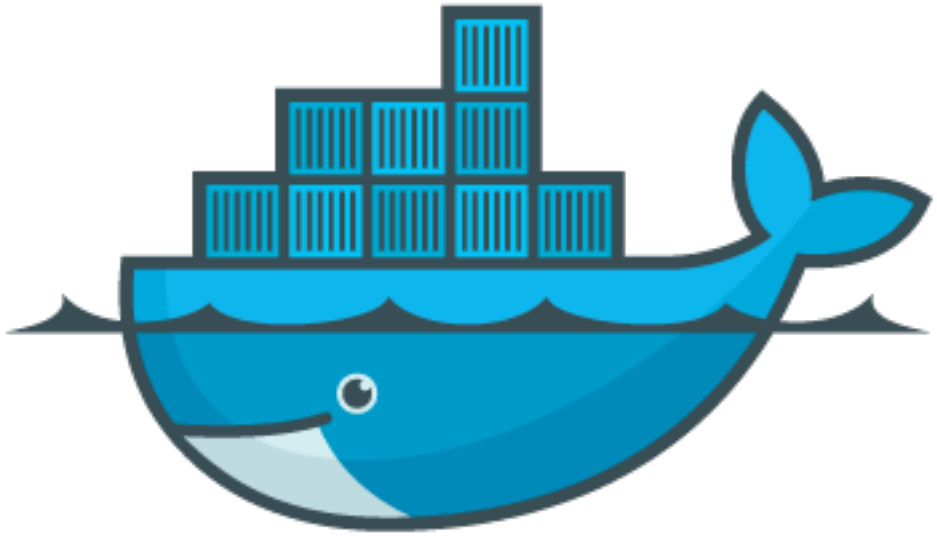
DEVOPS PART 5: IMMUTABLE SERVERS

MANISH KUMAR THAKUR
MTHAKUR@XEBIA.COM

Xebia INSPIRE CHANGE ACT ON RESULTS

DOCKER



WHAT'S THE PROBLEM WITH VM'S?

Within our private PaaS approach we like to keep things simple by creating a virtual machine per appliance

- Good for:
  - managebility
  - operability
  - resource management
- But simplicity comes at a price...
  - License cost
  - Resources overhead / performance

ENTER DOCKER

- You get all the goodies of virtual machine per appliance, but without the cost.
  - Filesystem isolation
  - Resource isolation
  - Network isolation

- And it is fast!

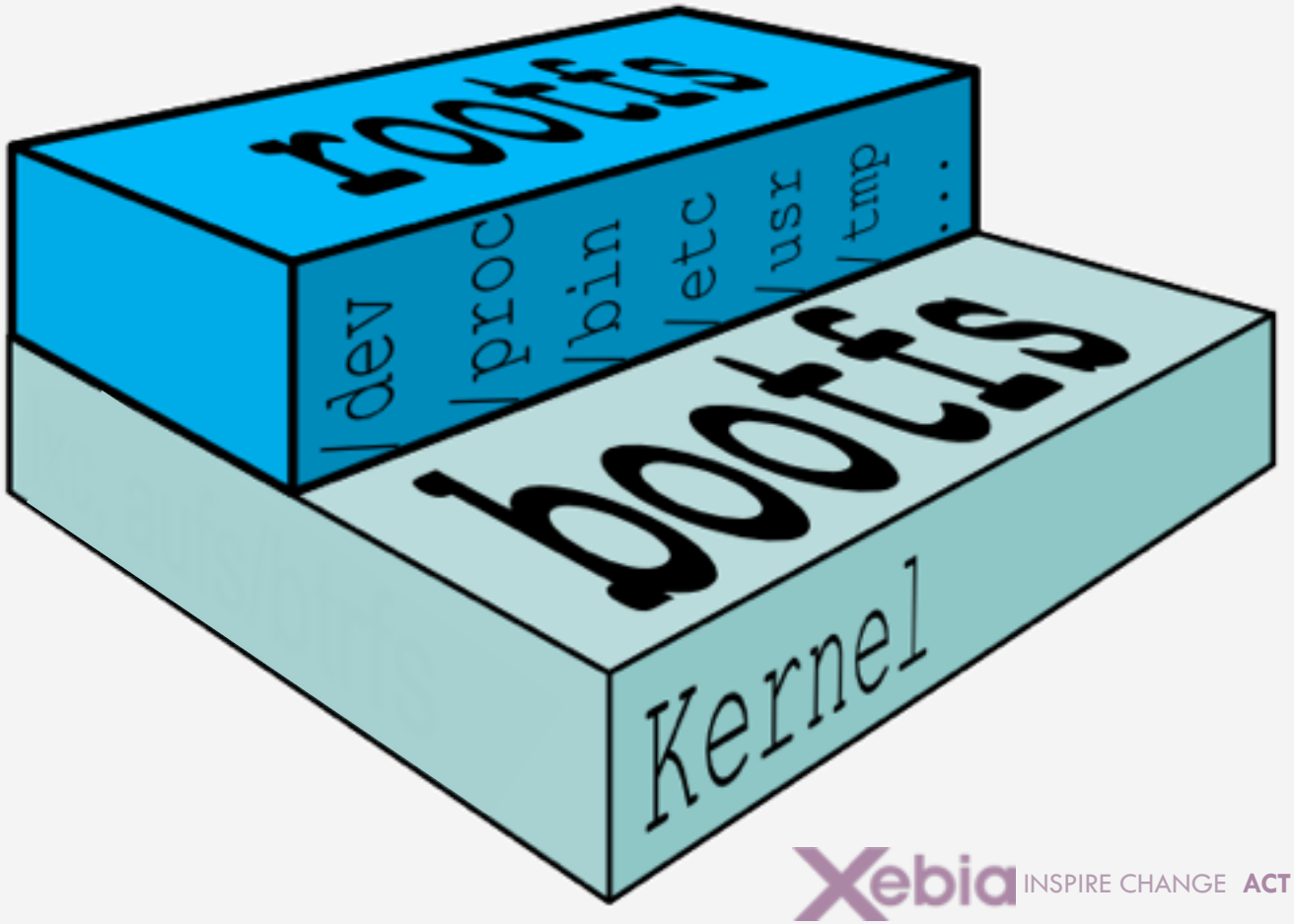
WHAT IS DOCKER?

Lightweight application containers offered by Linux

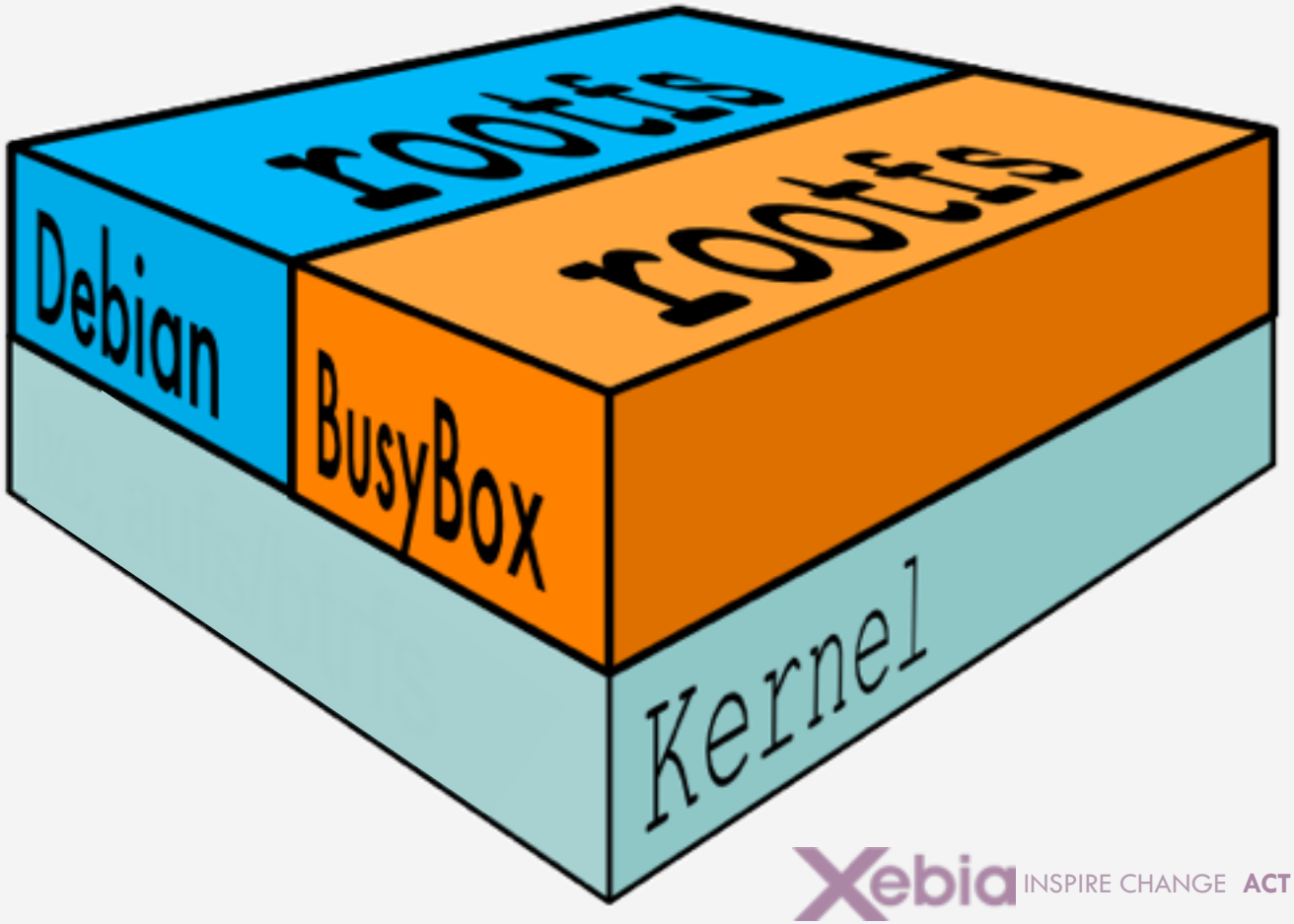
PORTABLE IMAGES

- Images contain everything needed to run your application
- When instantiated, it runs exactly one primary process (from which you could spawn many more)
- Images are portable across daemons
- Images are built in layers

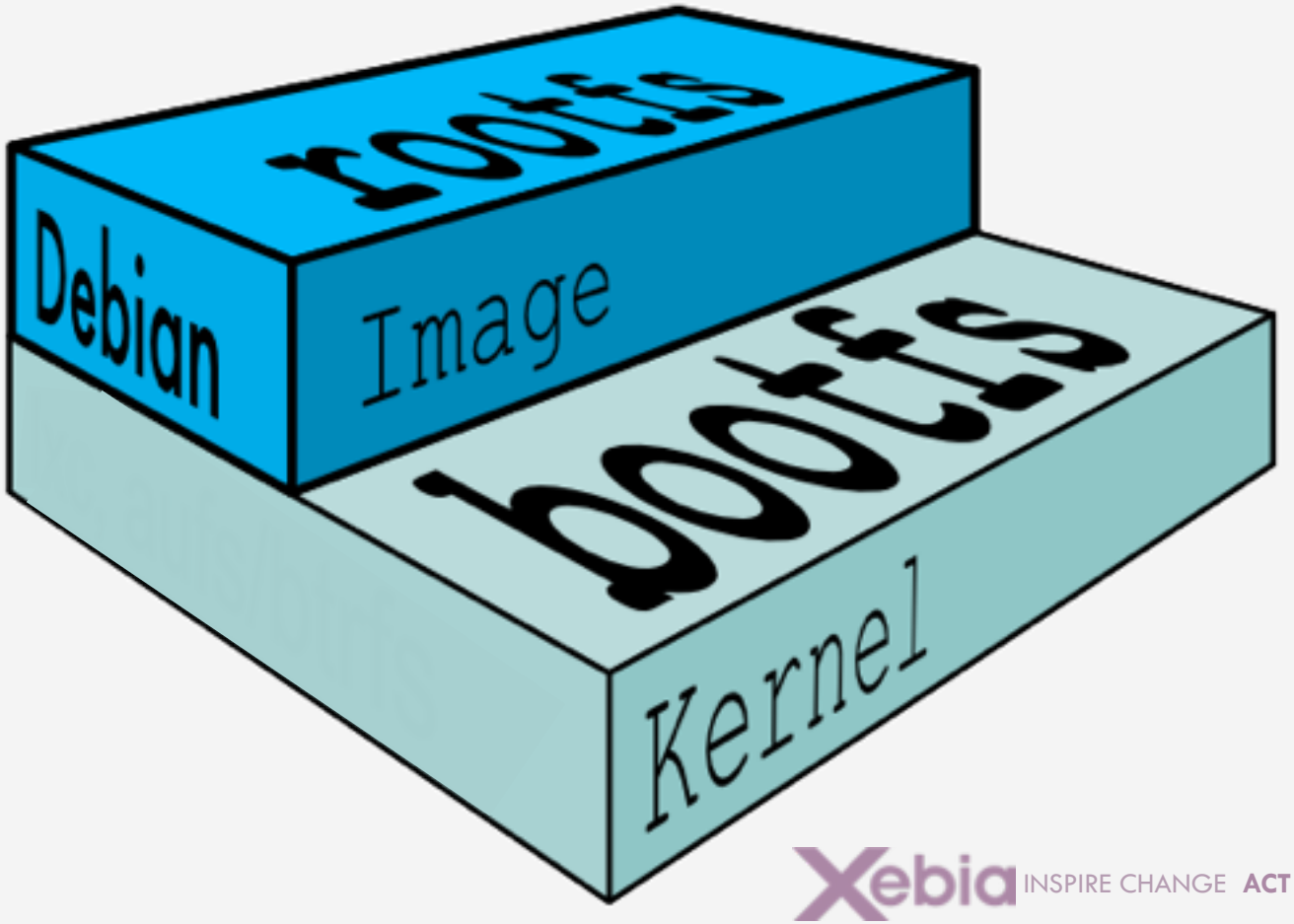
Docker Requires Two File Systems

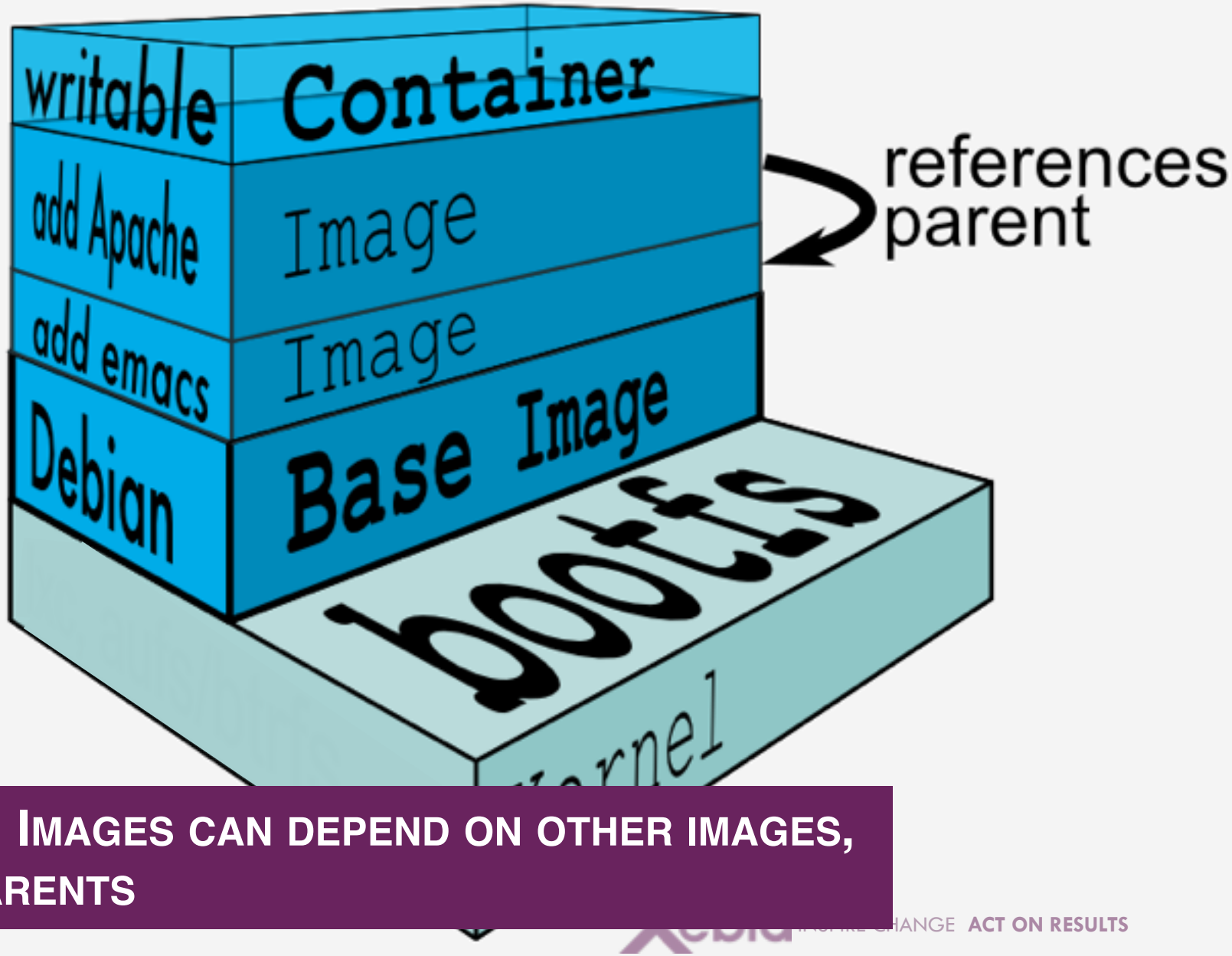


You Can Have Multiple rootfs

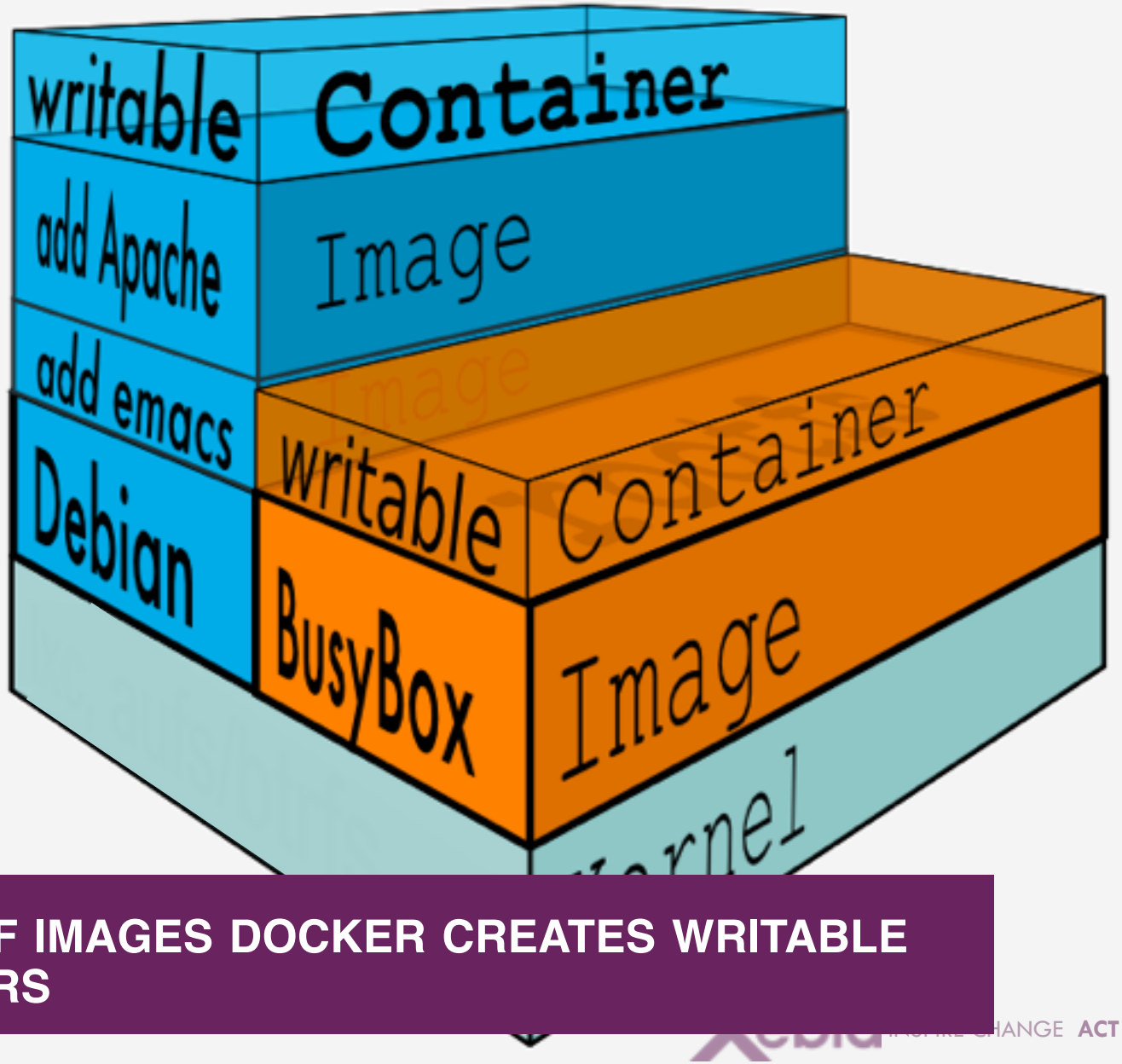


Read-only layers are called images





STACKING: IMAGES CAN DEPEND ON OTHER IMAGES, CALLED PARENTS



On top of images docker creates writable containers

Getting Started

```
docker run ubuntu:14.04 /bin/echo "hello world"
```

Interactive Containers

```
docker run -ti ubuntu:14.04 bash
```

AND TRY SOME COMMANDS…

```
# look around all your processes
ps -ef

# Checkout your file system
ls

# and your network
ifconfig

# logout (container is stopped as /bin/bash exits)
exit
```

“THOU SHALT NOT SSH INTO DOCKER CONTAINERS”

Detached Containers

```
# run -d means detached detach
DOCKER_ID=$(docker run -d ubuntu:14.04 \
   bash -c 'while true ; \
        do sleep 1; \
        echo hello world at $(date); \
        done' )
echo $DOCKER_ID            # shows id of container
docker attach $DOCKER_ID   # attach to stdout of the container
docker ps                     # shows all running containers
docker stop $DOCKER_ID     # stops specified container
docker ps -a                  # shows stopped and running containers
docker rm $DOCKER_ID       # removes the container
```

CREATING A PYTHON IMAGE

Create a Dockerfile

```
FROM    ubuntu:14.04

RUN     apt-get -y install python
ADD     index.html /var/index.html
WORKDIR /var
EXPOSE  8085
CMD python -m SimpleHTTPServer 8085
```

CREATING A PYTHON IMAGE

Create an index.html

```
<html>
  <body>
    <h2>Hello World from Python this time</h2>
   </body>
</html>
```

CREATING A PYTHON IMAGE

Build your Docker image

```
docker build -t adejonge/python-web .
```

Run your Docker image

```
docker run --name python-web -p 8085:8085 adejonge/
   python-web
```

Immutable Servers

"An Immutable Server is [...] a server that once deployed, is never modified, merely replaced with a new updated instance"

Kief Morris @ martinfowler.com

ANTONYMS

- Snowflake Server
- Phoenix Server

The deliverable of the DevOps/NoOps team is a fully installed and configured server image.

How do you configure an image without modifying it?


CONSUL

“Consul "is a tool for discovering and configuring services in your infrastructure"

consul.io

Consul Features

- Service discovery
- Health checking
- Key value store
- Multi-datacenter

CONSUL WORKS WITH

- Peer to peer networking
- Gossip protocol (Serf)
- An agent per node
- A DNS interface (compatibility)
- A REST interface (rich API)

CONSUL-BASE/DOCKERFILE

```
FROM debian:wheezy
WORKDIR /opt
ENV PATH /opt:$PATH
RUN apt-get update
RUN apt-get -y install unzip wget curl dnsutils procps
RUN wget --no-check-certificate https://dl.bintray.com/
   mitchellh/consul/0.5.2_linux_amd64.zip
RUN unzip 0.5.2_linux_amd64.zip
RUN rm 0.5.2_linux_amd64.zip
```

BUILD & RUN THE IMAGES

```
docker build -t xebia/consul-base consul-base
docker run -ti xebia/consul-base bash
```

START CONSUL (MANUALLY)

```
consul agent -server -bootstrap-expect 1 -data-
  dir /tmp/consul > /var/consul.log &
```

Check that Consul is running

```
ps
consul members
ip addr
```

Save the IP address for a later step

CREATE A 2ND CONSUL CONTAINER

```
docker run -ti xebia/consul-base bash
consul agent -data-dir /tmp/consul > /var/
   consul.log &
```

JOIN CLUSTER

```
consul join {IP OF FIRST IMAGE}
consul members
```

You have just created your first, small consul cluster

- Your cluster consists of one server and one client
- In production, it is recommended to have 3 or 5 servers
- In the next step, we are going to replace the server

CONFIGURE DNS WITH CONSUL

CLEAN UP PREVIOUS CONTAINERS

```
docker rm -f {container-id or name}
```

Configure DNS for Consul

Add config/dns.json:

```
{
    "recursor": "8.8.8.8",
    "ports": {
        "dns": 53
    }
}
```

Create a new Dockerfile

```
FROM xebia/consul-base
ADD config /opt/config/
```

BUILD YOUR IMAGE AND TAG IT AS XEBIA/CONSUL-DNS

START THE DOCKER IMAGE AND CHECK THAT THE DNS DOES NOT WORK YET

```
docker run -ti  --dns 127.0.0.1 -h myhost
  xebia/consul-dns bash
ping google.com
cat /etc/resolv.conf
```

START CONSUL WITH THE NEW CONFIGURATION: -CONFIG-DIR /OPT/CONFIG/ AND CHECK AGAIN:

```
consul agent -server -bootstrap-expect 1 -config-
   dir /opt/config/ -data-dir /tmp/consul > /var/
   consul.log &
ping google.com
```

This server we will use as central point throughout the rest of the workshop so please save the IP address again

NOW TRY INTERNAL NODES

```
dig myhost.node.consul
ping myhost.node.consul
```

CONFIGURE SERVICE DEFINITION

CREATE A NEW IMAGE THAT ADDS SERVICE.JSON TO /CONFIG DIRECTORY:

```
{
    "service": {
        "name": "fruit",
        "tags": ["master"],
        "port": 8080
    }
}
```

CONFIGURE SERVICE DEFINITION

This time we add the command to startup Consul to avoid typing the same thing over and over

```
FROM xebia/consul-dns
ADD config /opt/config/
CMD /opt/consul agent -data-dir /tmp/consul -
   config-dir /opt/config/ -dc xebia -client 0.0.0.0
   -bind 0.0.0.0 > /var/consul.log & bash
```

NOTE

- In this workshop we are running Consul in each and every container. This will help us play around with Consul. However, you may prefer a setup with a single, central Consul container per Docker host once you start using this setup in larger environments.

BUILD AND RUN YOUR SERVICE CONTAINER, CHECK THAT THE SERVICE IS RUNNING AND JOIN THE CLUSTER

```
docker build -t xebia/consul-service consul-service
docker run -ti --dns 127.0.0.1 -h mysvc xebia/
   consul-service
ps
consul join {IP OF SERVER CONTAINER}
```

USE SERVICE:

```
dig fruit.service.consul
```

USE TAG:

```
dig master.fruit.service.consul
```

Try adding one more container to the cluster and dig again

```
docker run -ti --dns 127.0.0.1 xebia/consul-service
consul join {IP OF SERVER CONTAINER}
dig fruit.service.consul
consul leave
```

ADD AN APP THAT OFFERS THE SERVICE

```
#!/usr/bin/python

PORT_NUMBER = 8085

class myHandler(BaseHTTPRequestHandler):

   def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.end_headers()
        self.wfile.write("Hello World from Docker
   image!")
        return

Try:
    server = HTTPServer(('', PORT_NUMBER), myHandler)

    server.serve_forever()

except KeyboardInterrupt:
    print '^C received, shutting down the web server'
    server.socket.close()
```

AND CHECK IF IT WORKS

```
curl fruit.service.consul:8085
```

PACKER

Xebia INSPIRE CHANGE **ACT ON RESULTS**

PACKER INTRODUCTION

"Packer is a tool for creating identical machine images for multiple platforms from a single source configuration."

– packer.io

PACKER FEATURES

- A single template creates images for multiple platforms
- Use (existing) configuration management
- Parallel image creation

PACKER WORKS WITH

- Packer template file (json)
- Builders (Docker, AWS, etc)
- Provisioners (Shell script, Salt, etc)

Install Packer

```
wget https://dl.bintray.com/mitchellh/packer/
  packer_0.8.2_linux_amd64.zip
unzip packer_0.8.2_linux_amd64.zip -d /usr/local/bin/
```

CHECK IF IT WORKS

```
packer --version
```

STEP 1 - CREATE PACKER TEMPLATE

example.json

```
{
  "builders": [{
    "type": "docker",
    "image": ”ubuntu:14.04",
    "export_path": "example.tar"
  }]
}
```

STEP 2 - BUILD IMAGE

Validate your template

```
packer validate example.json
```

Build your image

```
packer build example.json
```

Import the image into Docker

```
cat example.tar | docker import - repo:example
```

or

```
docker import - repo:example < example.tar
```

MAKE YOUR IMAGE DO SOMETHING

fruit.json

```
{
  "builders": [
  {
    "type": "docker",
    "image": "debian:wheezy",
    "export_path": "fruit.tar"
  }],
  "provisioners": [
  {
    "type": "shell",
    "inline": [
    "echo orange > /opt/fruit.txt"
    ]
  }]
}
```

Import the image into Docker

```
cat example.tar | docker import - repo:example
```

or

```
docker import - repo:example < example.tar
```

Import the image into Docker

```
cat example.tar | docker import - repo:example
```

or

```
docker import - repo:example < example.tar
```

End of Part 5