Introduction to Containers and Docker

Getting Started With Google Kubernetes Engine

Google Cloud

Version 1.5

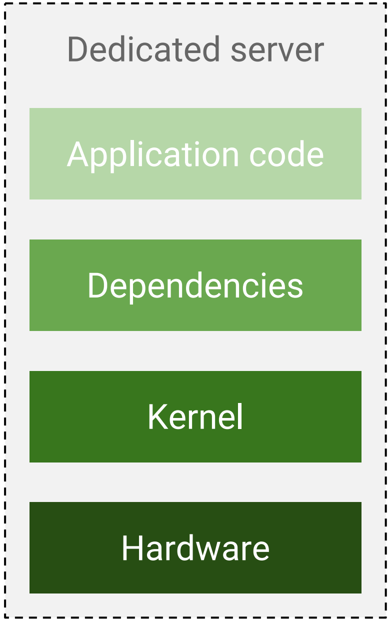
Agenda

Introduction to containers

Introduction to Docker

Lab

Looking back, you used to build applications on individual servers



Dedicated server

Application code

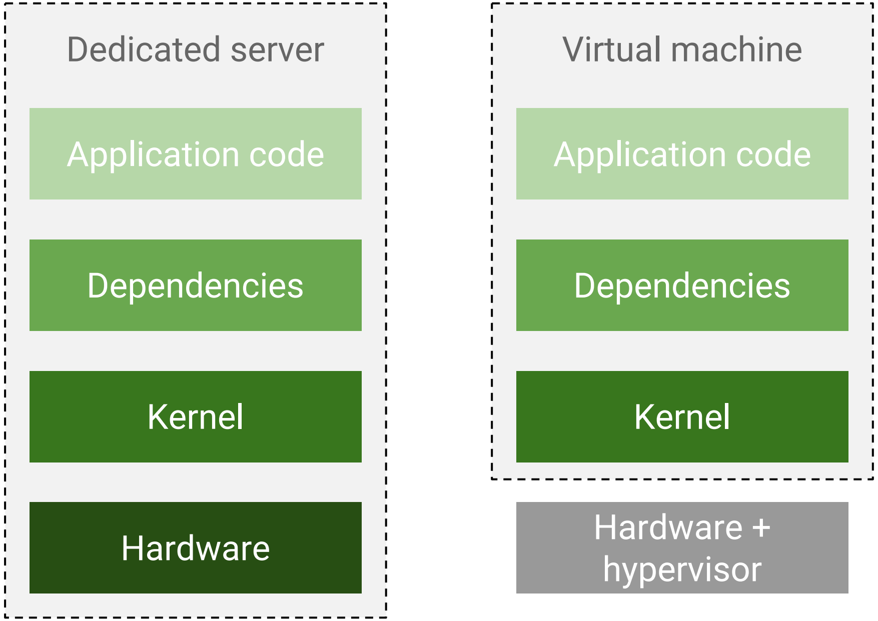
Dependencies

Kernel

Hardware

Deployment ~months
Low utilization
Not portable

Then VMware popularized running multiple servers and operating systems on the same hardware



Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

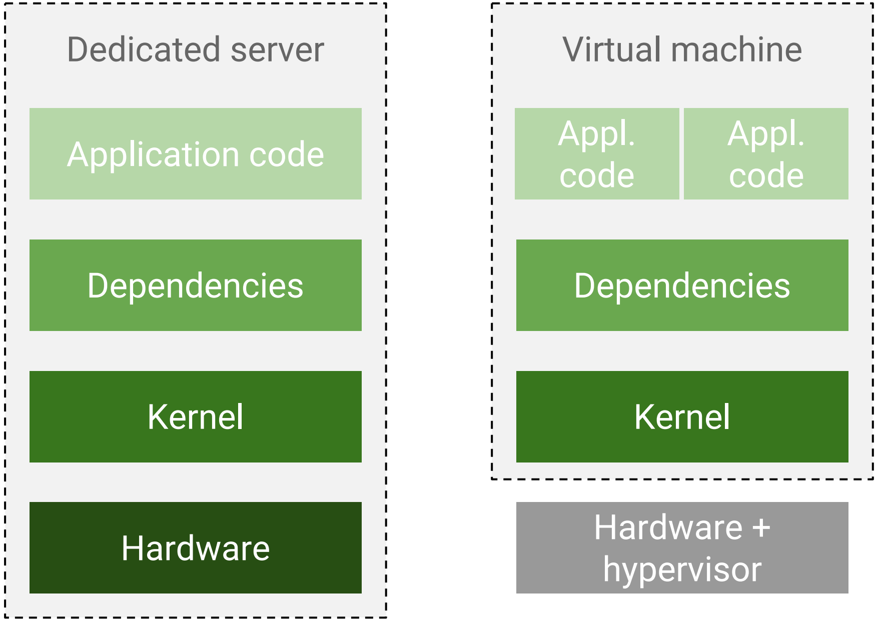
Kernel

Hardware + hypervisor

Deployment ~months
Low utilization
Not portable

Deployment ~days (mins)
Improved utilization
Hypervisor-specific

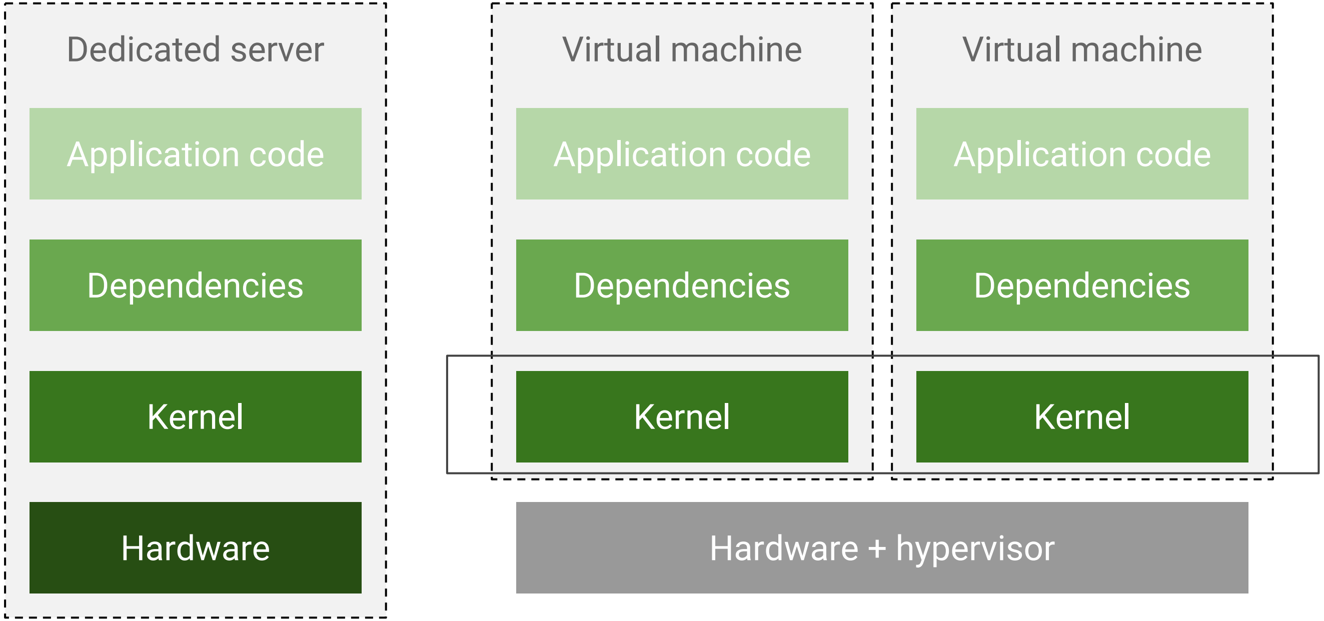
But it was difficult to run and maintain multiple applications on a single VM, even with policies



Deployment ~months
Low utilization
Not portable

Deployment ~days (mins)
Hypervisor-specific
Low isolation; tied to OS

The VM-centric way to solve this is to run each app on its own server with its own dependencies, but that's wasteful



Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

Kernel

Virtual machine

Application code

Dependencies

Kernel

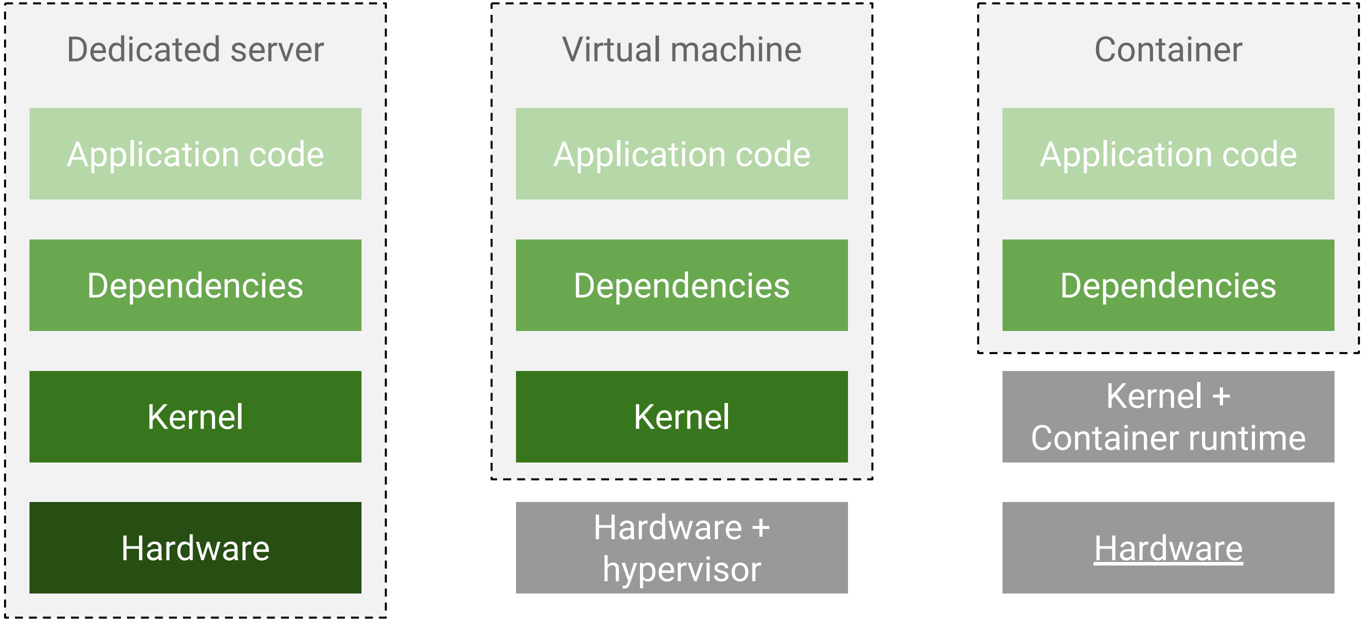
Hardware + hypervisor

Deployment ~months
Not portable
Low utilization

Deployment ~days (mins)
Hypervisor-specific
Low isolation; tied to OS

Deployment ~days (mins)
Hypervisor-specific
Redundant OS

So you raise the abstraction one more level and virtualize the OS



Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

Kernel

Hardware + hypervisor

Container

Application code

Dependencies

Kernel + Container runtime

Hardware

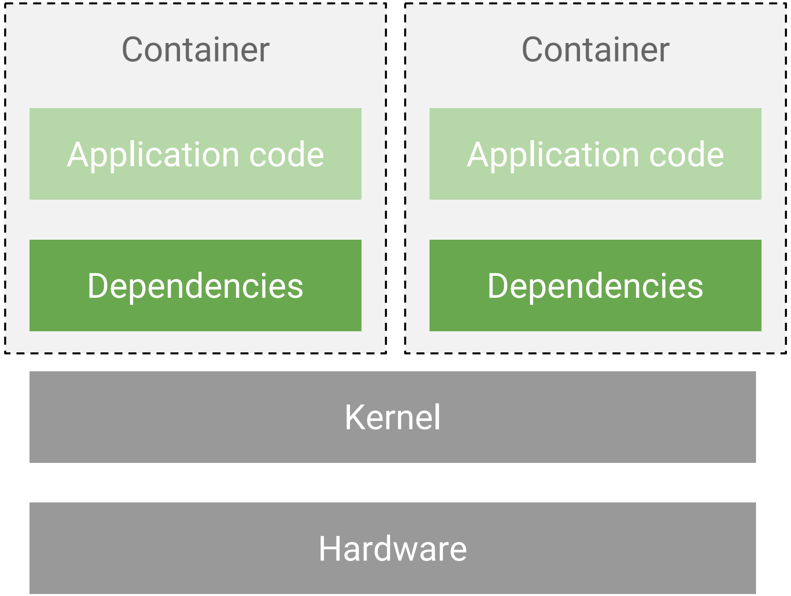
Deployment ~months
Not portable
Low utilization

Deployment ~days (mins)
Hypervisor-specific
Low isolation, Tied to OS

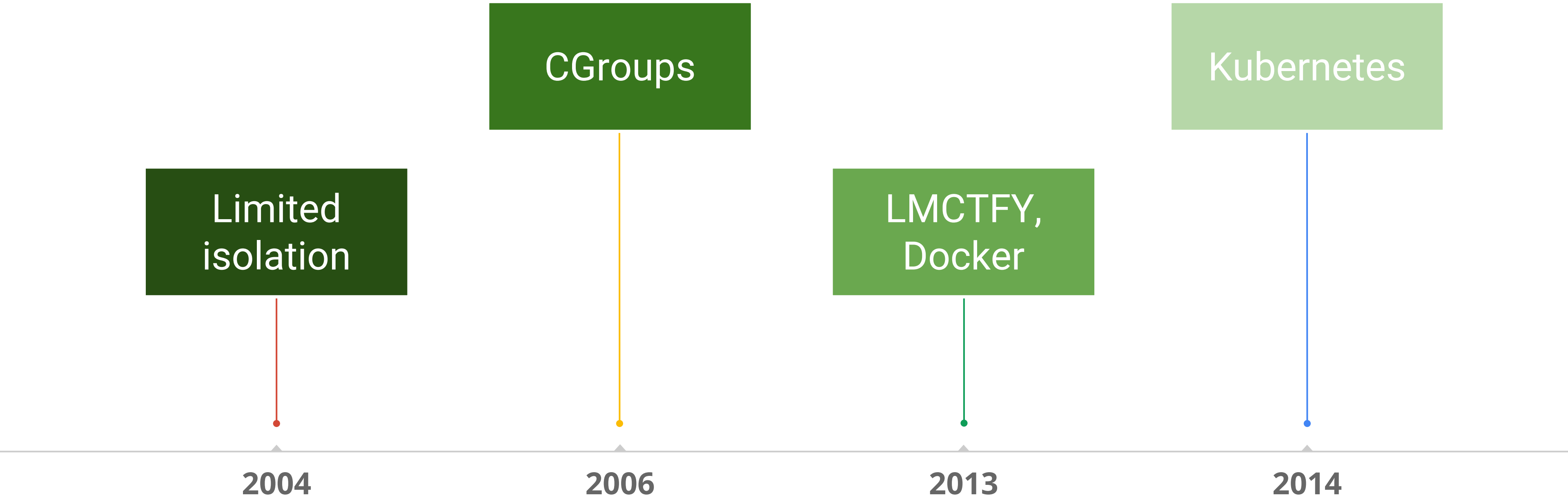
Deployment ~mins (sec)
Portable
Very efficient

Why developers like containers

- Code works the same everywhere:
  - Across dev, test, and production
  - Across bare-metal, VMs, and cloud
- Packaged apps speed development:
  - Agile creation and deployment
  - Continuous integration/delivery
  - Single file copy
- They provide a path to microservices:
  - Introspectable, isolated, and elastic



Google has been developing and using containers to manage its applications for 12 years



Limited isolation

2004

CGroups

2006

LMCTFY, Docker

2013

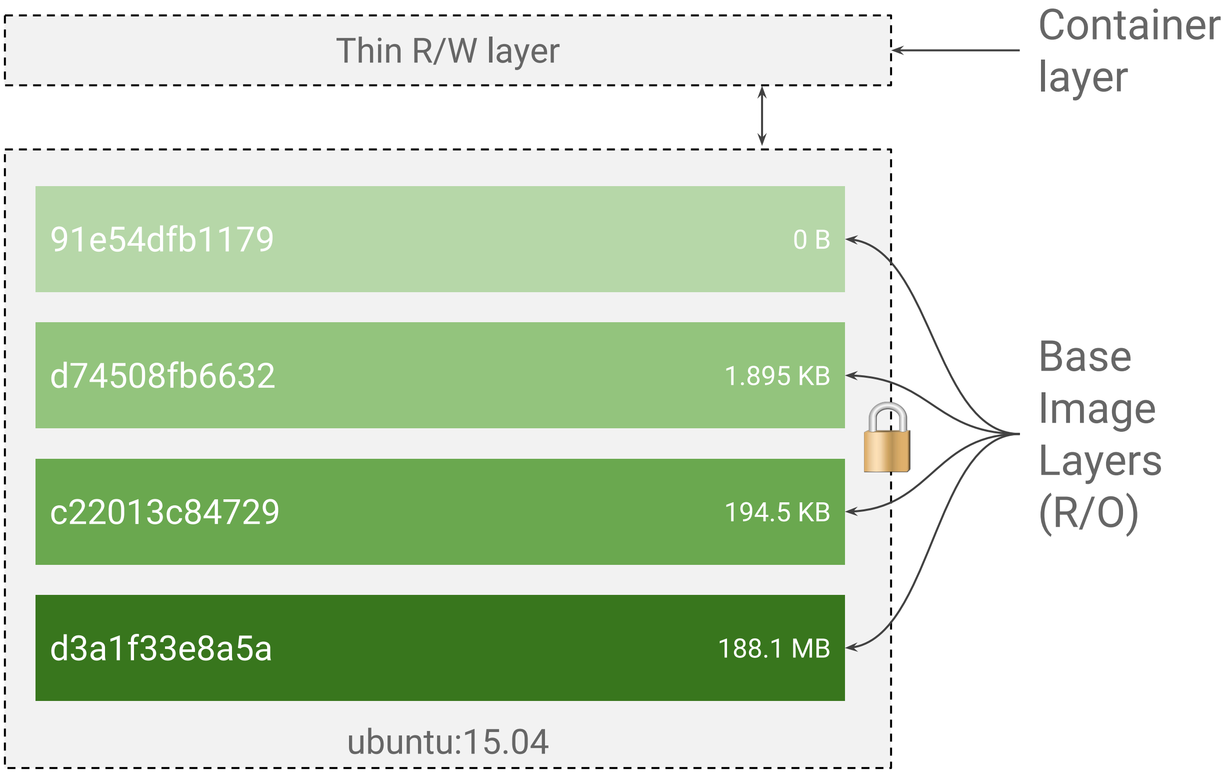
Kubernetes

2014

Containers use a layered file system with only the top layer writable

Dockerfile

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Thin R/W layer

Container layer

91e54dfb1179 0 B

d74508fb6632 1.895 KB

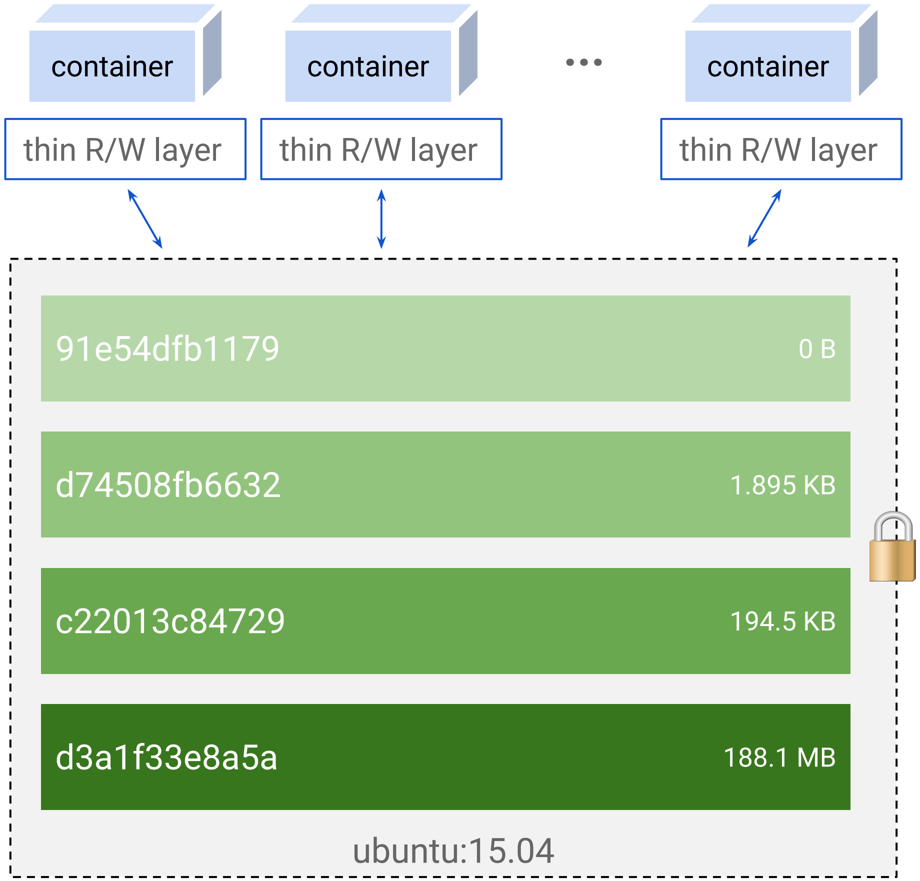
c22013c84729 194.5 KB

d3a1f33e8a5a 188.1 MB

ubuntu:15.04

Base Image Layers (R/O)

Containers promote smaller shared images



container

container

...

container

thin R/W layer

thin R/W layer

thin R/W layer

91e54dfb1179 0 B

d74508fb6632 1.895 KB

c22013c84729 194.5 KB

d3a1f33e8a5a 188.1 MB

ubuntu:15.04

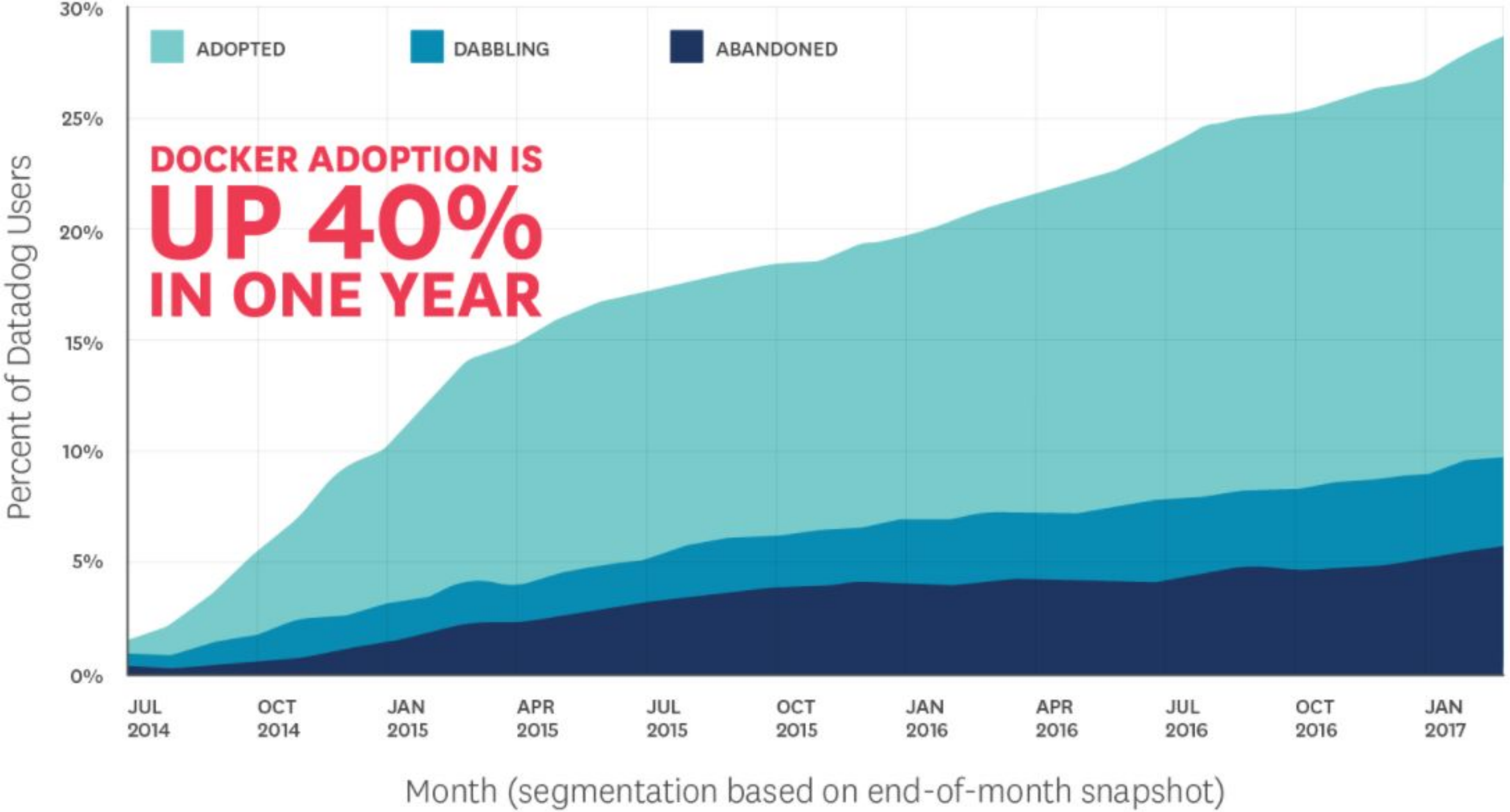
Agenda

Introduction to containers

Introduction to Docker

Lab

Docker Adoption Behavior



DOCKER ADOPTION IS UP 40% IN ONE YEAR

Source: Datadog

Here's is a simple python app

```
$> python web-server.py
```

```
import tornado.ioloop
import tornado.web
import socket

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hostname: " +
socket.gethostname())

def make_app():
    return tornado.web.Application([
      (r"/", MainHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8080)
    tornado.ioloop.IOLoop.current().start()
```

dependencies

listening on a port

Containerize it with Docker

```
$> docker build -t py-web-server .

$> docker run -d py-web-server
```

```
FROM library/python:3.6.0-alpine
RUN pip install tornado
ADD web-server.py /web-server.py
CMD ["python", "/web-server.py"]
```

```
You can also do stuff like:
$> docker images
$> docker ps
$> docker logs <container id>
$> docker stop py-web-server
```

In the real world you'll push and pull your image from a registry

```
docker build -t gcr.io/$PROJECT_ID/py-web-server:v1 .
```

build a container image

```
gcloud docker -- push gcr.io/$PROJECT_ID/py-web-server:v1
```

push it to a registry

```
docker run -d -p 8080:8080 --name py-web-server  \
gcr.io/$PROJECT_ID/py-web-server:v1
```

run it

Containers are the new packaging format because they're efficient and portable

- App Engine supports Docker containers as a custom runtime
- Google Container Registry: private container image hosting on GCS with various CI/CD integrations
- Compute Engine supports containers, including managed instance groups with Docker containers
- The most powerful choice is a container **orchestrator**

© photohome.co.uk

Lab