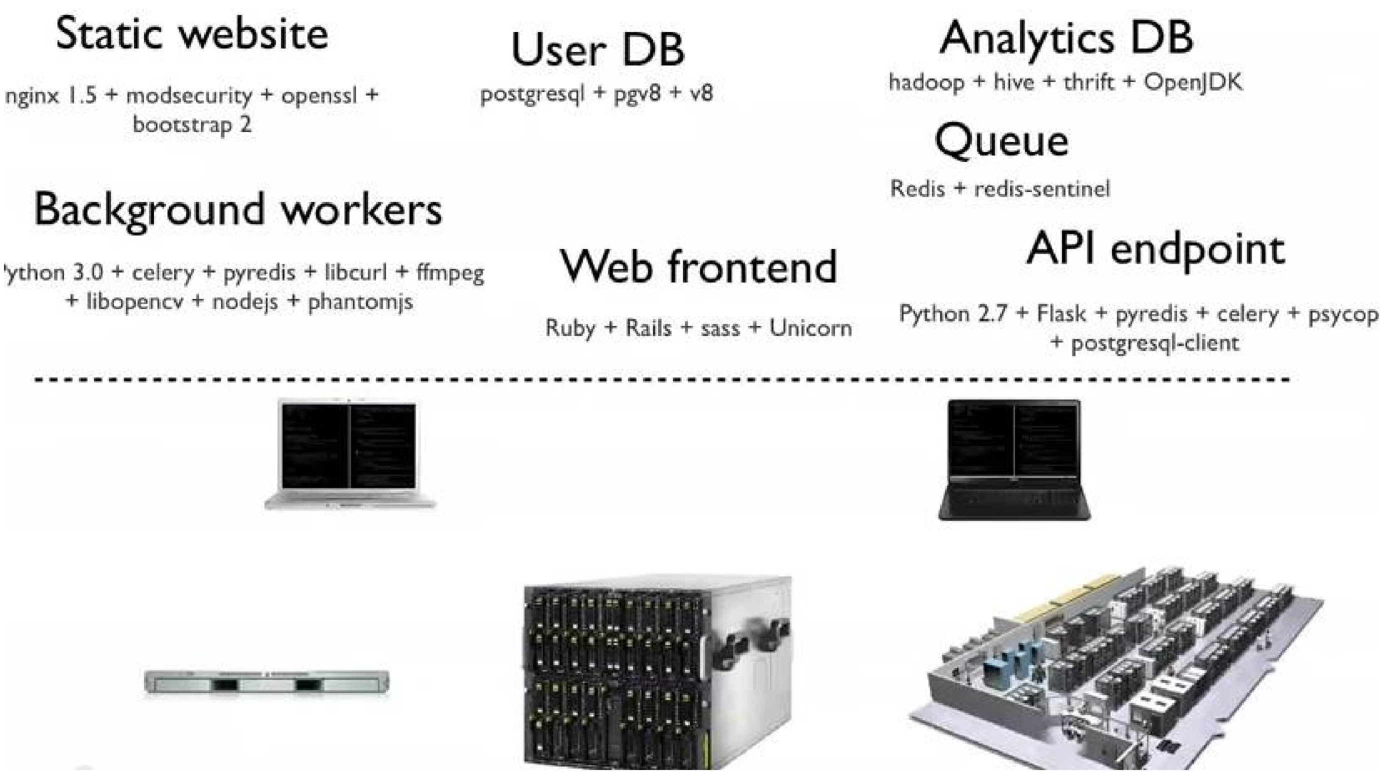
Data Storage

Once upon a time… A *software stack*

(Linux, Apache, MySQL, PHP)

LAMP

Now....much more distributed, complex...



Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Analytics DB

hadoop + hive + thrift + OpenJDK

Queue

Redis + redis-sentinel

Background workers

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

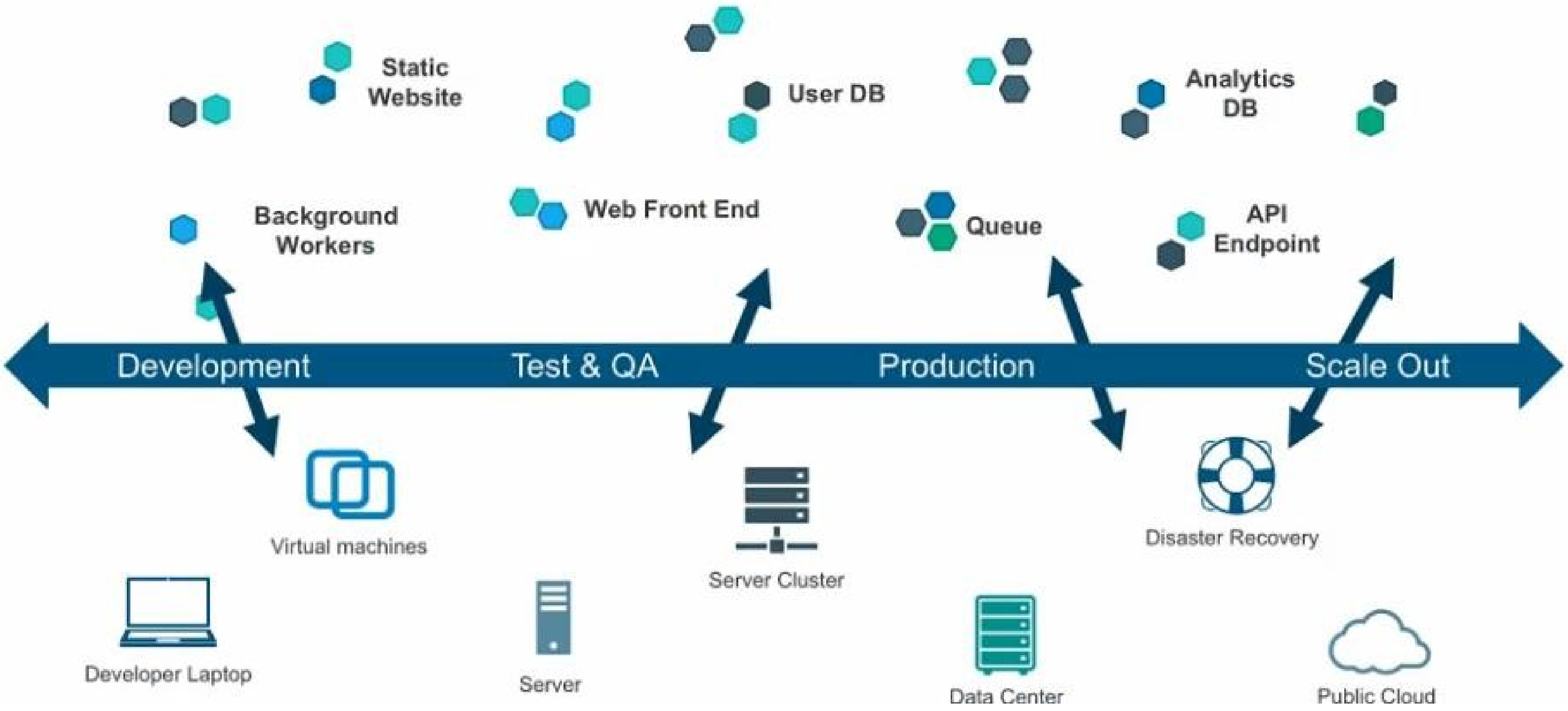
Web frontend

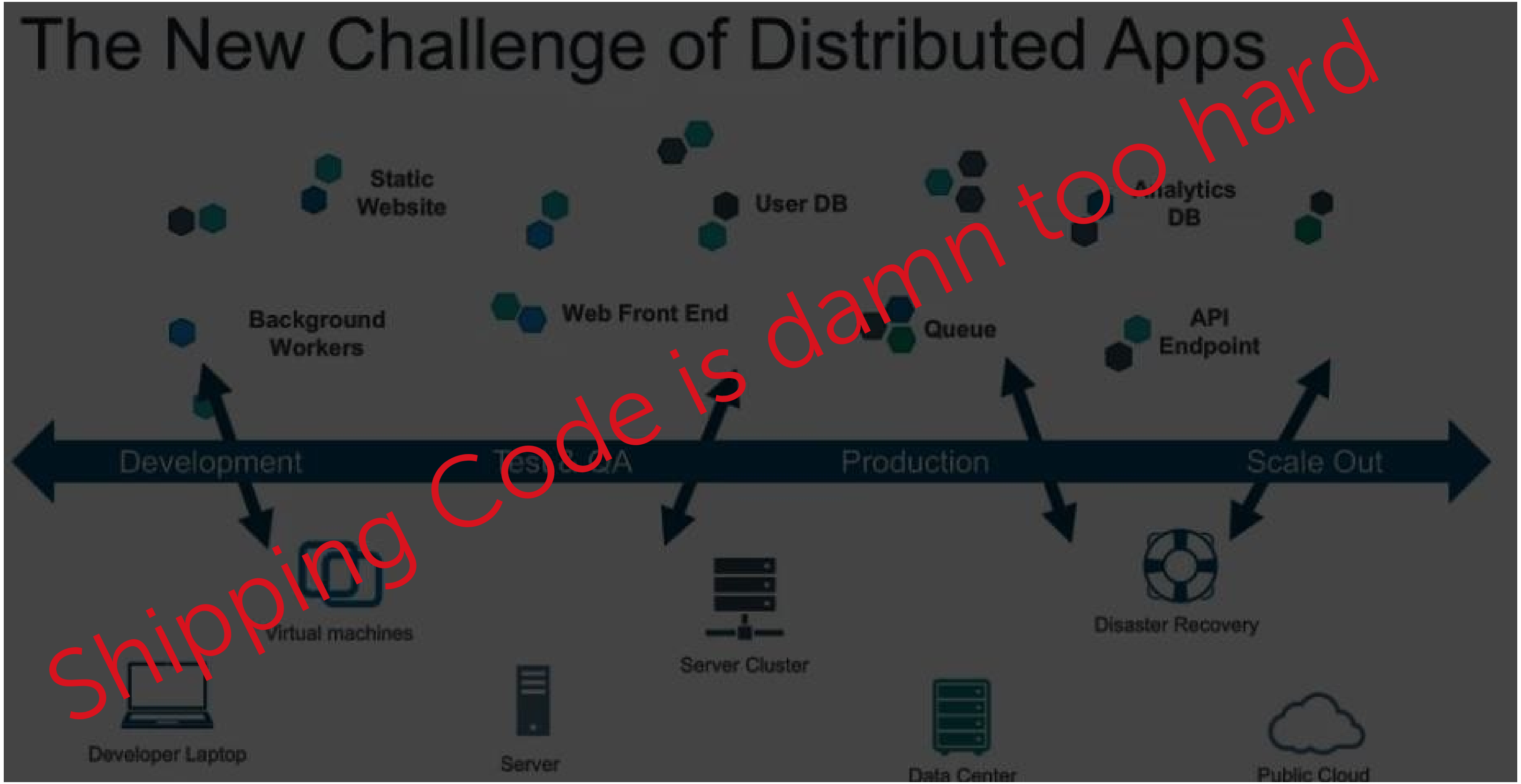
Ruby + Rails + sass + Unicorn

API endpoint

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

The New Challenge of Distributed Apps



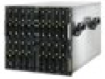
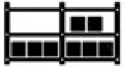
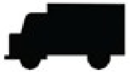


The New Challenge of Distributed Apps

Shipping Code is damn too hard

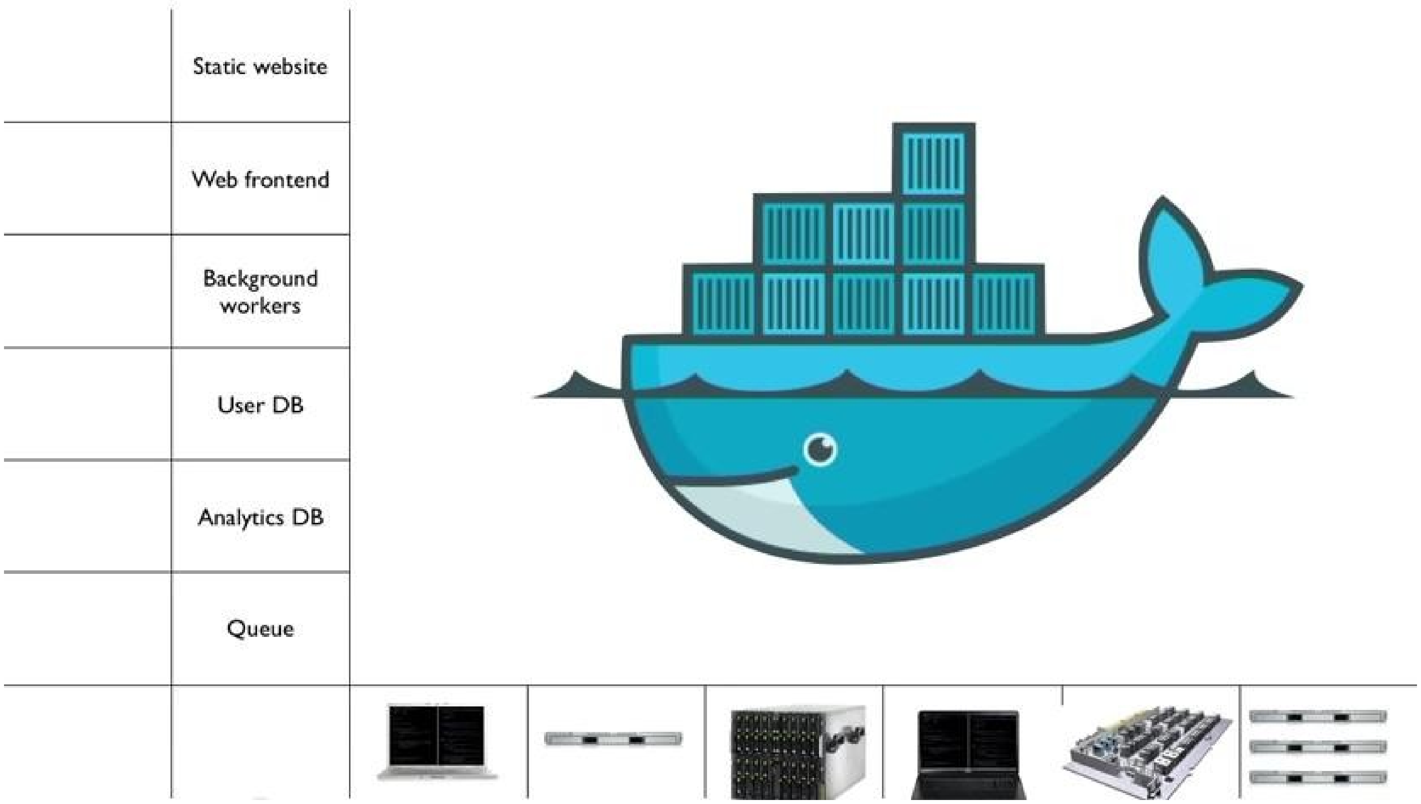
An Effort to “host” different “stacks”...

(Every possible goods) x (Every possible way to ship)

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

A Solution…

Docker ~ Brings standardization on packaging stacks



Static website

Web frontend

Background workers

User DB

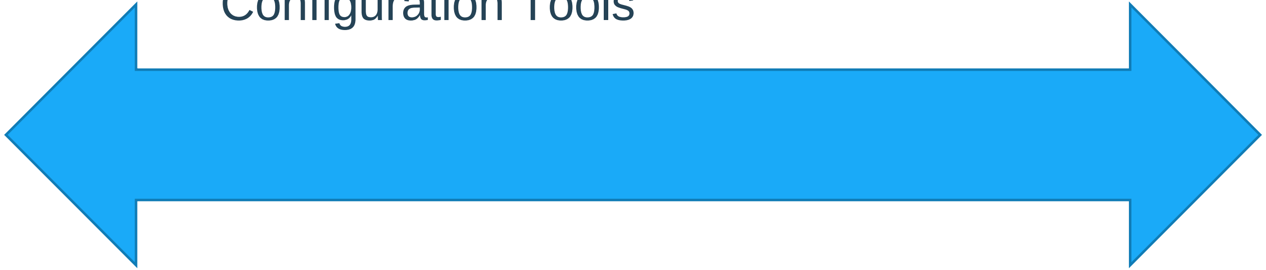
Analytics DB

Queue

Less Portable,
Minimal Overhead

More Portable,
Lots of Overhead

Configuration Tools

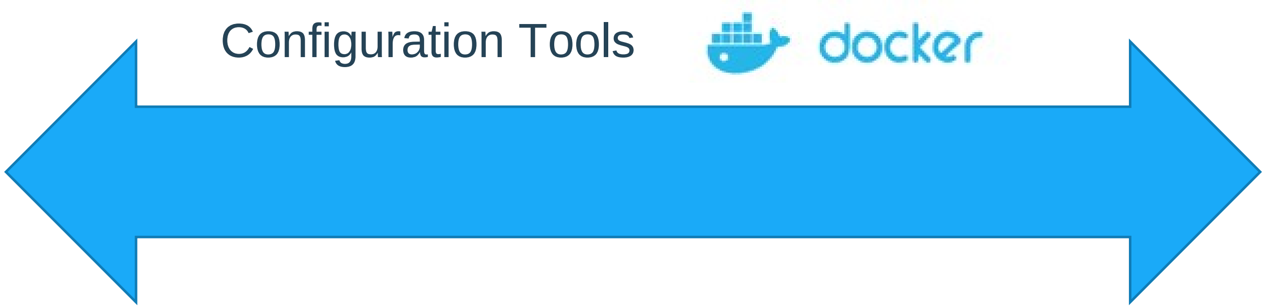


Manual Configuration

Traditional VMs

Less Portable,
Minimal Overhead

More Portable,
Lots of Overhead



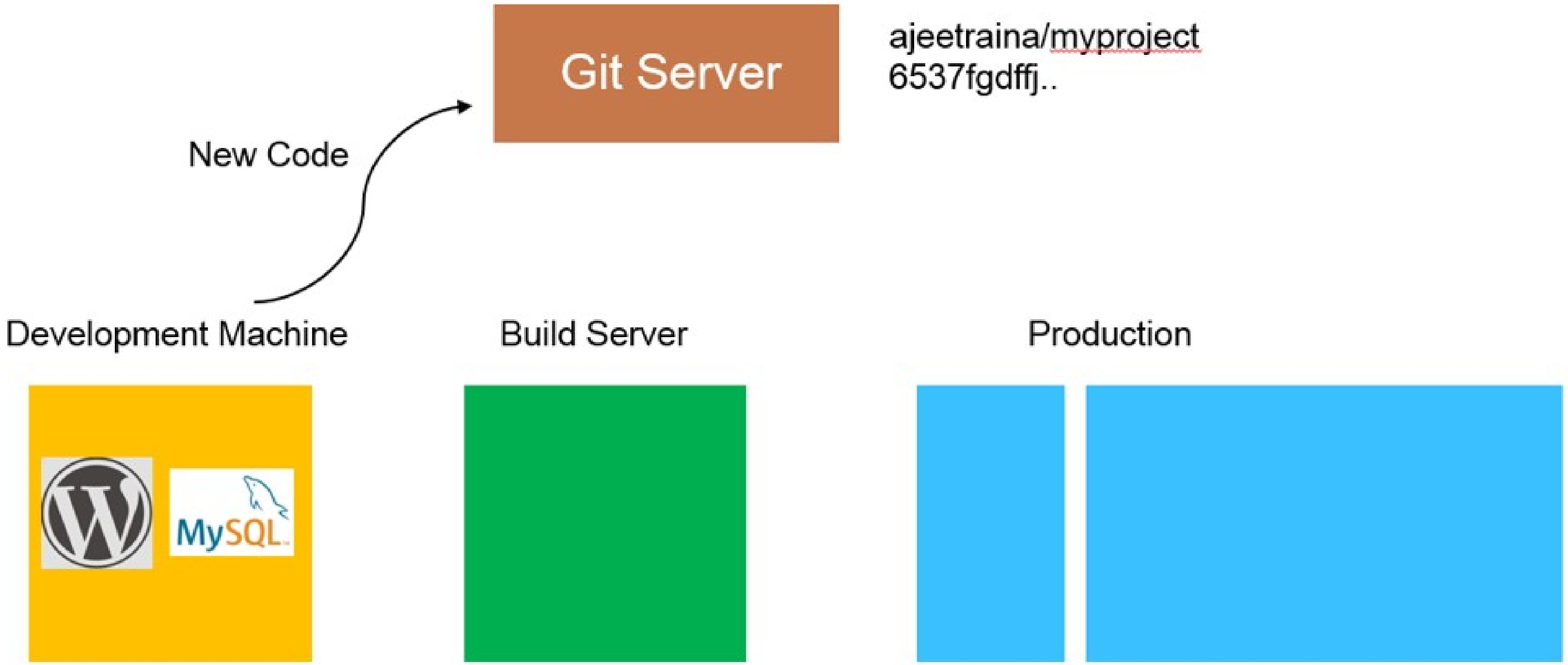
Configuration Tools

docker

Manual Configuration

Traditional VMs

Development workflow (without Docker)



Git Server

ajeetraina/myproject
6537fgdffj..

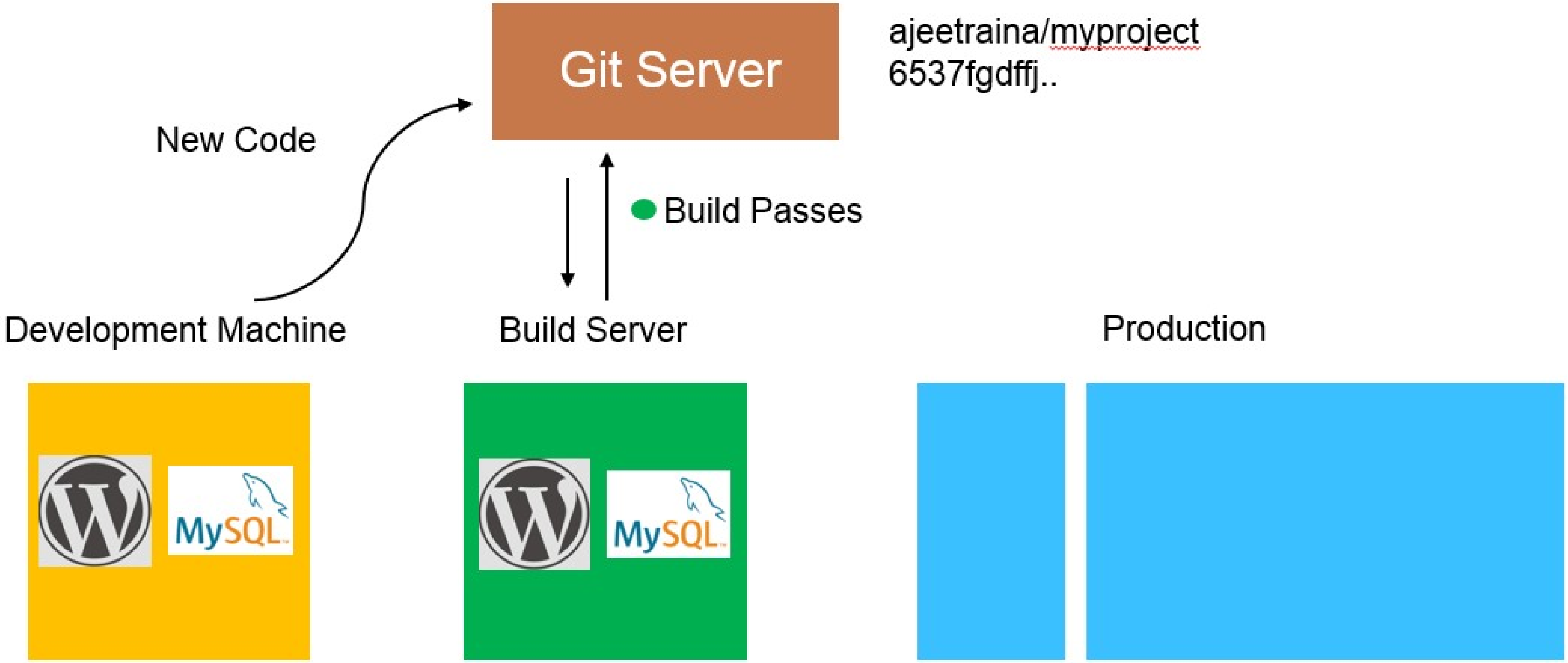
New Code

Development Machine

Build Server

Production

Development workflow (without Docker)



Git Server

ajeetraina/myproject
6537fgdffj..

New Code

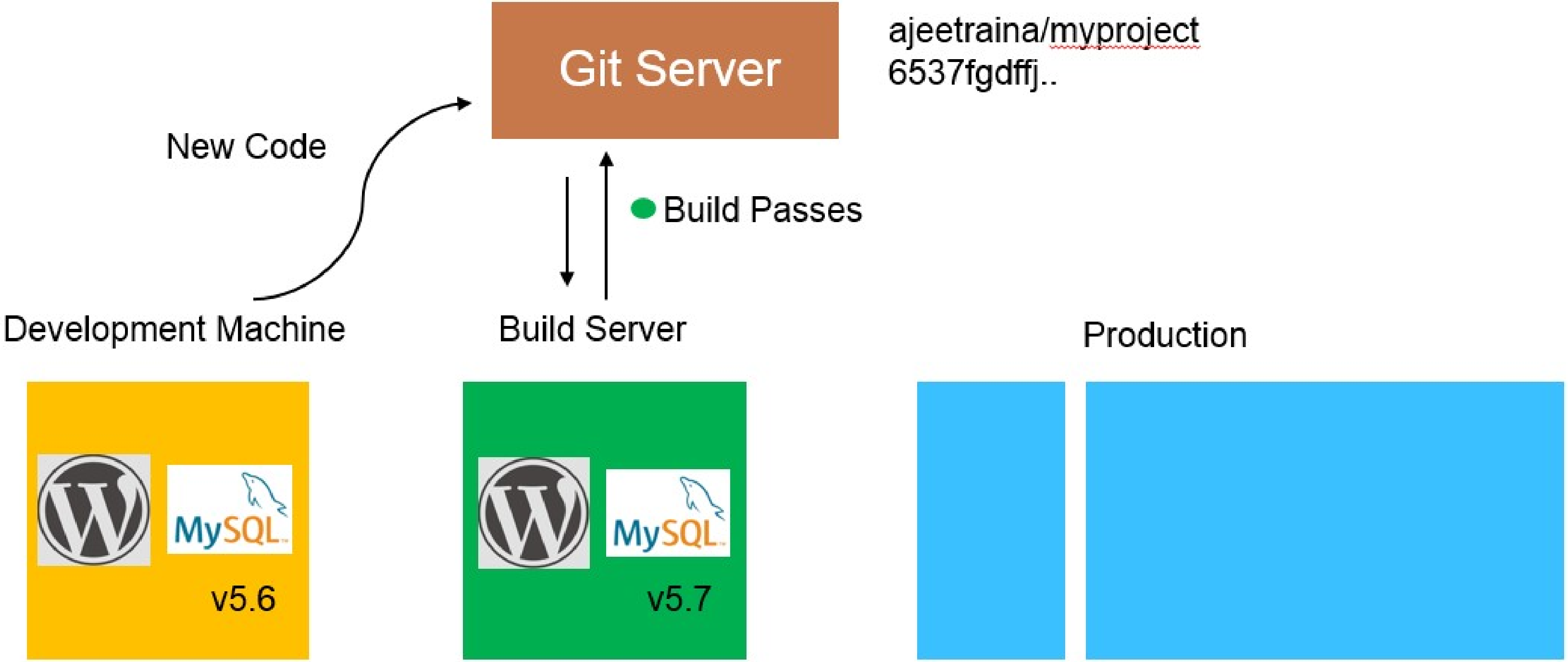
● Build Passes

Development Machine

Build Server

Production

Development workflow (without Docker)



Git Server

ajeetraina/myproject
6537fgdffj..

New Code

● Build Passes

Development Machine

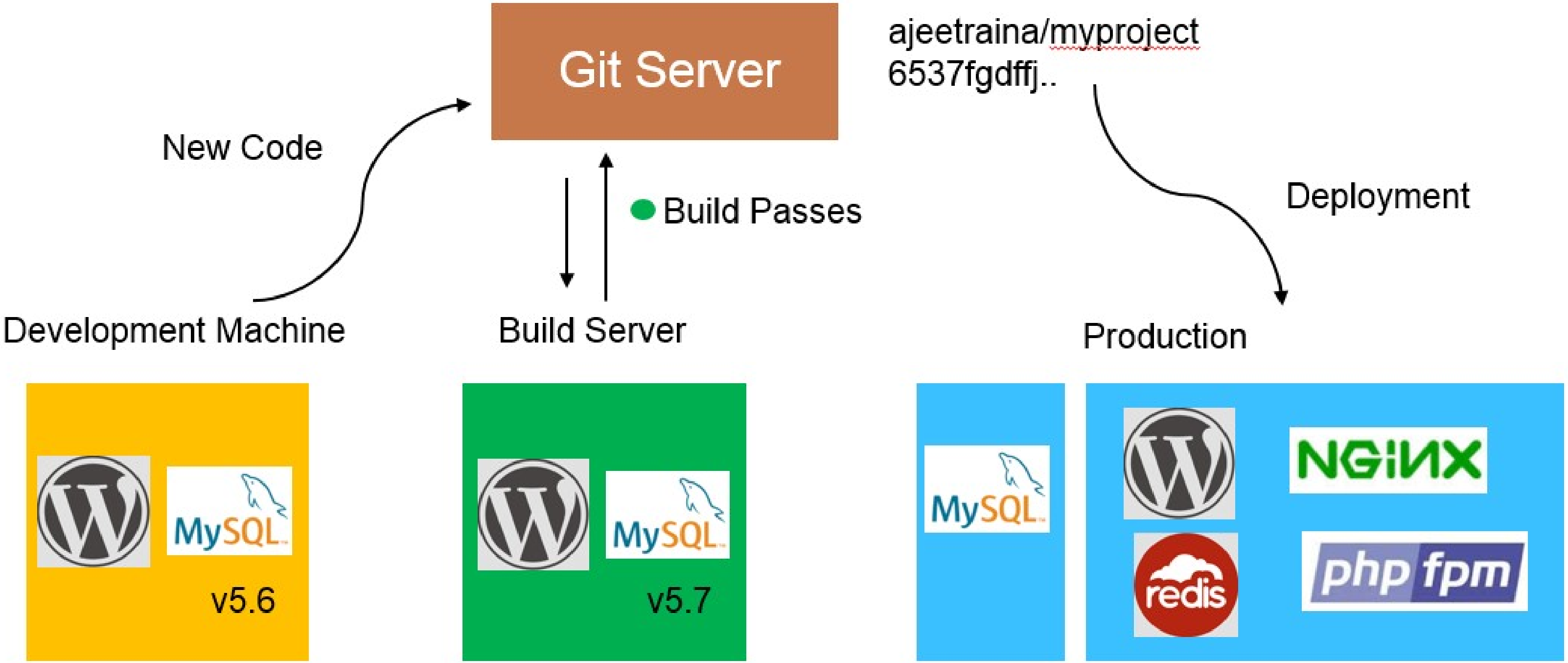
Build Server

Production

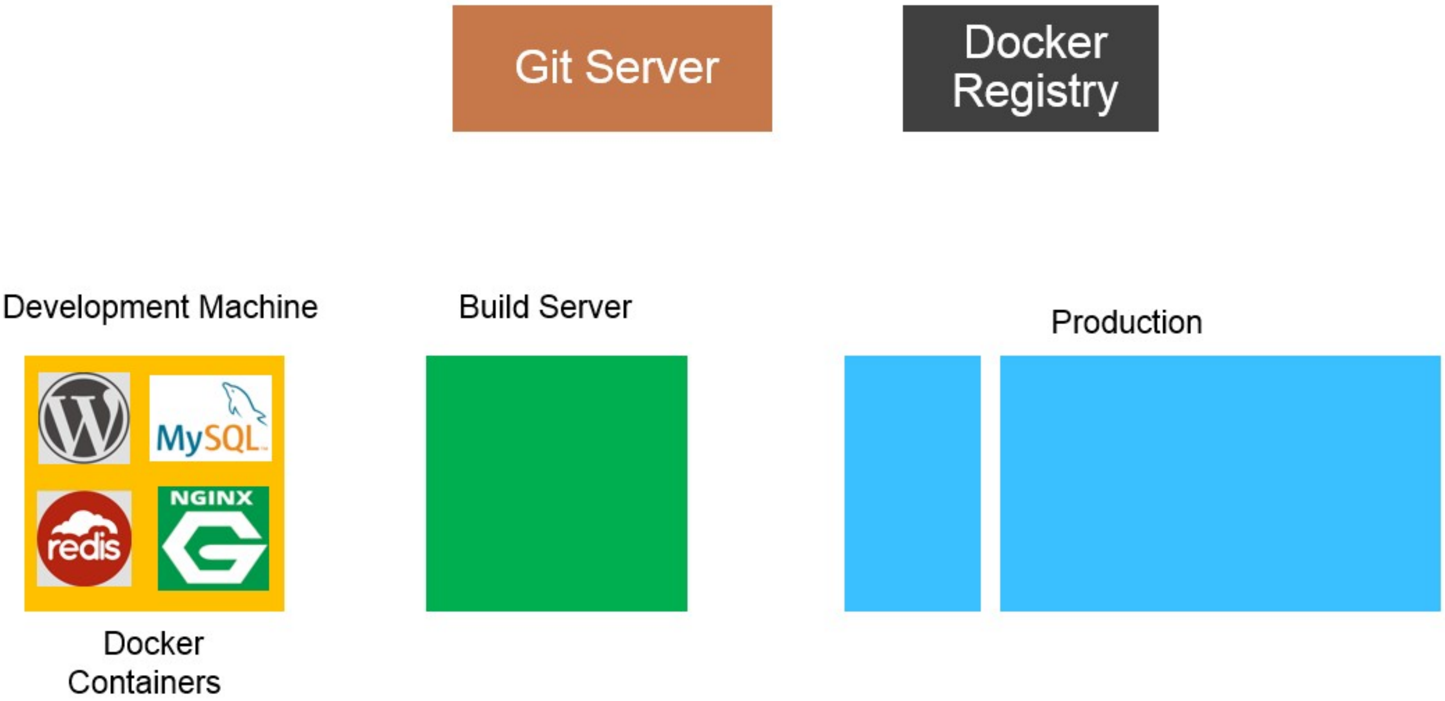
v5.6

v5.7

Development workflow (without Docker)



Development workflow (with Docker)



Git Server

Docker Registry

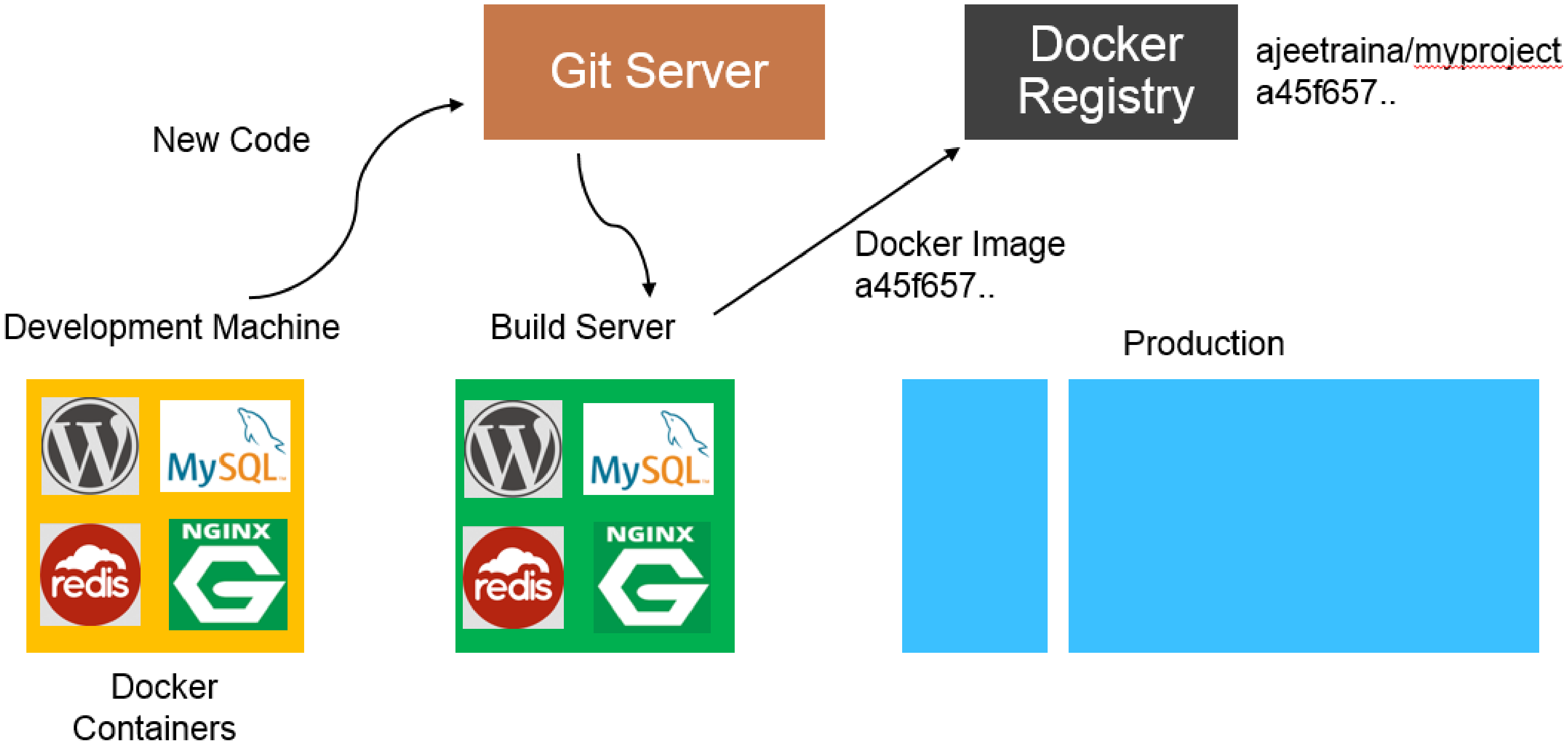
Development Machine

Build Server

Production

Docker Containers

Development workflow (with Docker)



Git Server

Docker Registry

ajeetraina/myproject
a45f657..

New Code

Docker Image
a45f657..

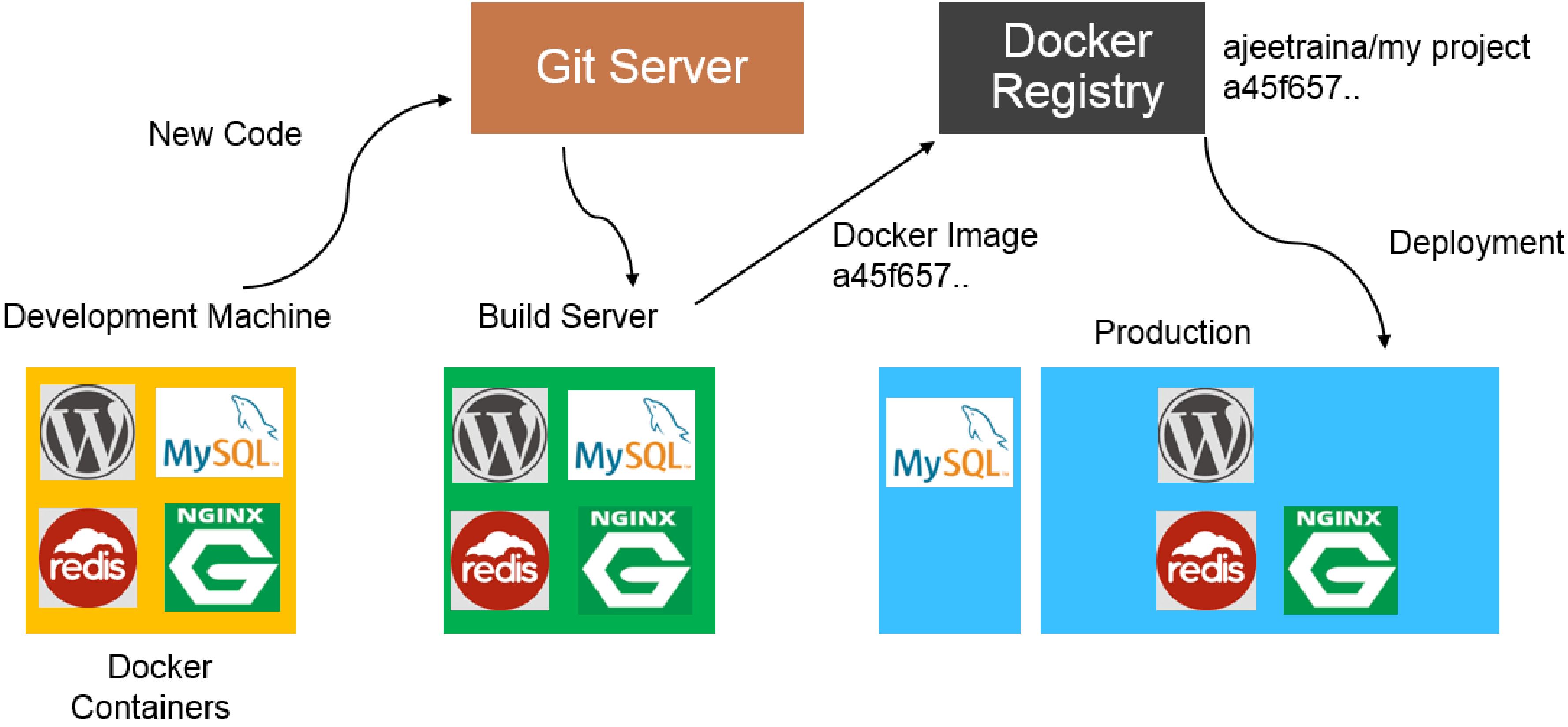
Development Machine

Build Server

Production

Docker
Containers

Development workflow (with Docker)



Git Server

Docker Registry

ajeetraina/my project
a45f657..

New Code

Docker Image
a45f657..

Deployment

Development Machine

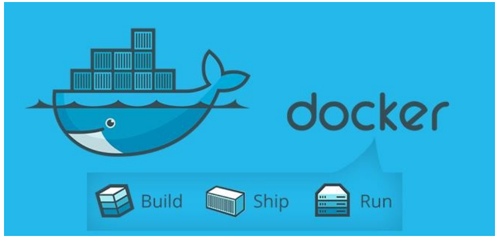
Build Server

Production

Docker
Containers

What is Docker?

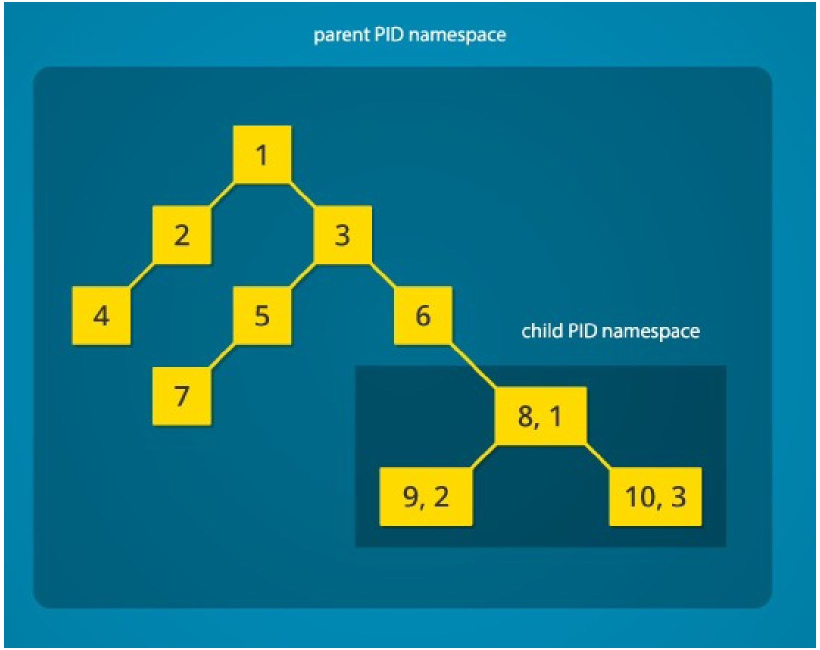
- A tool that can package an application and its dependencies in a *virtual container*
- Implementation of a container which is portable using a concept of *image*
- Docker uses the host OS kernel, there is no custom or additional kernel inside running containers
- Docker uses resource isolation features of the Linux kernel such as cgroups and kernel namespaces to allow independent “containers” to run within a single Linux instance, avoiding the overhead of starting virtual machines



A note on Linux namespaces

- Isolation for several aspects of processes and resources
- See for example: Process ID isolation
  Processes in the child namespace do not see the parent process’s existence;
  processes in the parent namespace have a complete view of processes in the child namespace

- Still, processes can compete for exclusive access to shared **real** resources (e.g. open a socket on port 80)

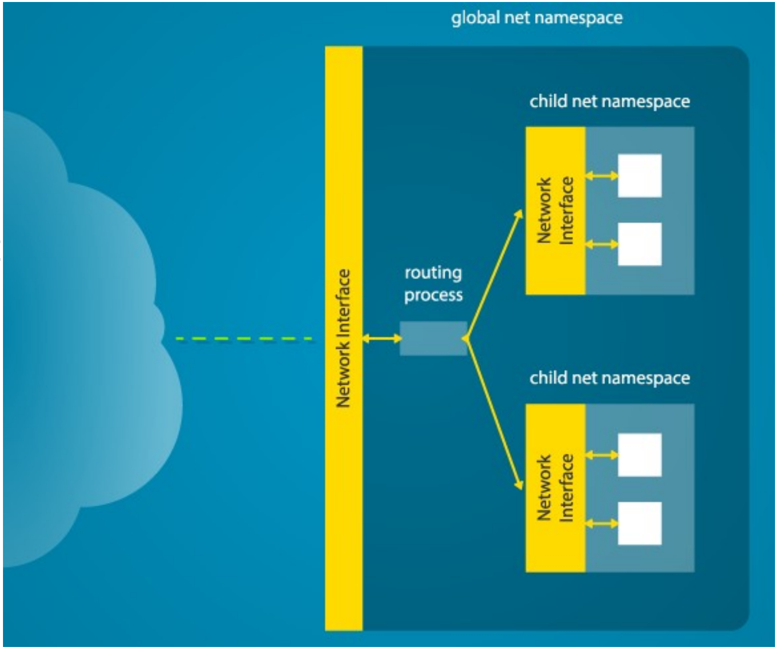


A note on Linux namespaces (cont.)

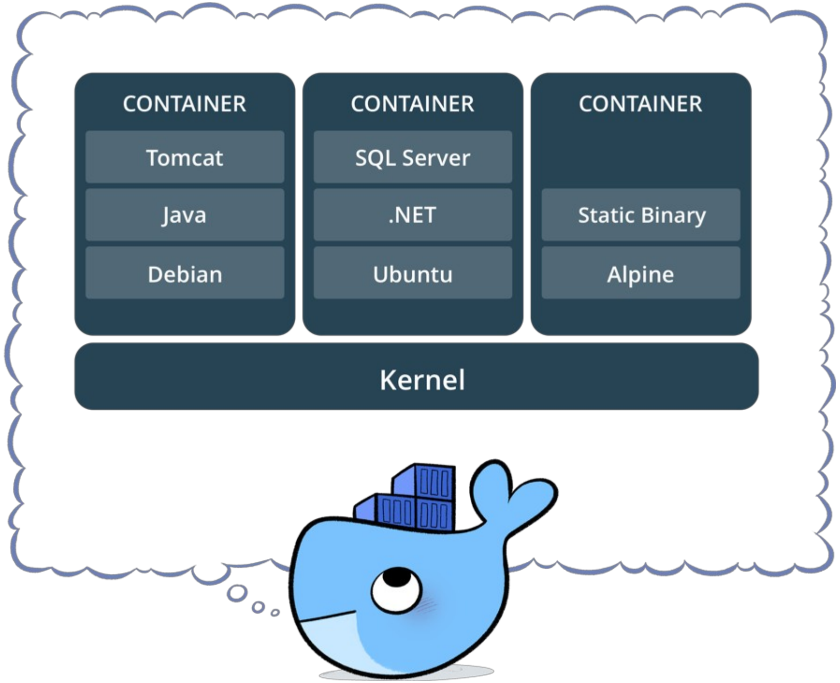
- Isolation for several aspects of processes and resources
- See for example: Process ID isolation
  Processes in the child namespace do not see the parent process’s existence;
  processes in the parent namespace have a complete view of processes in the child namespace

- Still, processes can compete for exclusive access to shared **real** resources (e.g. open a socket on port 80)

  - A pair of virtual Ethernet connections (ends) must be created, between a parent and a child namespace
  - Both ends must be assigned a virtual IP address



What is Docker?



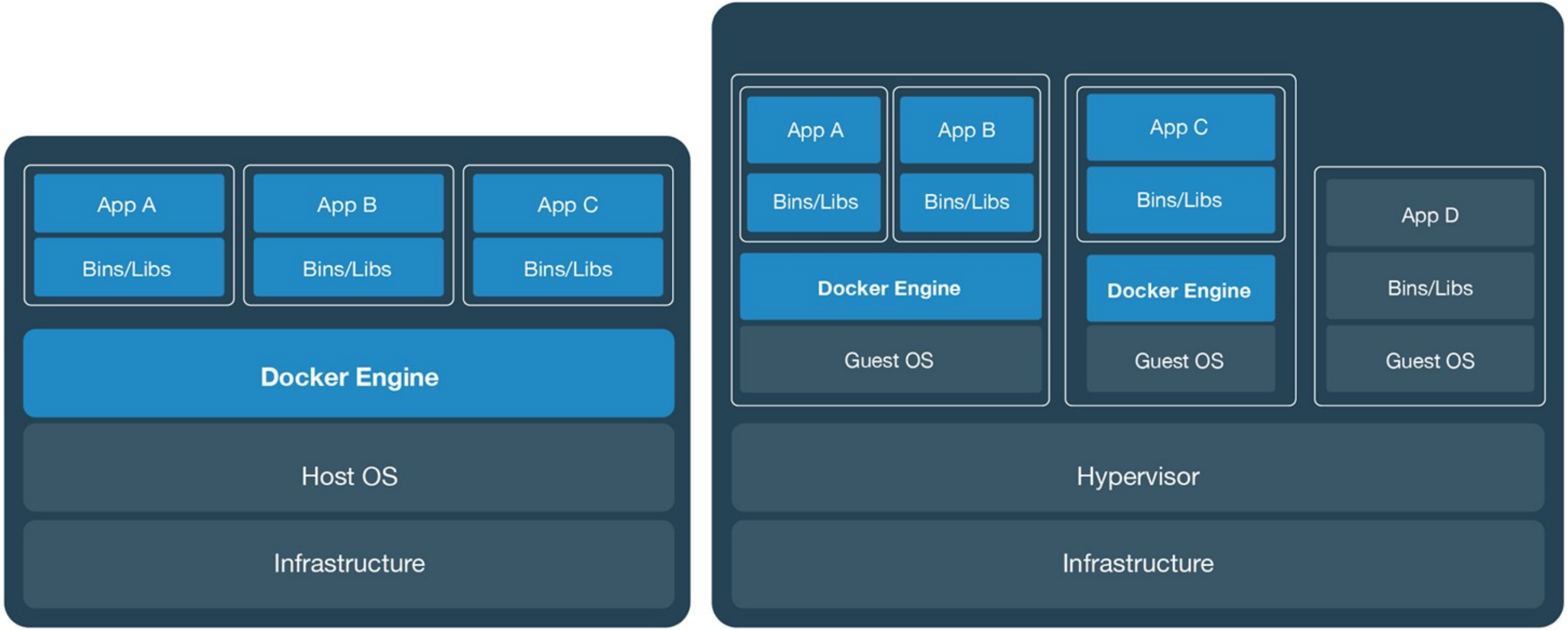
CONTAINER	CONTAINER	CONTAINER
Tomcat	SQL Server	
Java	.NET	Static Binary
Debian	Ubuntu	Alpine
Kernel		

- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Available for Windows (Server, since 2016) and MacOS

VMs vs Docker - Differences

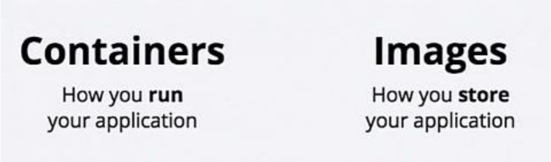
Virtual Machines	Docker
Each VM runs its own OS	All containers share the same kernel of the host
Boot up time is in minutes	Containers instantiate in seconds
VMs snapshots are used sparingly	Images are built incrementally on top of another like layers. Lots of images/snapshots
Not effective diffs. Not version controlled	Images can be diffed and can be version controlled. Dockerhub is like GITHUB
Cannot run more than couple of VMs on an average laptop	Can run many Docker containers in a laptop.
Only one VM can be started from one set of VMX and VMDK files	Multiple Docker containers can be started from one Docker image

Containers versus VMs



- When and when not? The GPU example...

Some Docker vocabulary



Containers

How you **run** your application

Images

How you **store** your application

Docker Image

The basis of a Docker container. Represents a full application
Specified via *Dockerfiles*

Docker Container

The standard unit in which the application service resides and executes

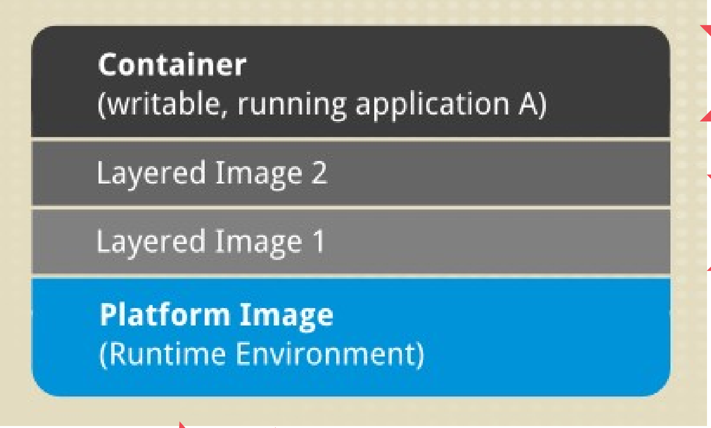
Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub or Docker Trusted Registry)

Cloud or server based storage and distribution service for your images

Image Layering



Container
(writable, running application A)

Layered Image 2

Layered Image 1

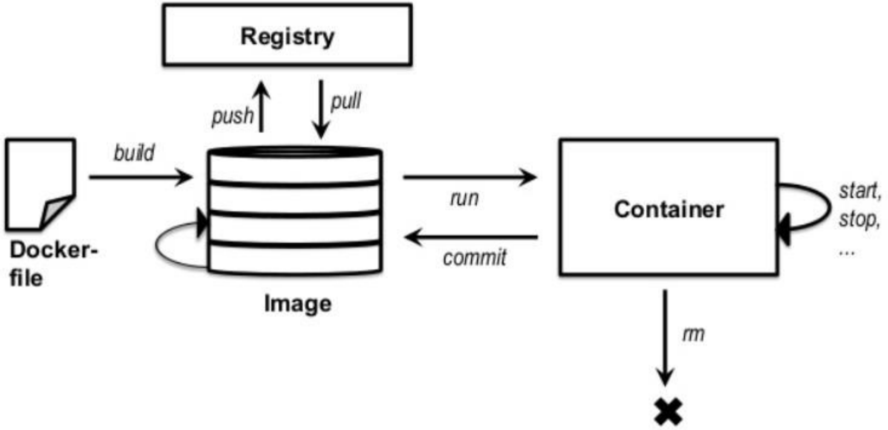
Platform Image
(Runtime Environment)

- An application sandbox.
- Each container is based on an image that holds necessary config data.
- When you launch a container from an image, a writable layer is added on top of this image

- A static snapshot of the containers' configuration.
- Image is a read-only layer that is never modified, all changes are made in top-most writable layer, and can be saved only by creating a new image.
- Each image depends on one or more parent images

- An image that has no parent.
- Platform images define the runtime environment, packages and utilities necessary for containerized application to run.

Basic Docker Commands



Pulling Docker Image

$ docker pull fedora/httpd:version1.0

Listing out/removing Docker Images

$ docker image ls
$ docker rmi fedora/httpd:version1.0

Running Docker Containers

$ docker container run –d –p 5000:5000 –-name httpserver fedora/httpd:version1.0

Stopping the container

$ docker container stop httpserver (or <container id>)

Copying files from/to a container (volumes can also be used)

$ docker cp <container id>:<path> <host_path>

Execute commands in a running container

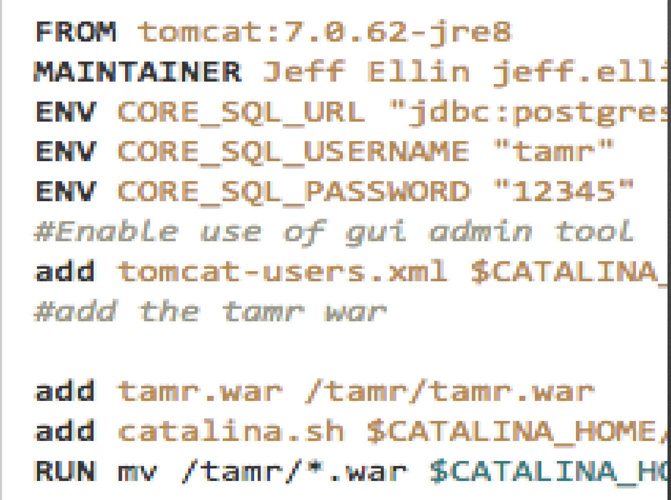
$ docker exec -it <container id> /bin/bash

Dockerfile Basics

Docker Images are built from a base image.

Base Images are built up using simple instructions such as

- Run a command.
- Add a file or directory.
- Create an environment variable.
- What process to run when this image.



```
FROM tomcat:7.0.62-jre8
MAINTAINER Jeff Ellin jeff.elli
ENV CORE_SQL_URL "jdbc:postgres
ENV CORE_SQL_USERNAME "tamr"
ENV CORE_SQL_PASSWORD "12345"
#Enable use of gui admin tool
add tomcat-users.xml $CATALINA_
#add the tamr war

add tamr.war /tamr/tamr.war
add catalina.sh $CATALINA_HOME/
RUN mv /tamr/*.war $CATALINA_H
```

FROM

The FROM instruction sets the Base Image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. The image can be any valid image – it is especially easy to start by pulling an image from the Public Repositories.

```
FROM java:8-jre
```

ENV

The ENV instruction is also useful for providing required environment variables specific to services you wish to containerize, such as Postgres’s PGDATA.

```
ENV TOMCAT_MAJOR 8
ENV TOMCAT_VERSION 8.0.26
```

RUN

The instruction will execute any commands in a new layer on top of the current image and commit results. The resulting committed image is used for the next step in the Dockerfile.

```
RUN apt-get update && apt-get install -y \
bzr \
cvs \
git
```

ADD and Copy

These commands can be used to add files to the container

- For ADD if source is a tar file it is extracted

- ADD allows source file to be a URL

- Use a trailing slash to indicate a directory vs a file.

```
COPY hom* /mydir/        # adds files starting with "hom"
COPY hom?.txt /mydir/  # ? replaced with any single char
```

EXPOSE

Informs Docker that the container
will listen on the specified network ports at runtime.
This is used to interconnect containers using links
(see the Docker User Guide) and to determine which
ports to expose to the host when using the -P flag.

```
EXPOSE 8080
```

WORKDIR

The WORKDIR instruction sets the working directory for any RUN, CMD, COPY and ADD instructions that follow it in the Dockerfile.

It can be used multiple times in the Dockerfile. If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction.

```
WORKDIR $CATALINA_HOME
```

CMD

The main purpose of a CMD is to provide defaults for an executing container.

Can be overridden with arguments to docker run

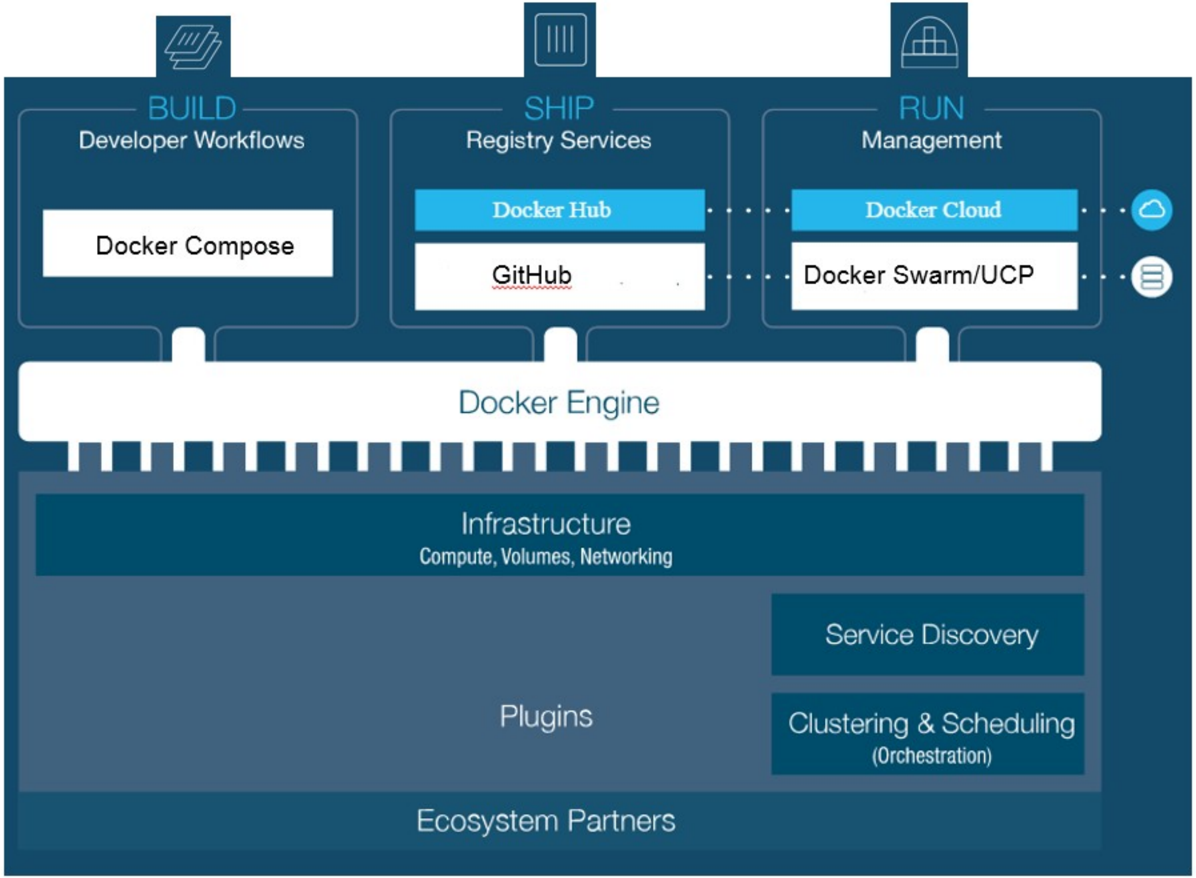
```
CMD ["catalina.sh", "run"]
```

Hands-on exercise

a) Install Docker (sudo apt install docker.io)
b) Create a folder, a bash script and a Dockerfile
c) Instruct the Dockerfile to execute the script at container startup
d) The script shold list the contents of “/” and place the result in a file
e) Build the image (docker build -t image_name .)
f) Start a container based on the created image
g) Let us use “docker exec” to log in the container and show the results
h) Let us use “docker cp” to copy the output file into the host machine

Build, Ship & Run

Docker Mission



BUILD
Developer Workflows

Docker Compose

SHIP
Registry Services

Docker Hub

GitHub

RUN
Management

Docker Cloud

Docker Swarm/UCP

Docker Engine

Infrastructure
Compute, Volumes, Networking

Plugins

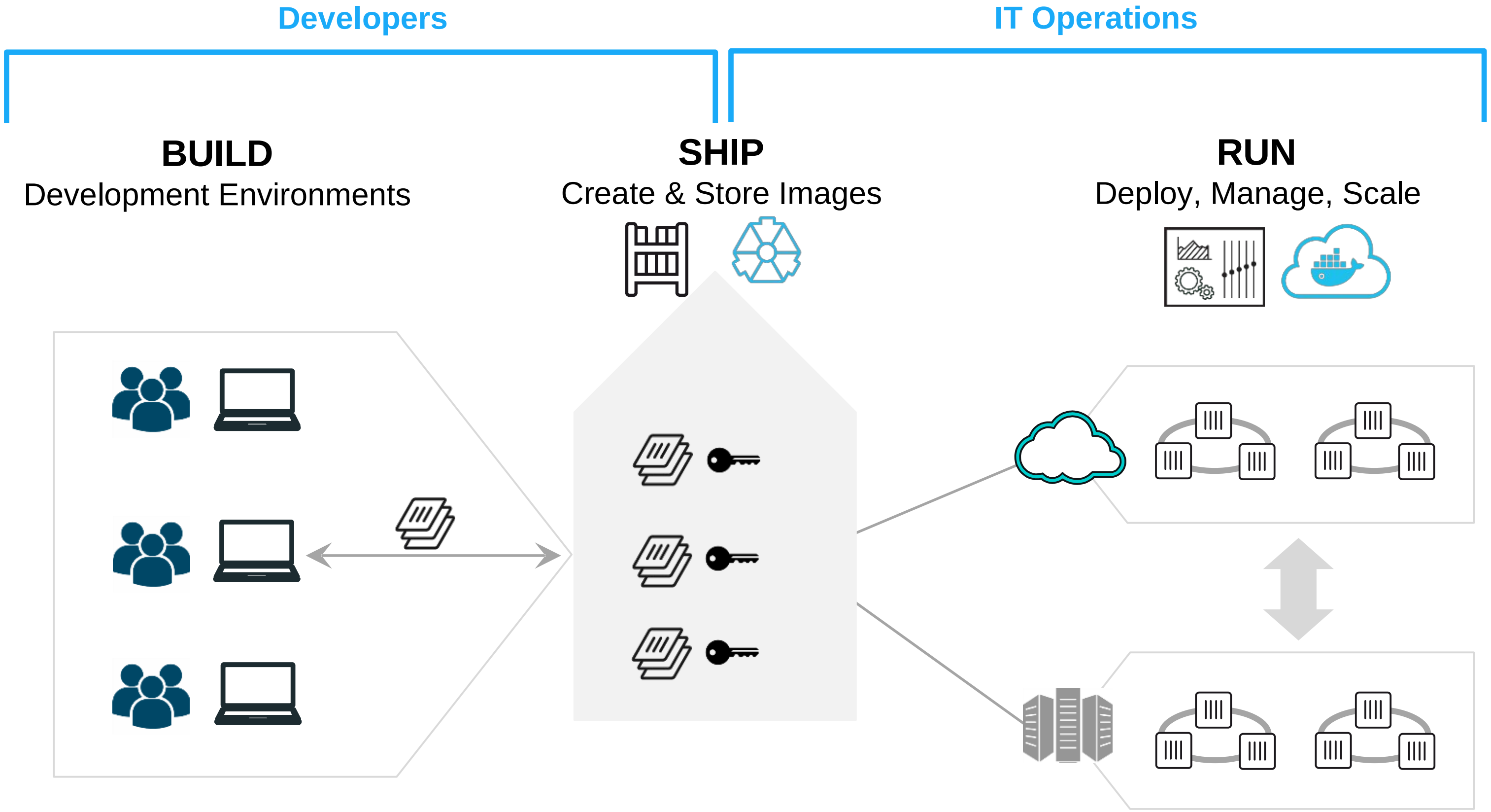
Service Discovery

Clustering & Scheduling
(Orchestration)

Ecosystem Partners

docker

Put it all together: Build, Ship, Run Workflow



Developers

IT Operations

BUILD

Development Environments

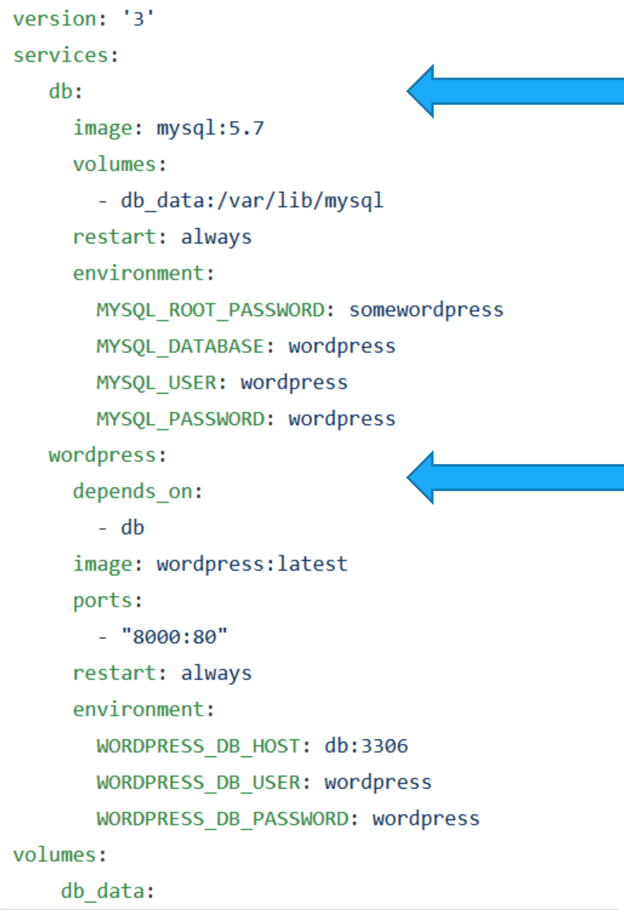
SHIP

Create & Store Images

RUN

Deploy, Manage, Scale

Docker Compose – Building Microservices in easy way



```
version: '3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Backend Service

Specify Volumes/Network

Environmental variables

Frontend Service

Specify Volumes/Network

Environmental variables