You make **possible**

Kuber-What?

Introduction to Kubernetes

Matt Johnson– Developer Advocacy, Cisco DevNet
@metahertz

DEVNET-1999

Barcelona | January 27-31, 2020

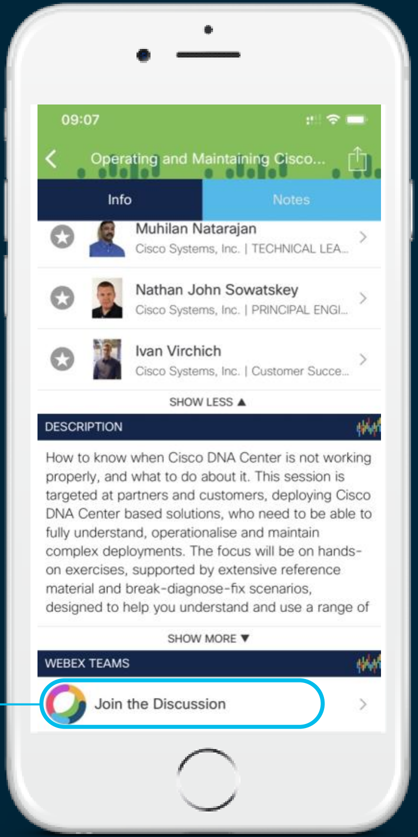
Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat with the speaker after the session

How

1. Find this session in the Cisco Events Mobile App
2. Click “Join the Discussion”
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space



09:07

Operating and Maintaining Cisco...

Info | Notes

Muhilan Natarajan
Cisco Systems, Inc. | TECHNICAL LEA...

Nathan John Sowatskey
Cisco Systems, Inc. | PRINCIPAL ENGI...

Ivan Virchich
Cisco Systems, Inc. | Customer Succe...

SHOW LESS ▲

DESCRIPTION

How to know when Cisco DNA Center is not working properly, and what to do about it. This session is targeted at partners and customers, deploying Cisco DNA Center based solutions, who need to be able to fully understand, operationalise and maintain complex deployments. The focus will be on hands-on exercises, supported by extensive reference material and break-diagnose-fix scenarios, designed to help you understand and use a range of

SHOW MORE ▼

WEBEX TEAMS

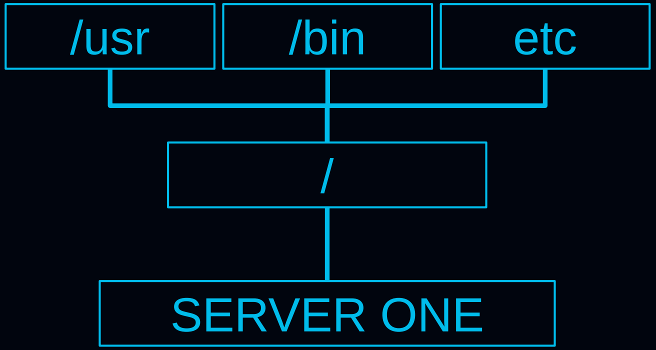
Join the Discussion

Agenda

- Introduction.
- A Brief Primer on Containers.
- The Problems with Containers at Scale.
- Orchestration Systems.
- Kubernetes Background.
- Using Kubernetes.
- Build vs Buy.

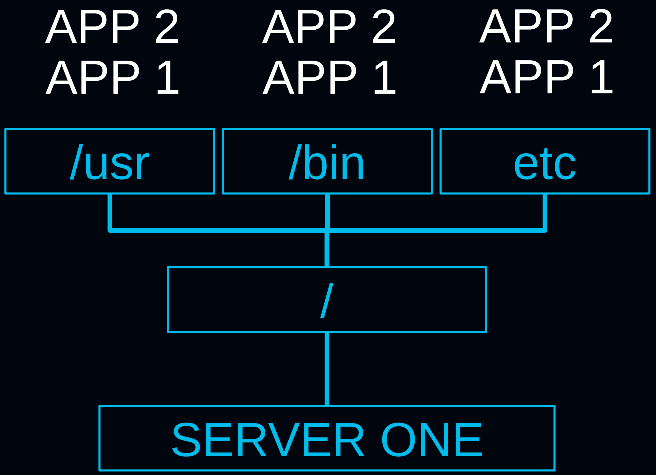
A Brief History Lesson One

Old School App Deployments.



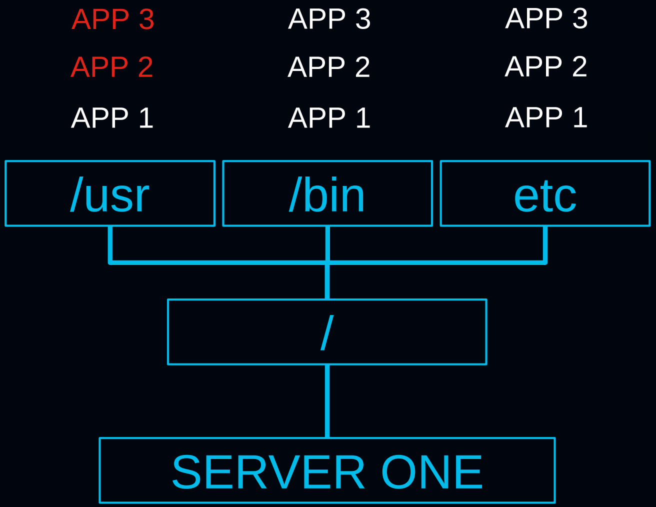
A Brief History Lesson One

Old School App Deployments.



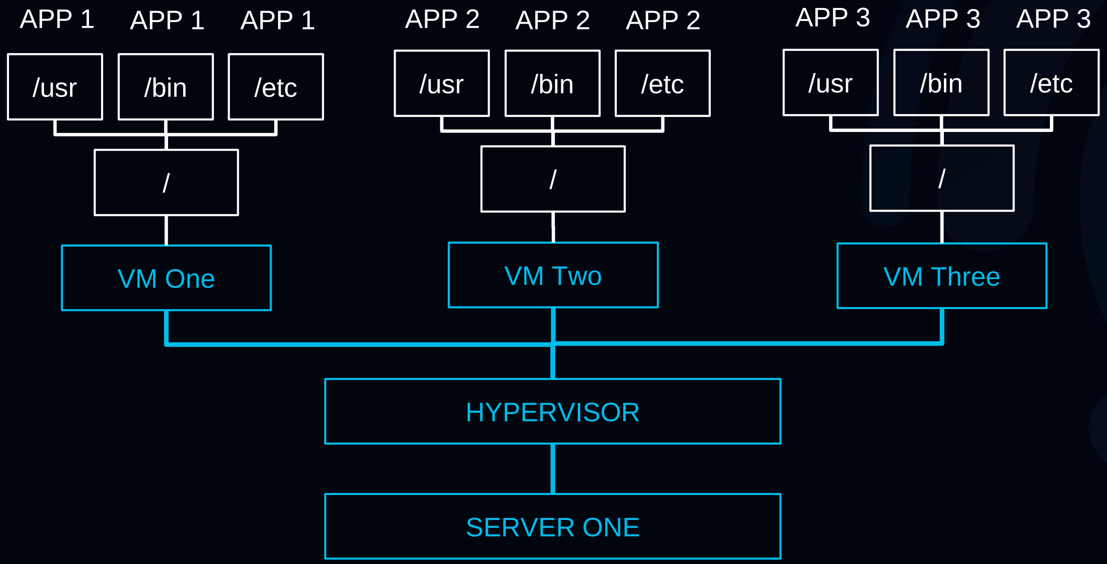
A Brief History Lesson One

Old School App Deployments.



A Brief History Lesson One

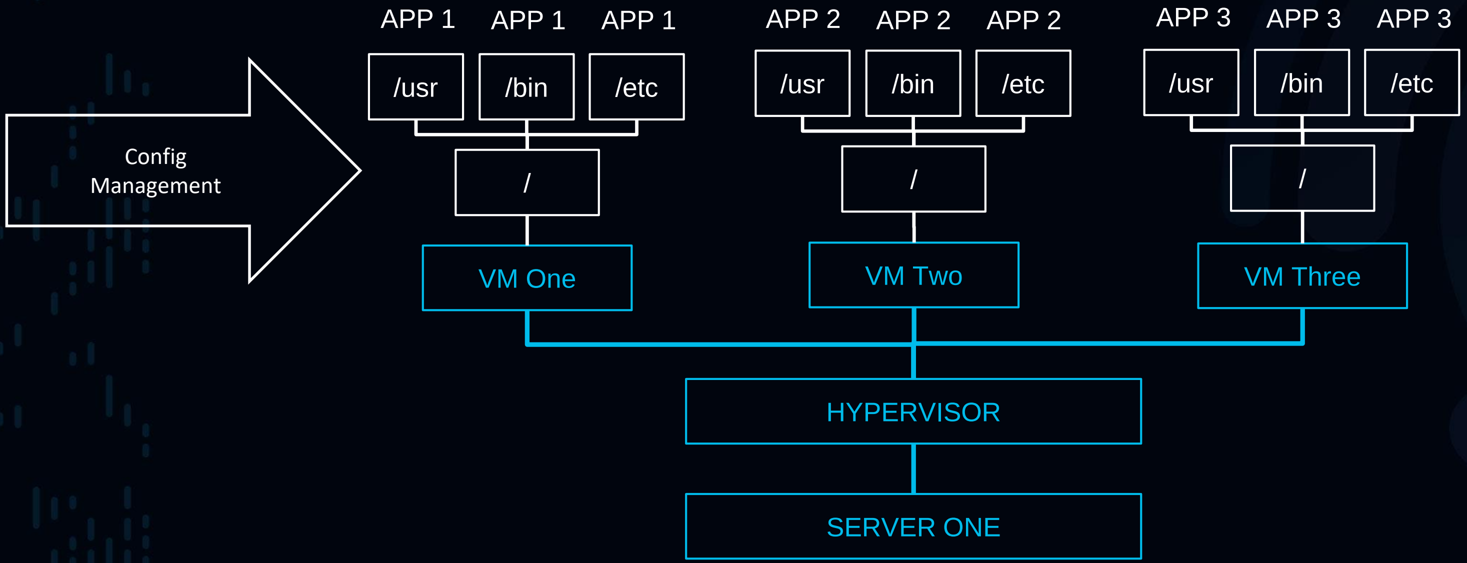
Virtual Machines



All Applications Happy. More “Servers” to manage.

A Brief History Lesson One

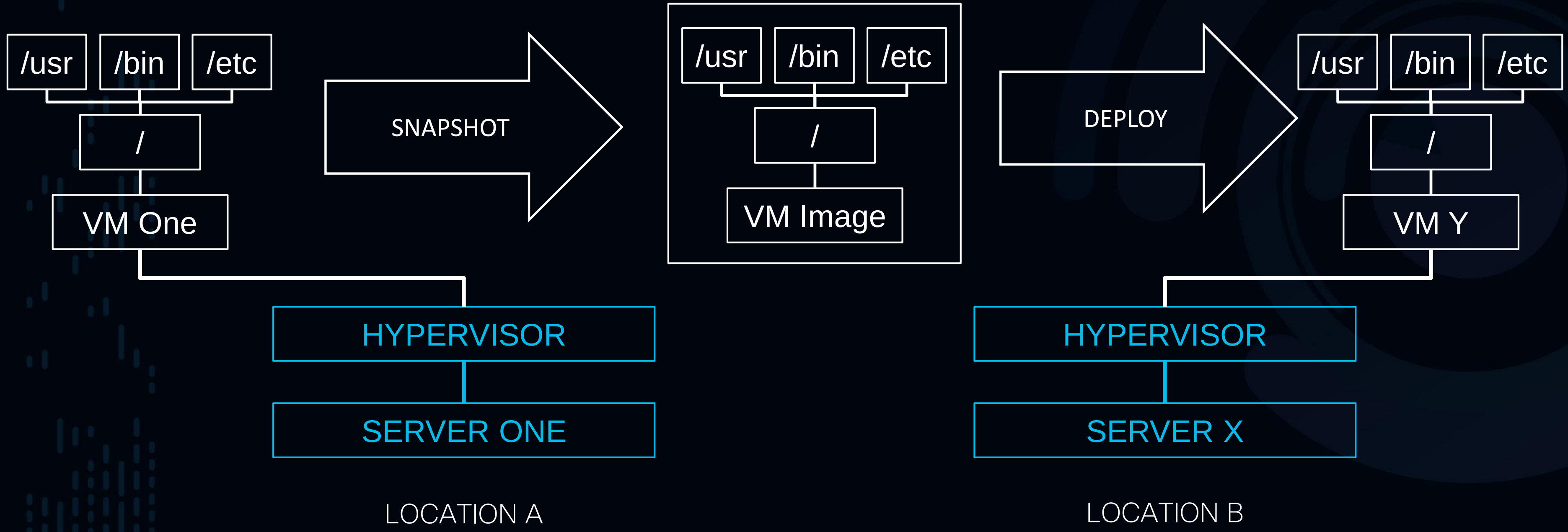
Virtual Machines



Configuration Management – Great until it isn’t.

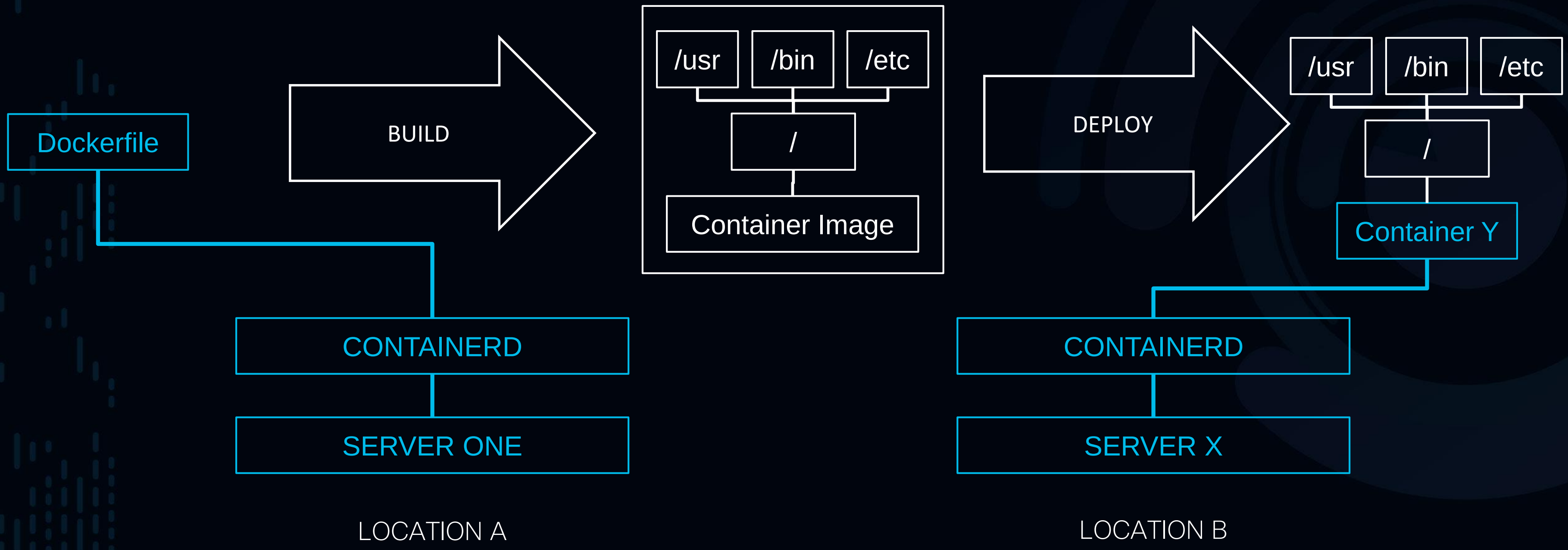
Known good images.

Things don’t change. New versions replace them.



Known good images: Take Two.

Things don’t change. New versions replace them.



Containers

Containers Are...

- A way to package up our applications and dependencies.
- A way to guarantee execution consistency and portability.
- A lightweight way to keep your applications isolated.
- A way to use your compute resources without the overhead of VM's.

Containers Are not…

- Microservices
  - We hear containers and microservice used a lot together.
  - Microservices benefit from a lightweight packaging, distribution and deployment solution.
  - However, you can put package anything into a container, including a badly written legacy app in some cases, using containers doesn’t magically make bad code better.
- VM’s
  - Containers are purely user-space, if you need kernel extensions/modules or a custom kernel, containers probably aren't what you’re looking for.
- Magic
  - They bring their own nuances and require deployment consideration just like any other toolchain.

“Container”

That same concept as building VM images, but with much better developer user experience.

BUILD TOOLING
$ docker build .

STANDARD FORMAT
$ docker images

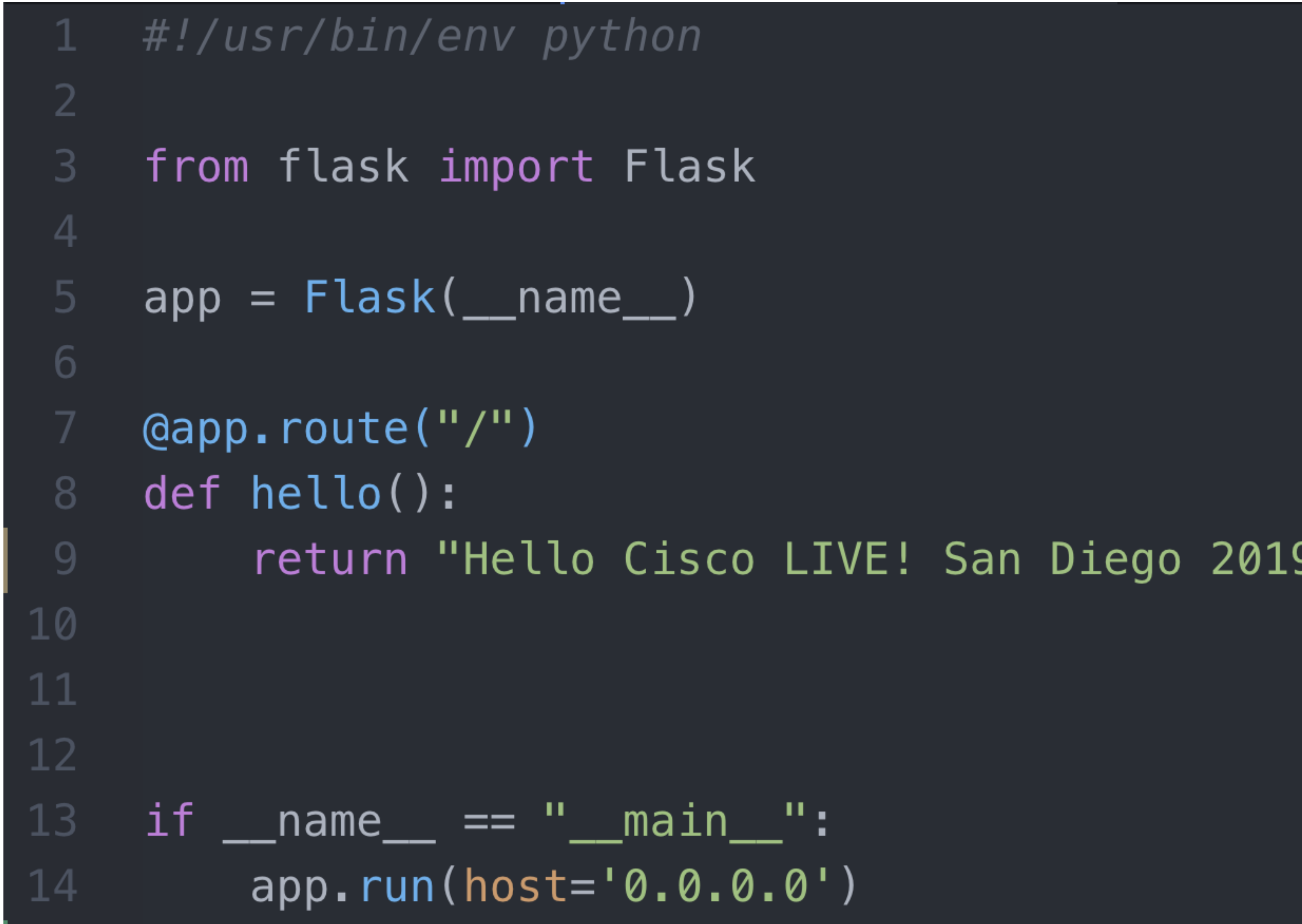
DISTRIBUTION & VERSIONING
$ docker push
$ docker pull

RUNNING
$ docker run

We’ll be talking about Docker’s flavor of container + toolchain from here on out.

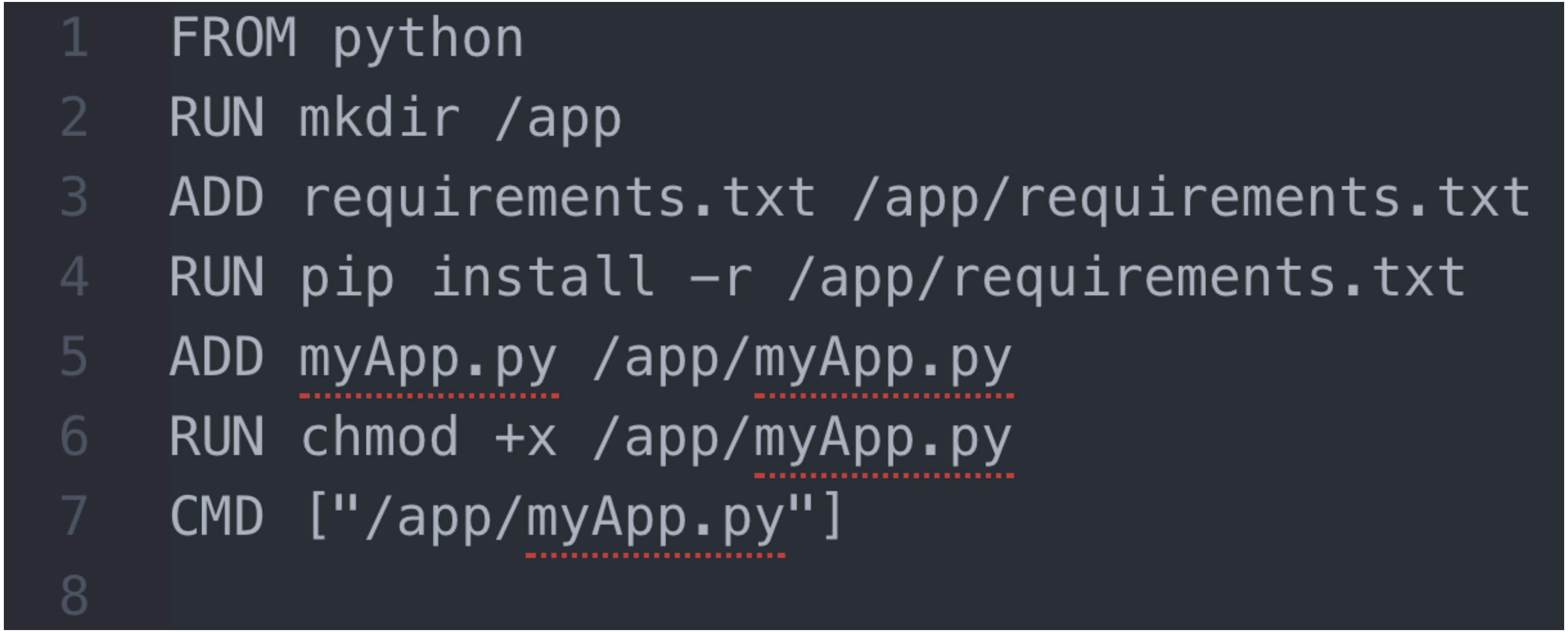
Docker

myApp.py



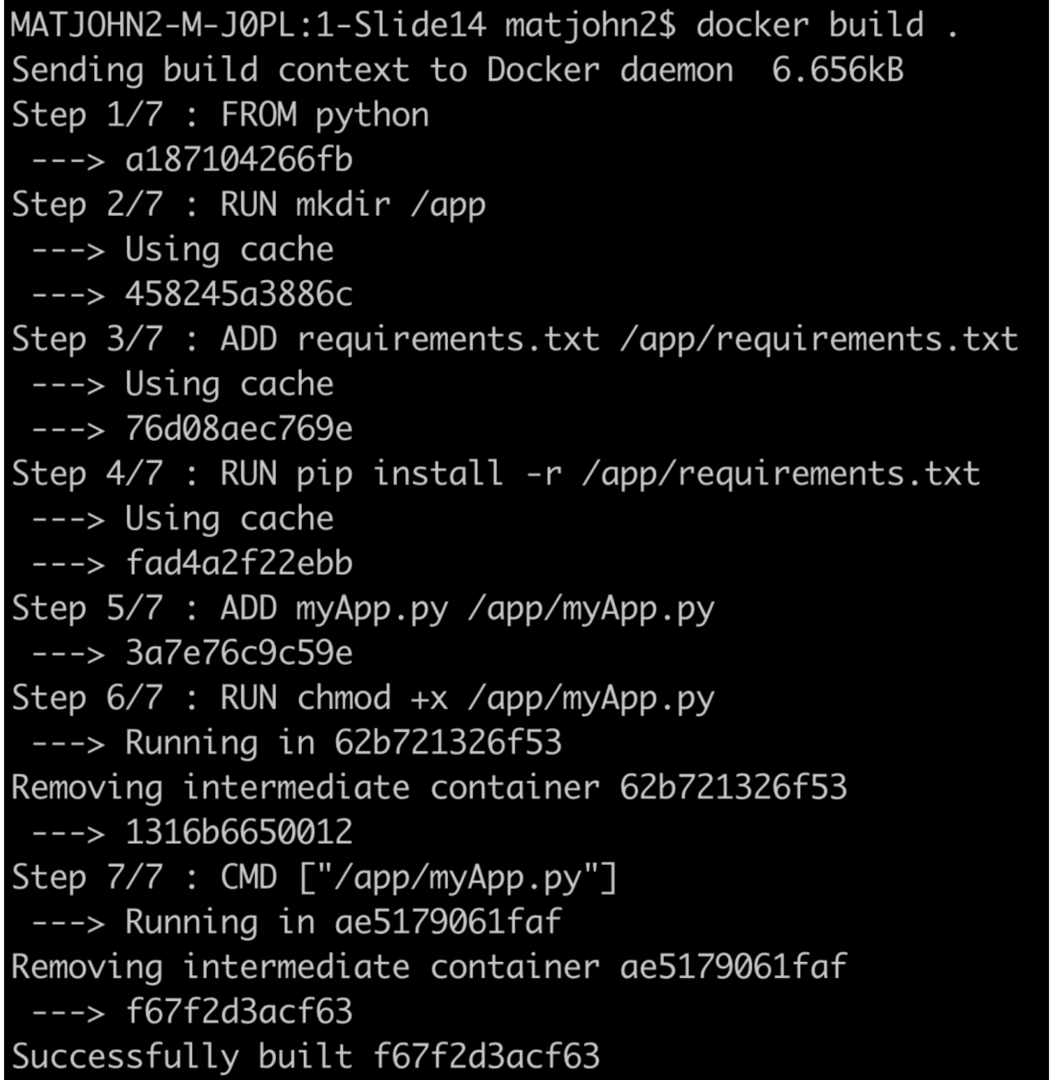
```python
#!/usr/bin/env python

from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello Cisco LIVE! San Diego 2019



if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Dockerfile

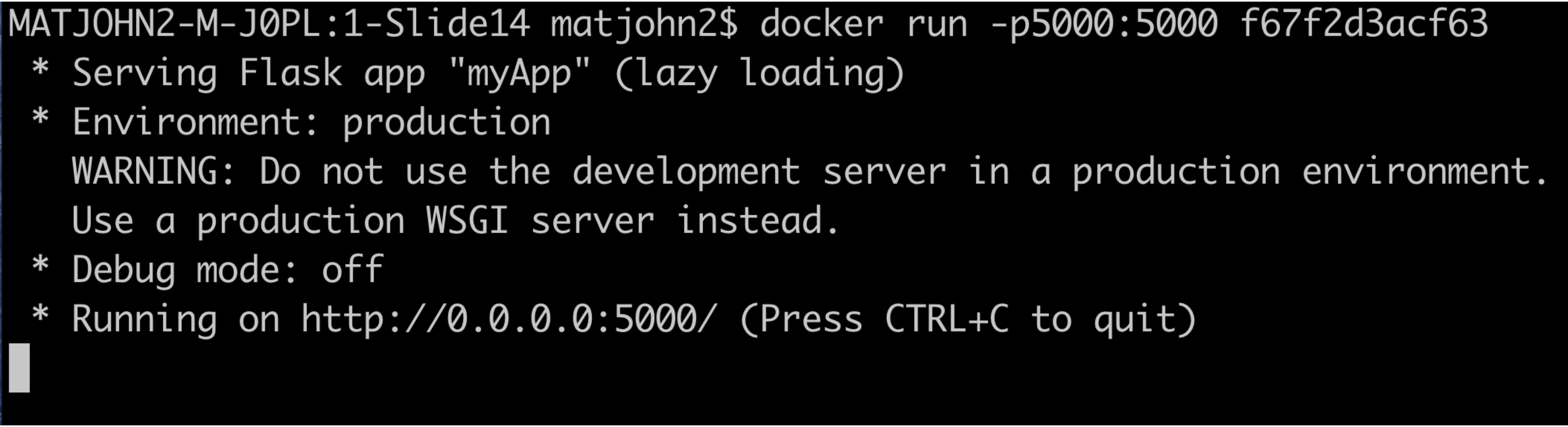


```
FROM python
RUN mkdir /app
ADD requirements.txt /app/requirements.txt
RUN pip install -r /app/requirements.txt
ADD myApp.py /app/myApp.py
RUN chmod +x /app/myApp.py
CMD ["/app/myApp.py"]
```

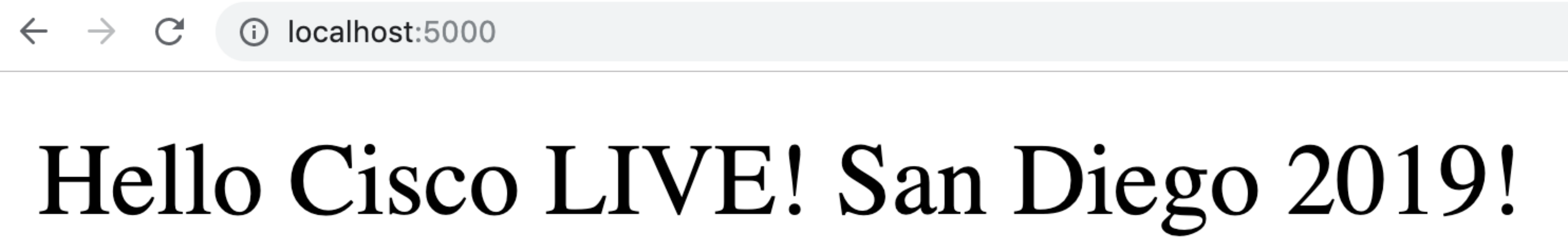
Docker Build



```
MATJOHN2-M-J0PL:1-Slide14 matjohn2$ docker build .
Sending build context to Docker daemon  6.656kB
Step 1/7 : FROM python
 ---> a187104266fb
Step 2/7 : RUN mkdir /app
 ---> Using cache
 ---> 458245a3886c
Step 3/7 : ADD requirements.txt /app/requirements.txt
 ---> Using cache
 ---> 76d08aec769e
Step 4/7 : RUN pip install -r /app/requirements.txt
 ---> Using cache
 ---> fad4a2f22ebb
Step 5/7 : ADD myApp.py /app/myApp.py
 ---> 3a7e76c9c59e
Step 6/7 : RUN chmod +x /app/myApp.py
 ---> Running in 62b721326f53
Removing intermediate container 62b721326f53
 ---> 1316b6650012
Step 7/7 : CMD ["/app/myApp.py"]
 ---> Running in ae5179061faf
Removing intermediate container ae5179061faf
 ---> f67f2d3acf63
Successfully built f67f2d3acf63
```

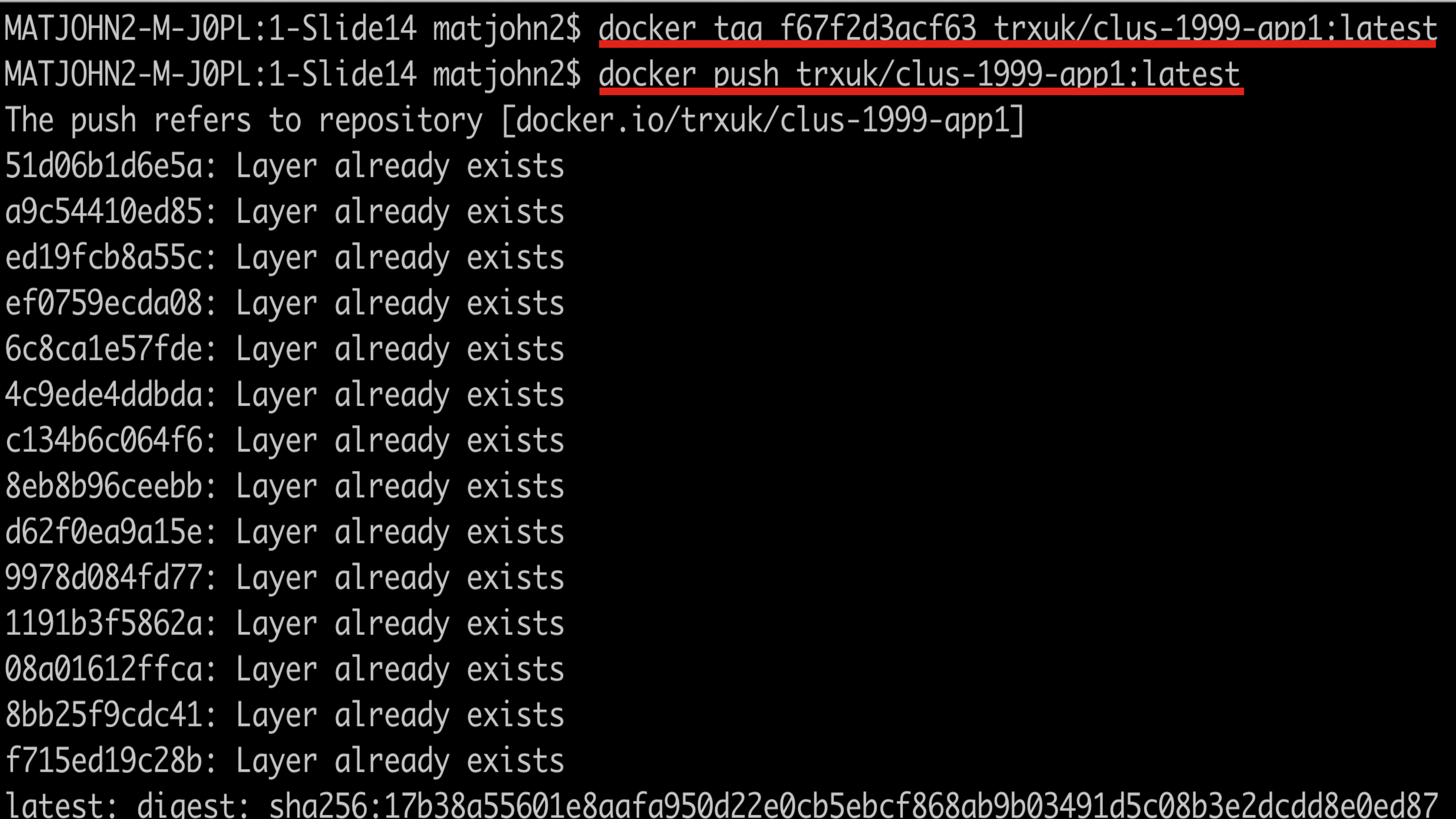


```
MATJOHN2-M-J0PL:1-Slide14 matjohn2$ docker run -p5000:5000 f67f2d3acf63
 * Serving Flask app "myApp" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```



localhost:5000

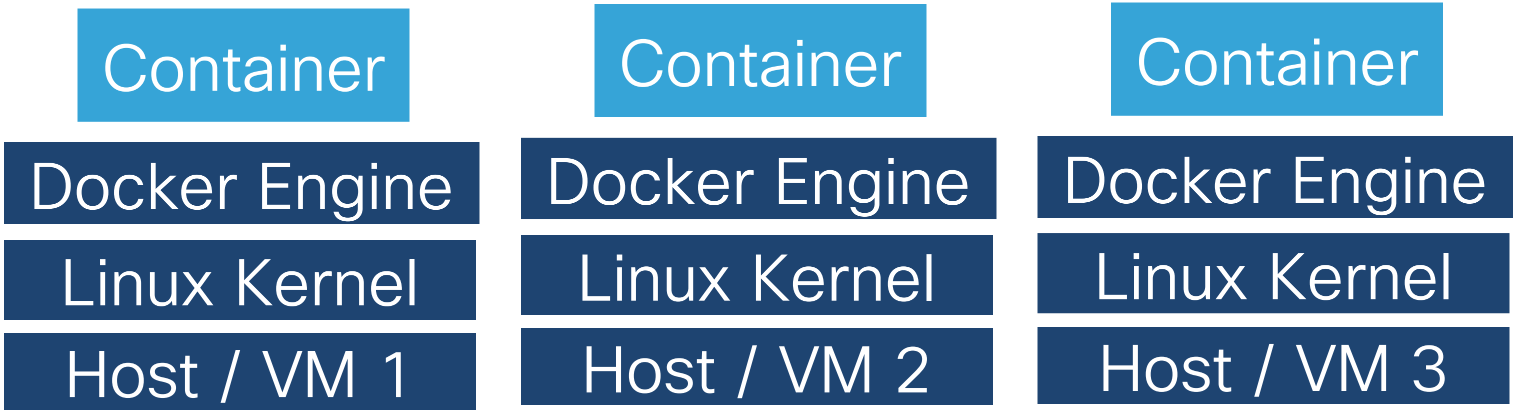
Hello Cisco LIVE! San Diego 2019!



```
MATJOHN2-M-J0PL:1-Slide14 matjohn2$ docker tag f67f2d3acf63 trxuk/clus-1999-app1:latest
MATJOHN2-M-J0PL:1-Slide14 matjohn2$ docker push trxuk/clus-1999-app1:latest
The push refers to repository [docker.io/trxuk/clus-1999-app1]
51d06b1d6e5a: Layer already exists
a9c54410ed85: Layer already exists
ed19fcb8a55c: Layer already exists
ef0759ecda08: Layer already exists
6c8ca1e57fde: Layer already exists
4c9ede4ddbda: Layer already exists
c134b6c064f6: Layer already exists
8eb8b96ceebb: Layer already exists
d62f0ea9a15e: Layer already exists
9978d084fd77: Layer already exists
1191b3f5862a: Layer already exists
08a01612ffca: Layer already exists
8bb25f9cdc41: Layer already exists
f715ed19c28b: Layer already exists
latest: digest: sha256:17b38a55601e8aafa950d22e0cb5ebcf868ab9b03491d5c08b3e2dcdd8e0ed87
```

Using `docker run` is good for Development.

But not Production…



Container
Docker Engine
Linux Kernel
Host / VM 1

Container
Docker Engine
Linux Kernel
Host / VM 2

Container
Docker Engine
Linux Kernel
Host / VM 3

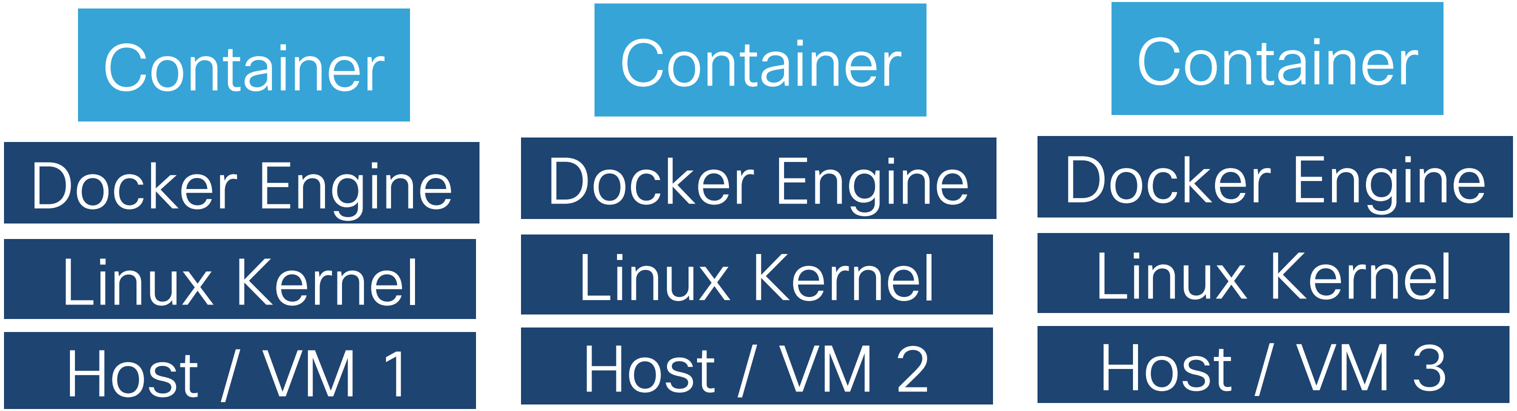
```
$ ssh host1
host1# docker run container
```

```
$ ssh host2
host2# docker run container
```

```
$ ssh host3
host3# docker run container
```

Using `docker run` is good for Development...

But not Production...



Container | Container | Container

Docker Engine | Docker Engine | Docker Engine

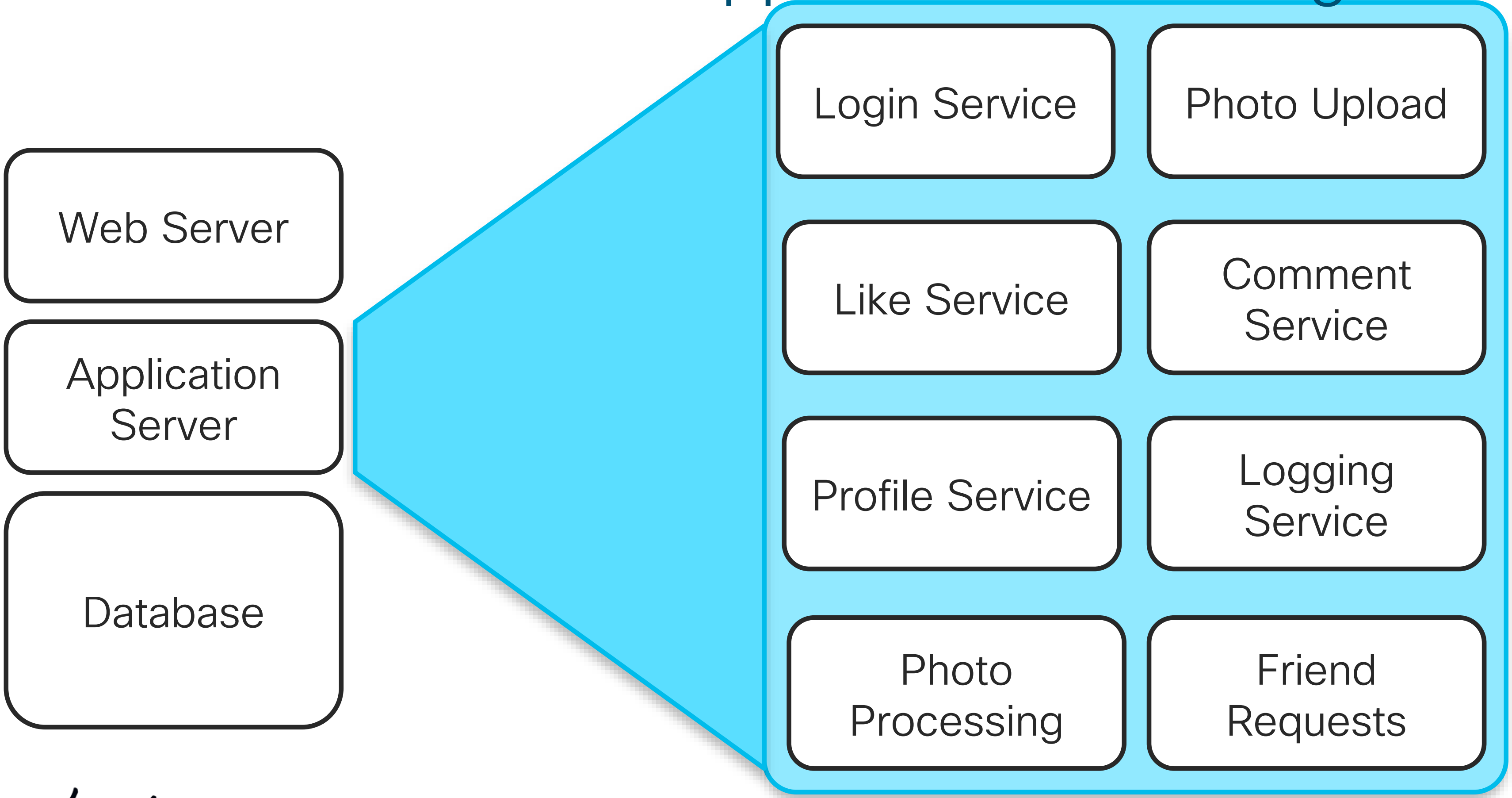
Linux Kernel | Linux Kernel | Linux Kernel

Host / VM 1 | Host / VM 2 | Host / VM 3

What about LoadBalancing?
What about Passwords and Secrets?
What about Networking?
What about monitoring and restarting?

Issues with Containers at Scale.

Microservices == More Applications to Manage



Web Server

Application Server

Database

Login Service

Photo Upload

Like Service

Comment Service

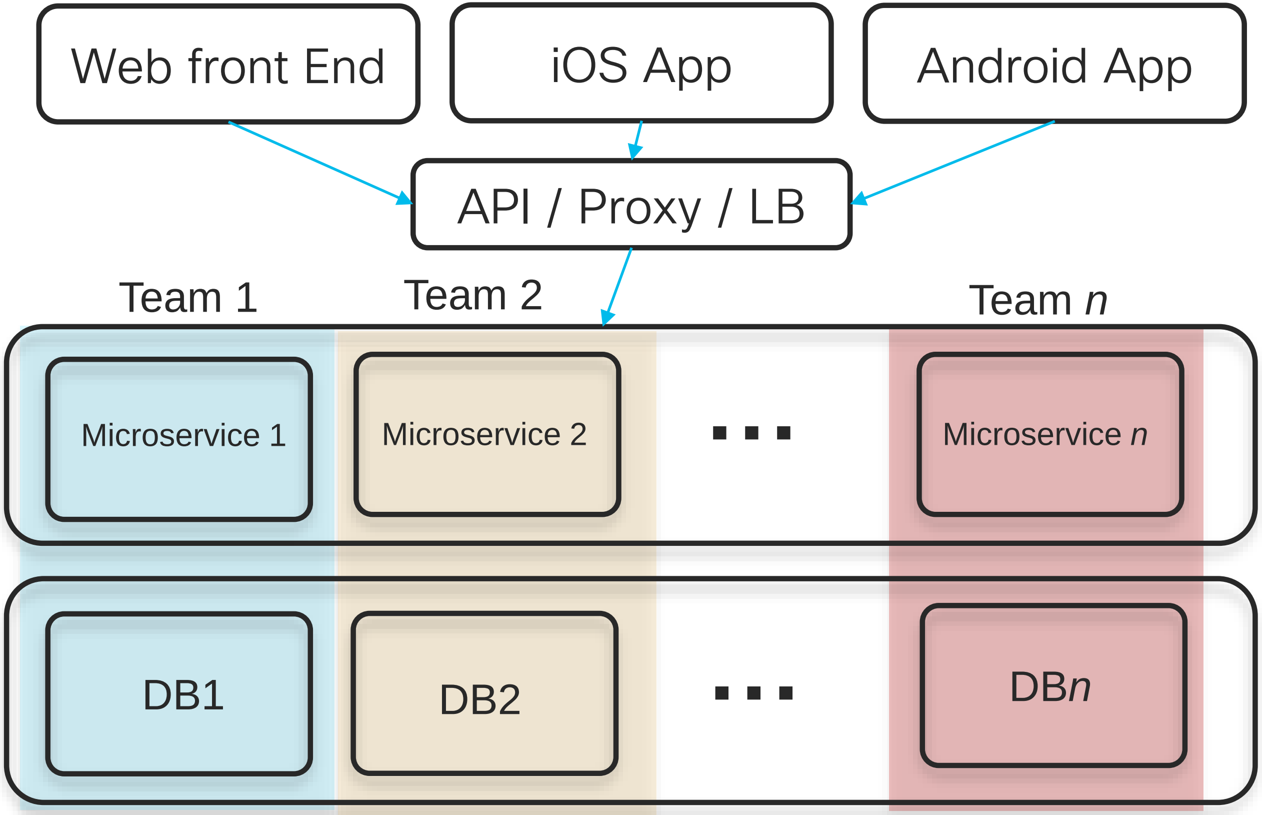
Profile Service

Logging Service

Photo Processing

Friend Requests

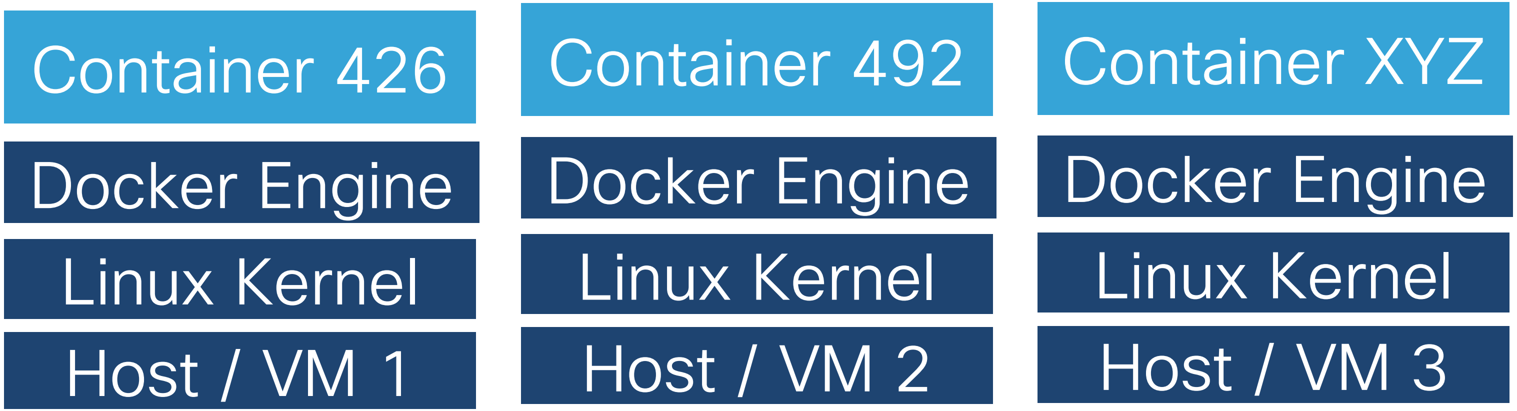
Allow Product Teams to be more productive.



Advantages of Microservices

- Autonomous
  - Microservice can be upgraded independent of other systems
  - Microservice can iterate as quickly as it needs
  - Microservice development doesn’t require understanding the whole applicaation.
- Polyglot application stacks (Technology Heterogenity)
  - Other microservices are black boxes to other services
- Service can be used by other projects in the organization.
  - Or between eachother.

However, lots more apps. Lots more Containers.



Container 426

Docker Engine

Linux Kernel

Host / VM 1

Container 492

Docker Engine

Linux Kernel

Host / VM 2

Container XYZ

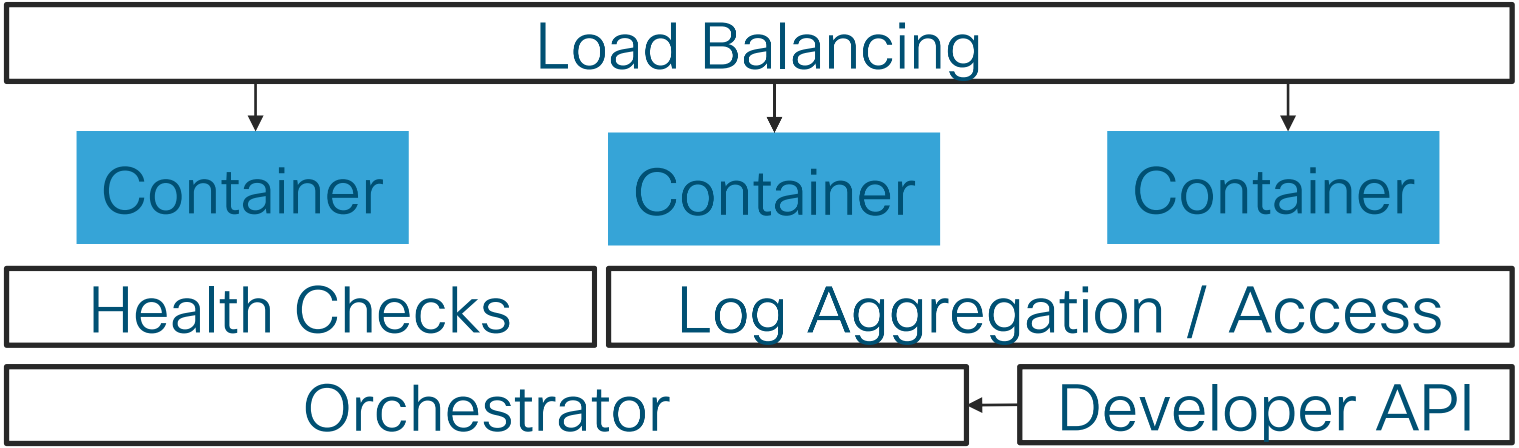
Docker Engine

Linux Kernel

Host / VM 3

Orchestrators

Container Orchestrators manage running containers across a pool of resources for you.



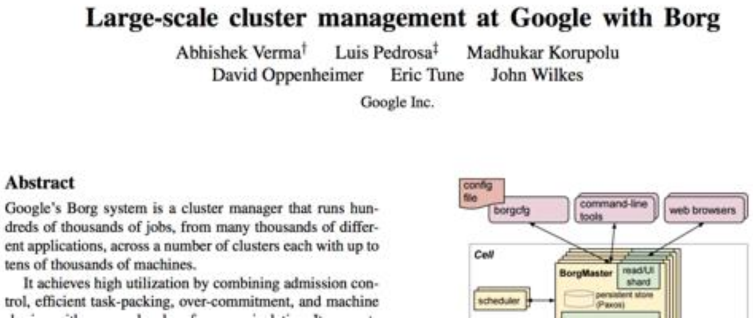
```
$ kubectl scale deployment <name> --replicas=3
```

Kubernetes

What are other orchestrators?

- Docker Swarm / Docker Enterprise Edition (EE)
- Apache Mesos+Marathon
  - (DC/OS)Confusing because it can run Kubernetes!
- Rancher, again, can also run kubernetes.

Borg



Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine

- 2015 paper from Google: https://research.google.com/pubs/pub43438.html
- Engineers who worked on Borg now work on Kubernetes: http://blog.kubernetes.io/2015/04/borg-predecessor-to-kubernetes.html
- Lessons Learned:
  - Multi-Job services could not be managed as a single entity
  - One IP address per Machine

What is Kubernetes?

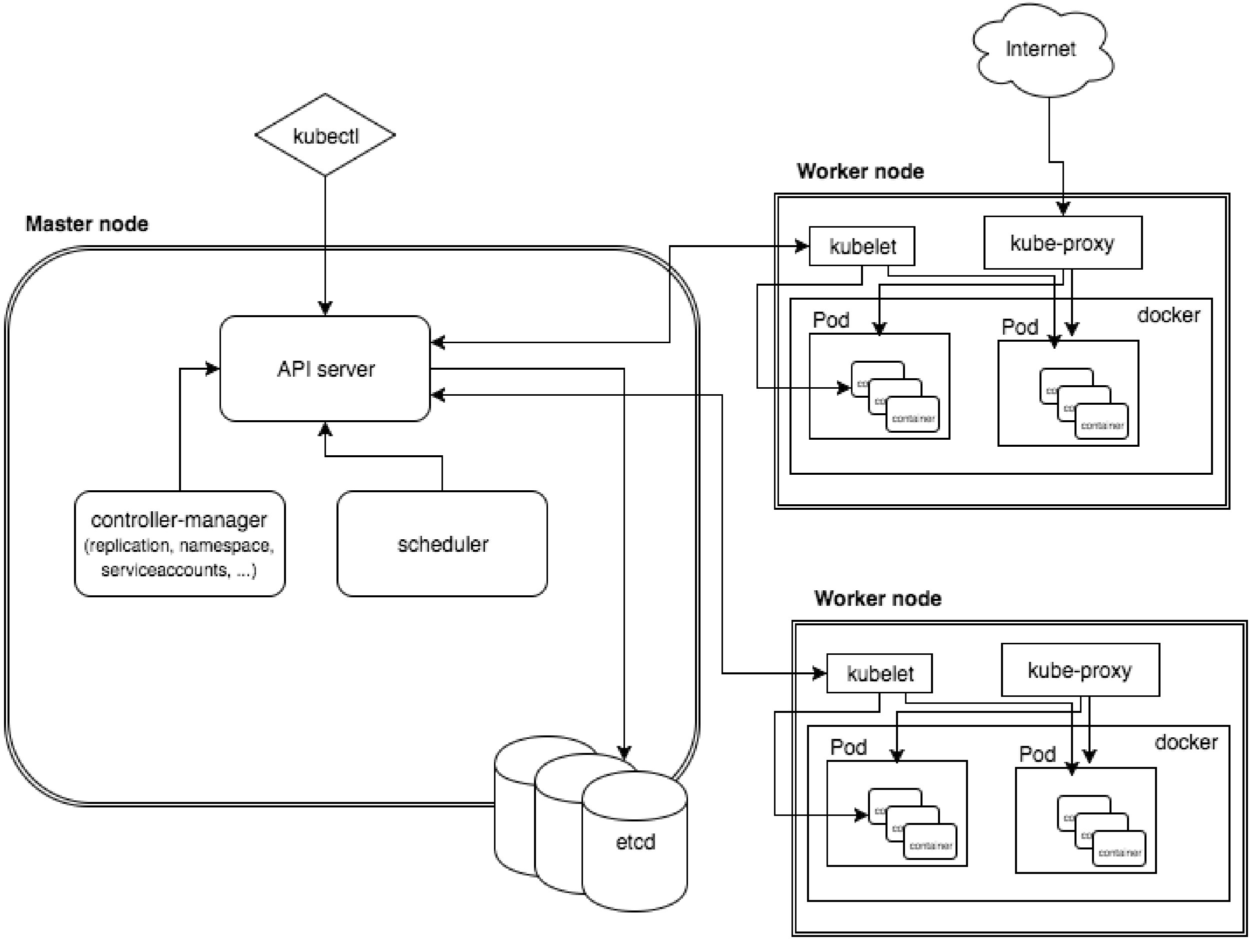
- Container Orchestration
- Keeping your containers up, scaling them, routing traffic to them.
- Kubernetes != Docker
  - It orchestrates containers, that we build.
  - Docker containers are the commonly used example.

Installation options – It’s just code.

- Testing
  - Docker Desktop
  - Play-with-k8s.com
  - MiniKube
- Managed Installs (on-premise)
  - Cisco Container Platform (More later)
- Managed Installs (public cloud)
  - Google Container Engine
  - Azure Container Service
  - Amazon EKS
- DIY / Roll Your Own
  - Kops
  - Kubespray (Ansible + Terraform)

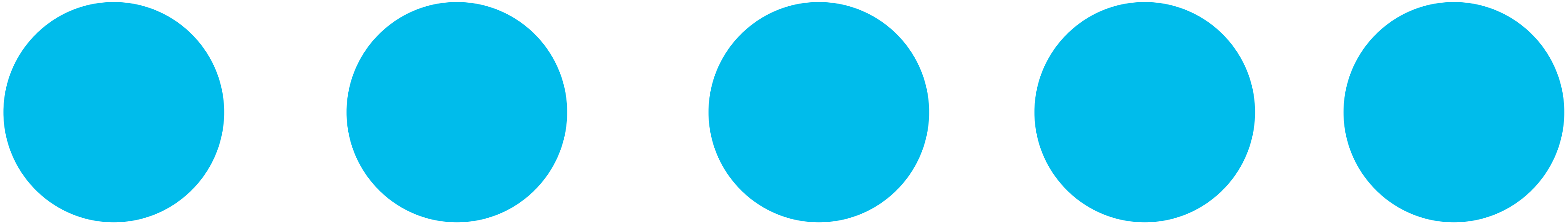
Deep learning: K8S the hard way.

- Step-by-step tutorial of how to assemble a kubernetes cluster
  - If you *WANT* to be in the weeds and how it all fits together.
  - Like I said, it’s just code ☺
  - Aimed at Kubernetes *OPERATORS/DEVELOPERS/DEBUGGERS*
- https://github.com/kelseyhightower/kubernetes-the-hard-way


Internet
kubectl
Master node
API server
controller-manager
(replication, namespace, serviceaccounts, ...)
scheduler
etcd
Worker node
kubelet
kube-proxy
docker
Pod
Pod
container
container
Worker node
kubelet
kube-proxy
docker
Pod
Pod
container
container

Using Kubernetes

Kubernetes Objects
Pods



Pods

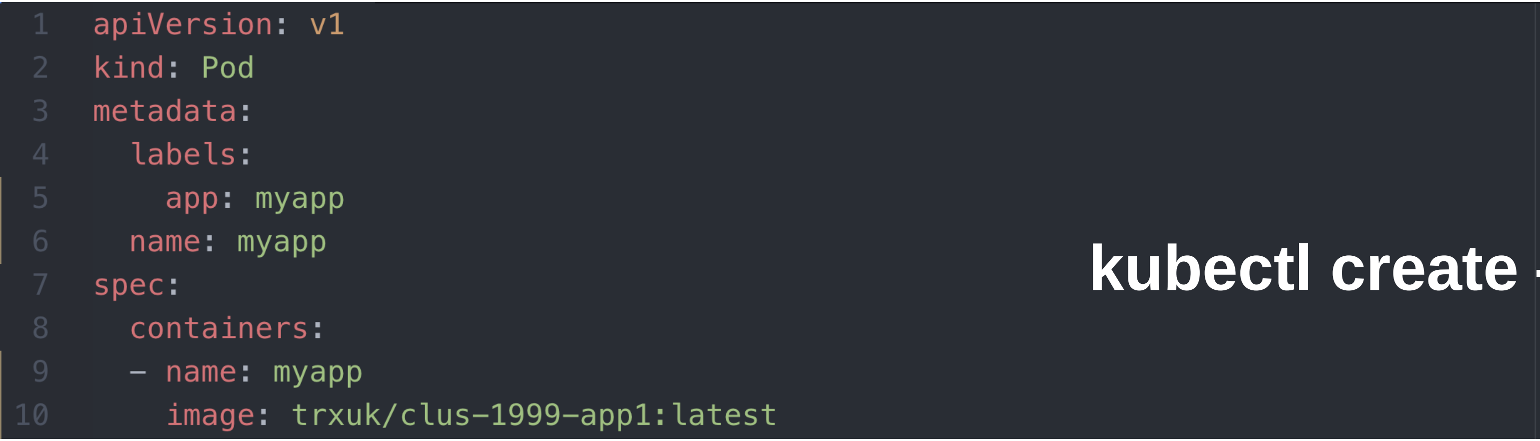
Deployments

Persistence

Services

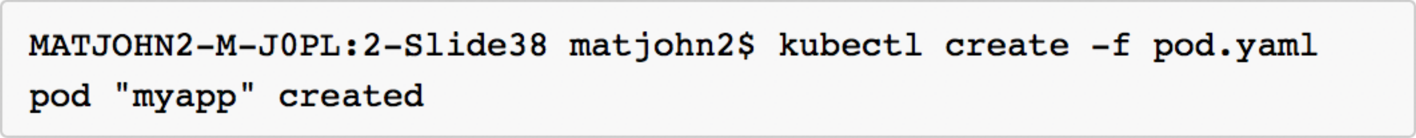
Ingress

One or more containers. “docker run”

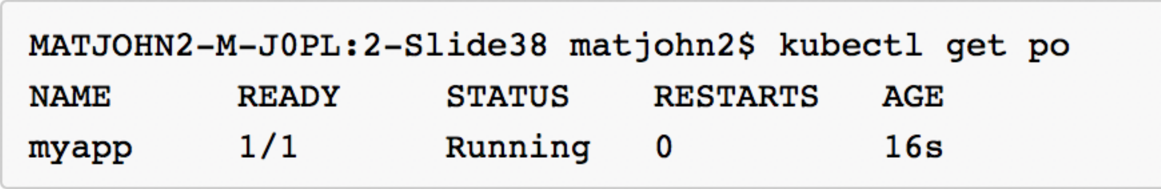


```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: myapp
  name: myapp
spec:
  containers:
  - name: myapp
    image: trxuk/clus-1999-app1:latest
```

kubectl create –f pod.yml

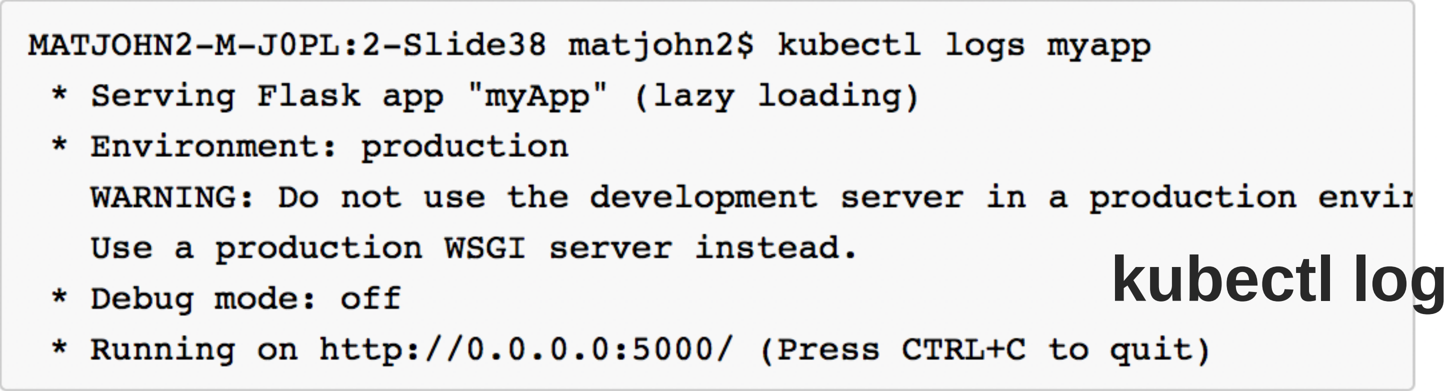


```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl create -f pod.yaml
pod "myapp" created
```



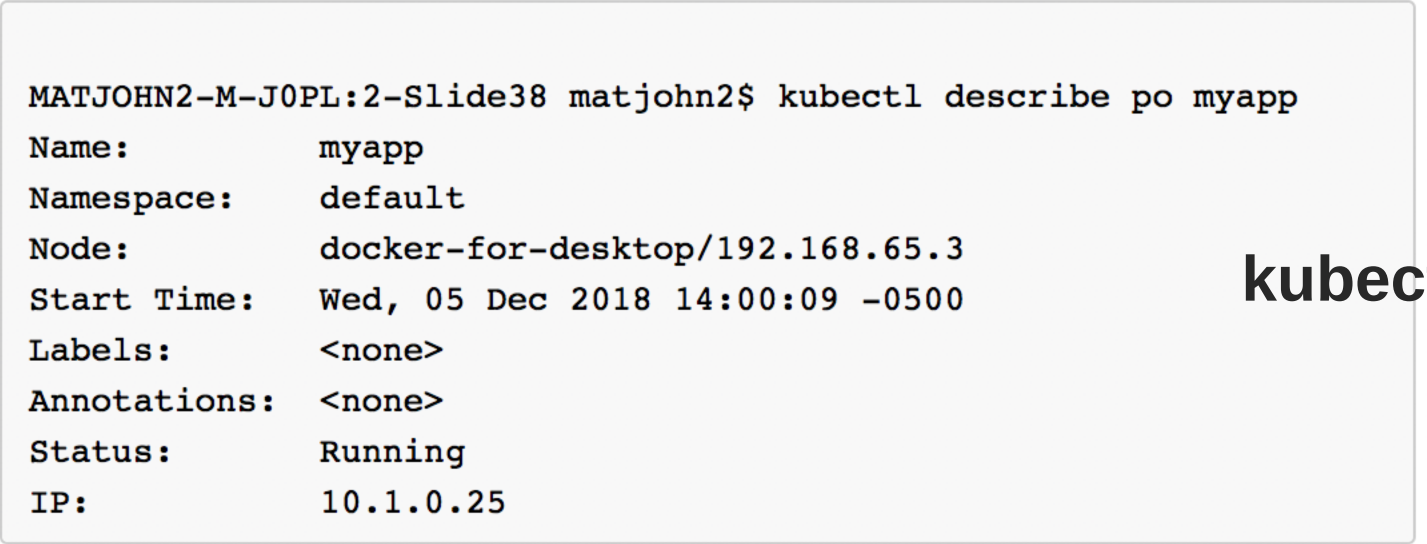
```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl get po
NAME      READY     STATUS    RESTARTS   AGE
myapp     1/1       Running   0          16s
```

kubectl get pod



```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl logs myapp
 * Serving Flask app "myApp" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production envir
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

kubectl logs <podname>

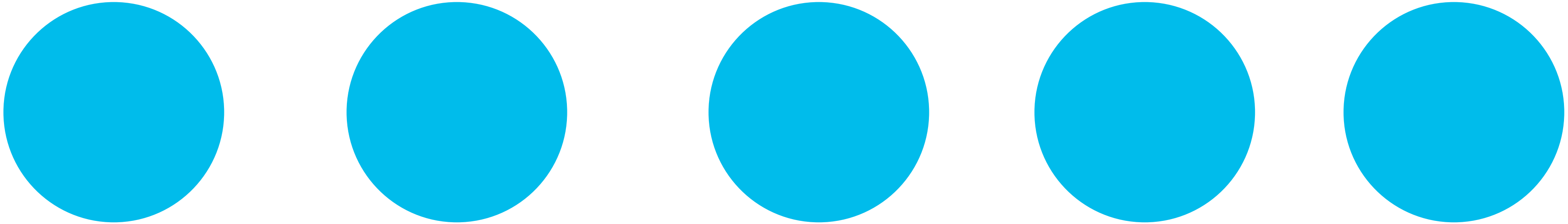


```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl describe po myapp
Name:          myapp
Namespace:     default
Node:          docker-for-desktop/192.168.65.3
Start Time:    Wed, 05 Dec 2018 14:00:09 -0500
Labels:        <none>
Annotations:   <none>
Status:        Running
IP:            10.1.0.25
```

kubectl describe

Kubernetes Objects

Services



Pods

Deployments

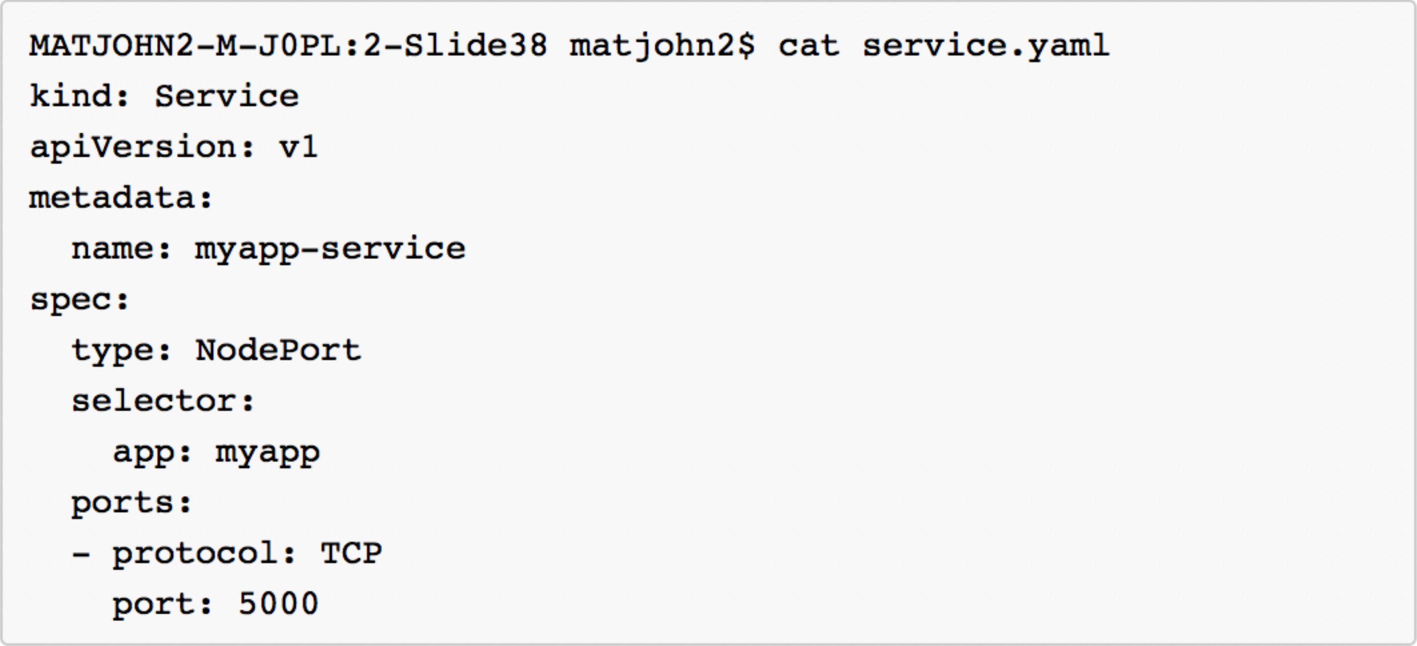
Persistence

Services

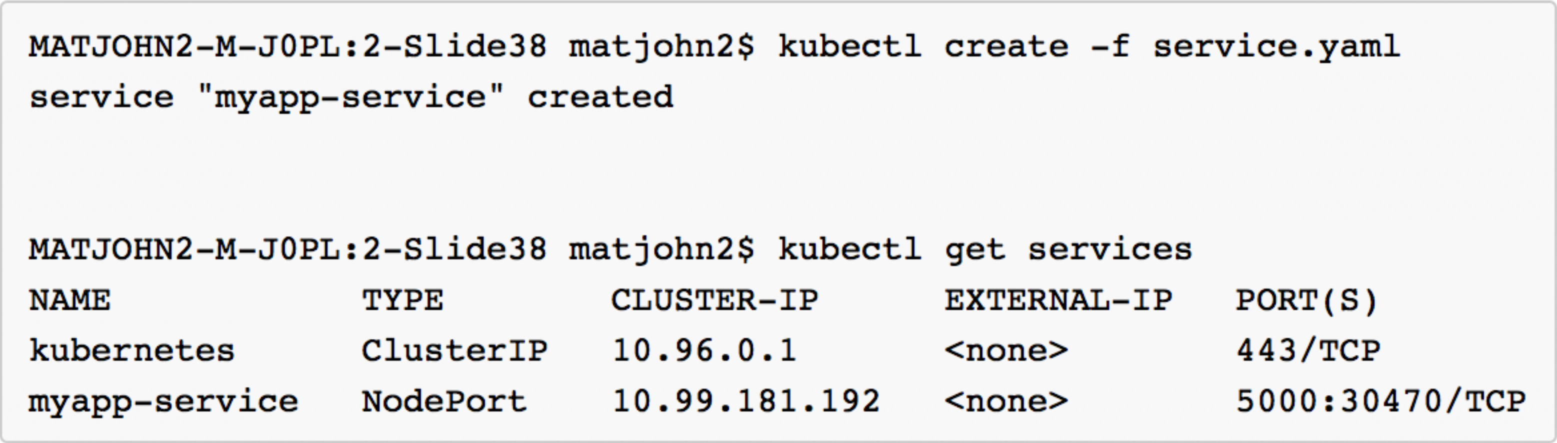
Ingress

One or more containers. “docker run”

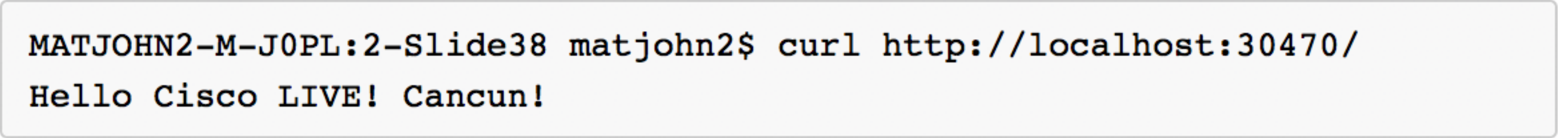
Provide access and Load Balancing



```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ cat service.yaml
kind: Service
apiVersion: v1
metadata:
  name: myapp-service
spec:
  type: NodePort
  selector:
    app: myapp
  ports:
  - protocol: TCP
    port: 5000
```



```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl create -f service.yaml
service "myapp-service" created


MATJOHN2-M-J0PL:2-Slide38 matjohn2$ kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP
myapp-service   NodePort    10.99.181.192   <none>        5000:30470/TCP
```



```
MATJOHN2-M-J0PL:2-Slide38 matjohn2$ curl http://localhost:30470/
Hello Cisco LIVE! Cancun!
```

Kubernetes Objects

Deployments

Pods

One or more containers. “docker run”

Deployments

Provide Replicas and updates.

Persistence

Services

Provide access and Load Balancing

Ingress

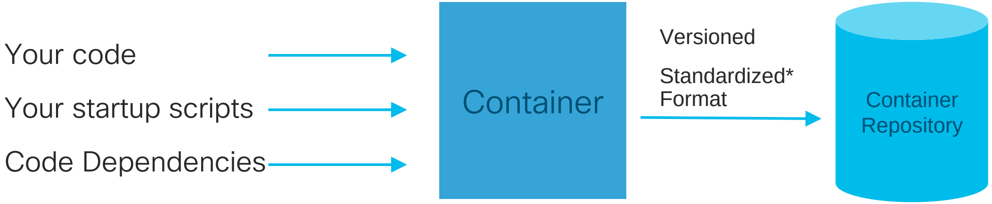
Followups

Deploying Containers

- Manual: Kubectl & ~/.kube/config
- The Real Way™: CI system

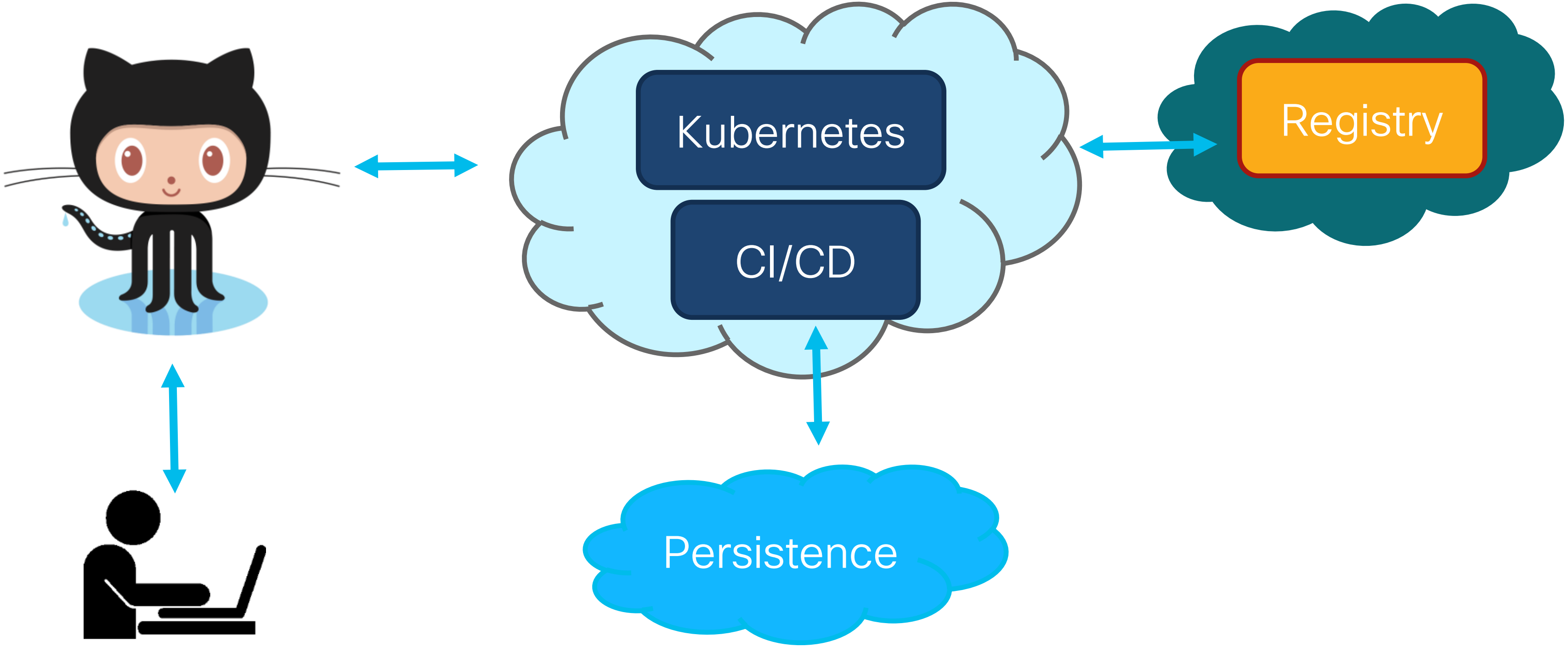
Manually running and waiting for $docker build.
+ Tests
+ Multiple Teams
+ Multiple Microservices

Solution: CI/CD



.... Is going to get boring fairly soon, especially if you build often.

Simple CI/CD Architecture



cisco Live!

https://bit.ly/2rkxFbP

@mattdashj

Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on Thursday) to receive your Cisco Live t-shirt.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

Continue your education

Demos in the
Cisco Showcase

Walk-In Labs

Meet the Engineer
1:1 meetings

Related sessions

Thank you

You make **possible**