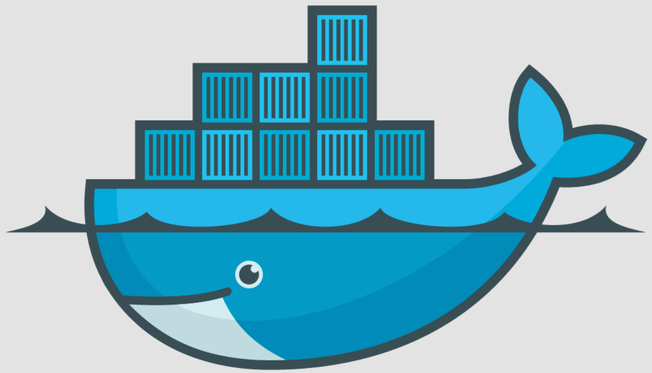
Docker and CoreOS

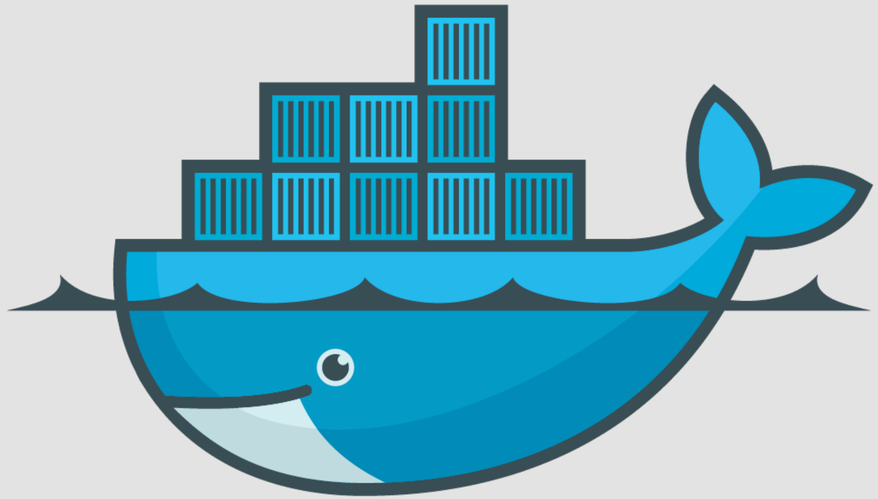
A bit of what why and how

Hi I’m Jim!



docker

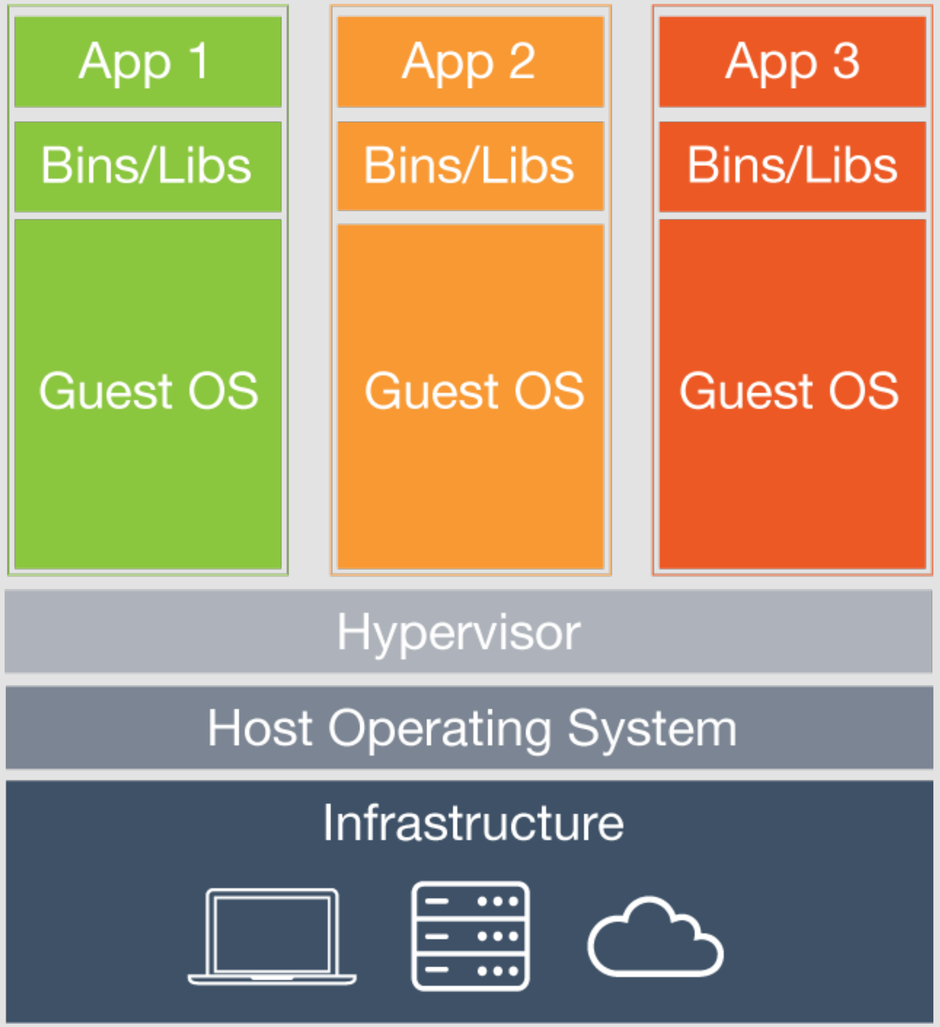
CoreOS



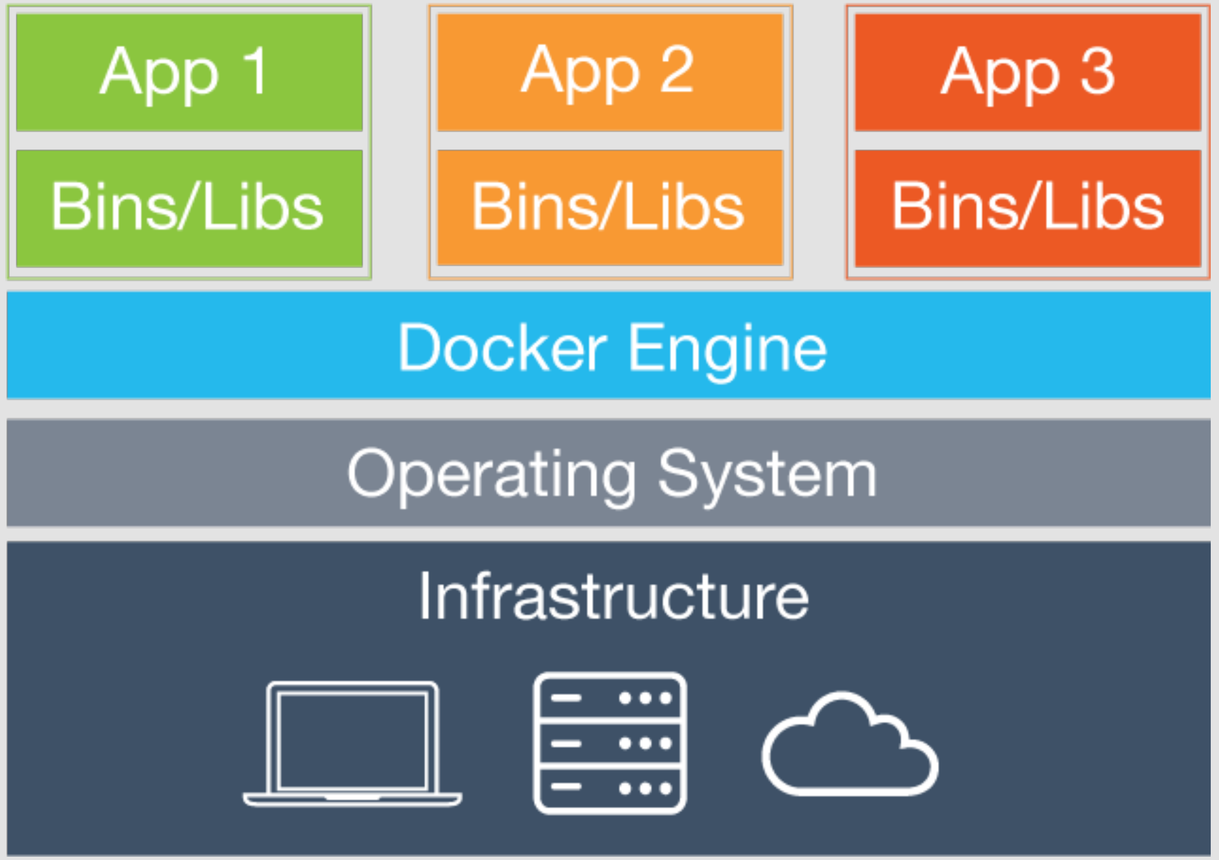
docker


YO DAWG WE HEARD YOU LIKED DOCKER
SO WE PUT A DOCKER IN YOUR DOCKER
SO YOU CAN DOCKER
WHILE YOU DOCKER

VMs


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure

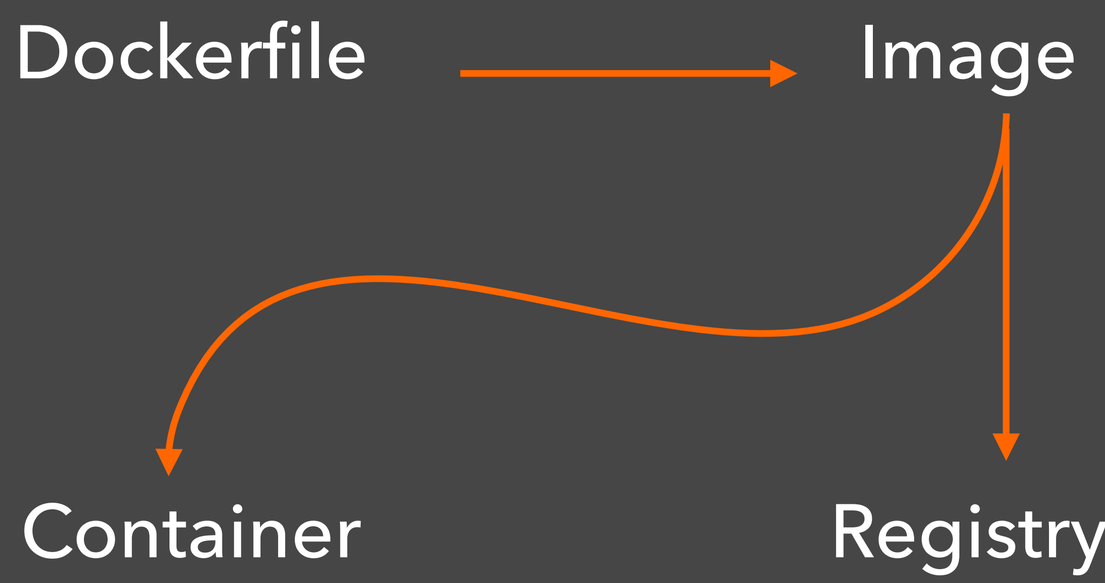
Containers


App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure

Some Docker Terminology

- A Dockerfile is a text document that contains all the commands to assemble an image
- An Image is a shareable snapshot of software
- A Container is a running instance of an image
- A Registry is a repo of docker images

How these parts relate



Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

Dockerfile

```
FROM golang:latest
MAINTAINER Jim Weber

RUN mkdir -p /go/src/valet/conf

COPY *.go /go/src/valet/
COPY ./conf/* /go/src/valet/conf/

RUN go get github.com/pkg/profile
RUN cd /go/src/valet ; go build

CMD /go/src/valet/valet -P 8000

EXPOSE 8000
```

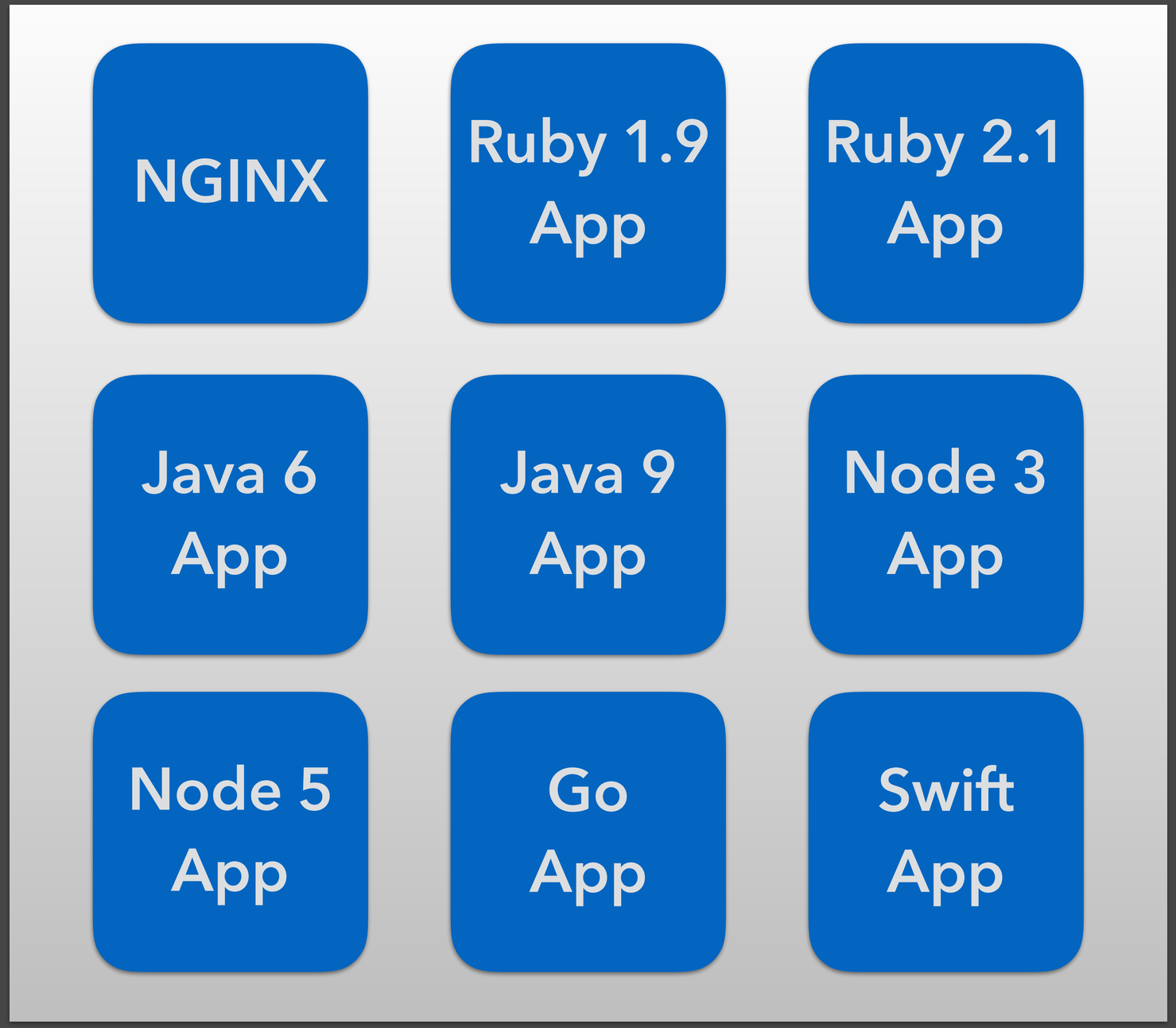
Let’s build an image

Ok ... But how do I use this in practice?


NGINX
Ruby 1.9 App
Ruby 2.1 App
Java 6 App
Java 9 App
Node 3 App
Node 5 App
Go App
Swift App

I’m a Server running docker
I don’t need

- compilers
- language run times
- special Libraries


NGINX
Ruby 1.9 App
Ruby 2.1 App
Java 6 App
Java 9 App
Node 3 App
Node 5 App
Go App
Swift App

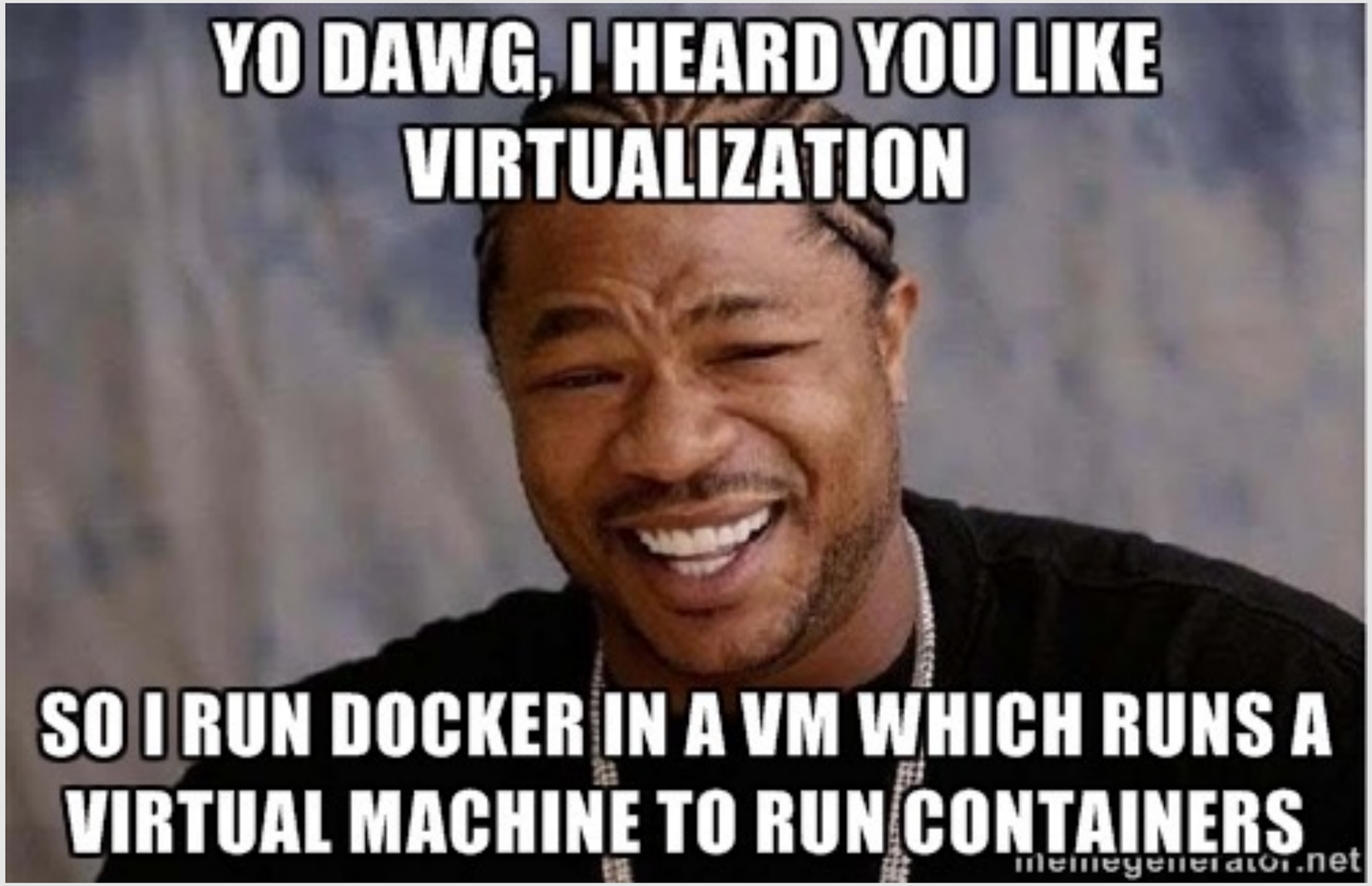
Do one thing
and, do it well

Do one thing and, do it well

It is the UNIX philosphy too


Whoa, this is heavy.

But Jim, How do I use docker in an HA Cluster?


YO DAWG, I HEARD YOU LIKE VIRTUALIZATION
SO I RUN DOCKER IN A VM WHICH RUNS A VIRTUAL MACHINE TO RUN CONTAINERS
memegenerator.net

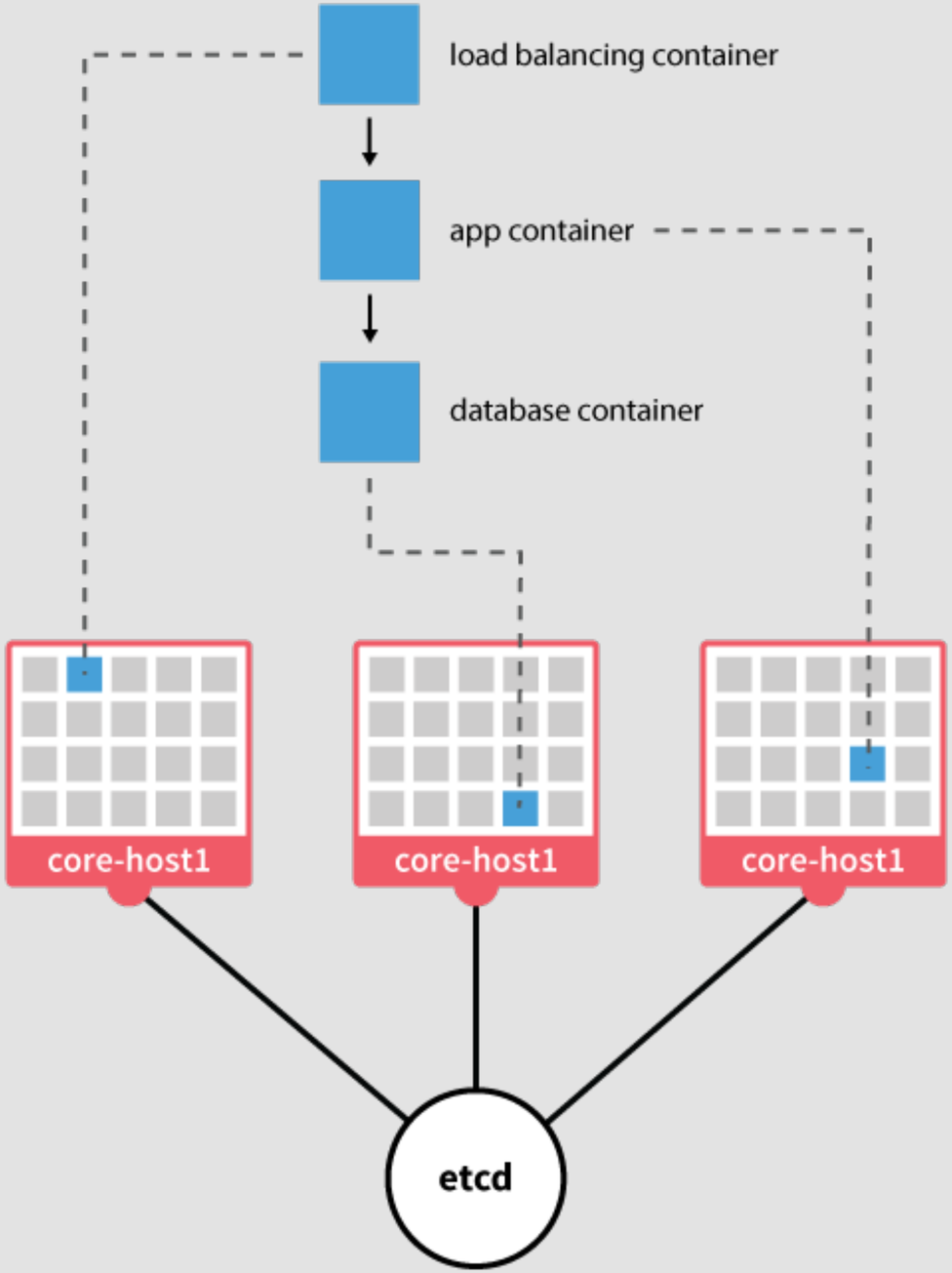
Ok, not really

CoreOS

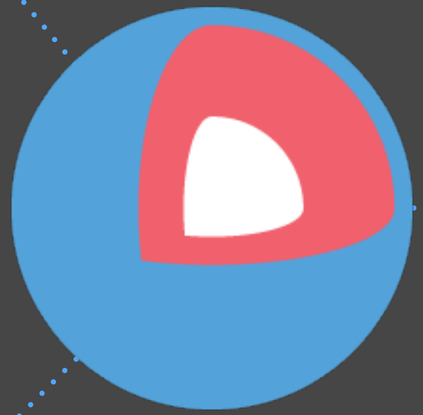
CoreOS

CoreOS is a powerful Linux distribution built to make large, scalable deployments on varied infrastructure simple to manage

A Linux distro made specifically for running containers in a clustered environment.
With some extra tools to make that work.


load balancing container
app container
database container
core-host1
core-host1
core-host1
etcd

confd



etcd

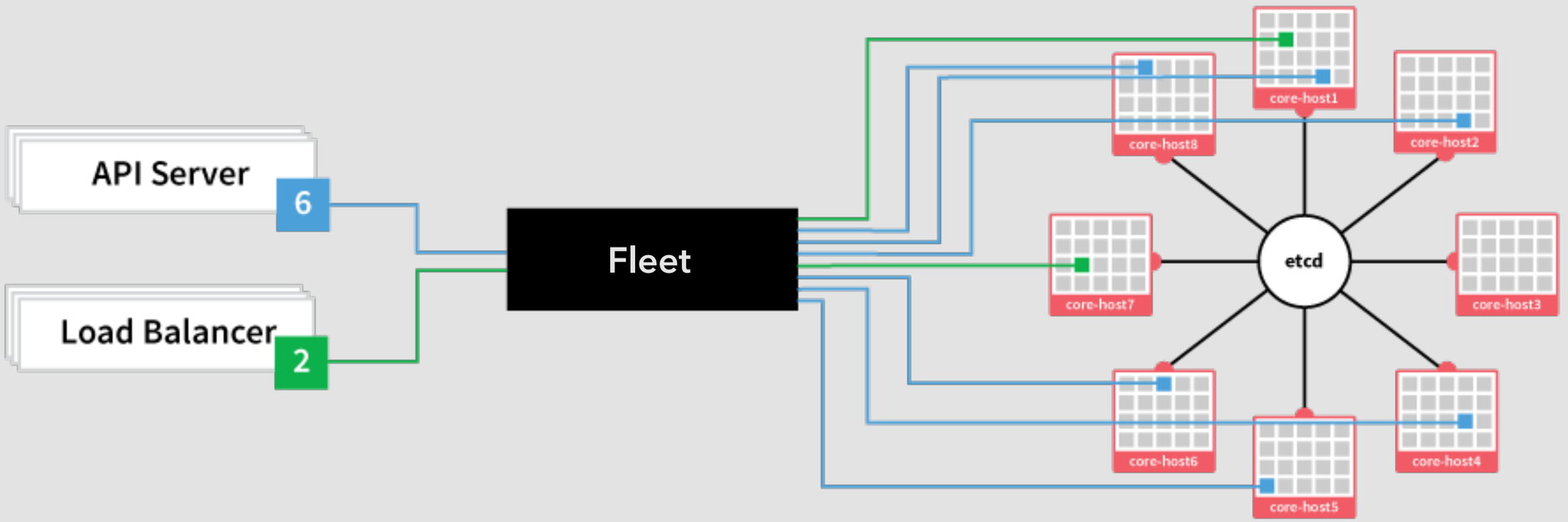
fleet

Etcd is a distributed, consistent key-value store for shared configuration and service discovery

- Curl'able user-facing API (HTTP+JSON)
- Optional SSL client cert authentication
- Benchmarked 1000s of writes/s per instance
- Distributed using Raft consensus algorithm

Fleet ties together systemd and **etcd** into a simple distributed init system.

Think of it as systemd that operates at the cluster level instead of the machine level.


API Server
6
Load Balancer
2
Fleet
core-host1
core-host8
core-host2
core-host7
etcd
core-host3
core-host6
core-host4
core-host5

Confd - Manage local application configuration files using templates and data from etcd or consul

This

```
upstream sweetapp_pool {
    {{ range getvs "/services/sweetapp/*" }}
        server {{ . }};
    {{ end }}
}

  …

 location /sweetapp/ {

  proxy_pass http://sweetapp/api;

  …

 }
```

Becomes This

```
upstream sweetapp_pool {
        server 10.253.156.164:8811;
        server 10.253.156.165:8810;
}

  ...

 location /sweetapp/ {

  proxy_pass http://sweetapp/api;

  …

 }
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

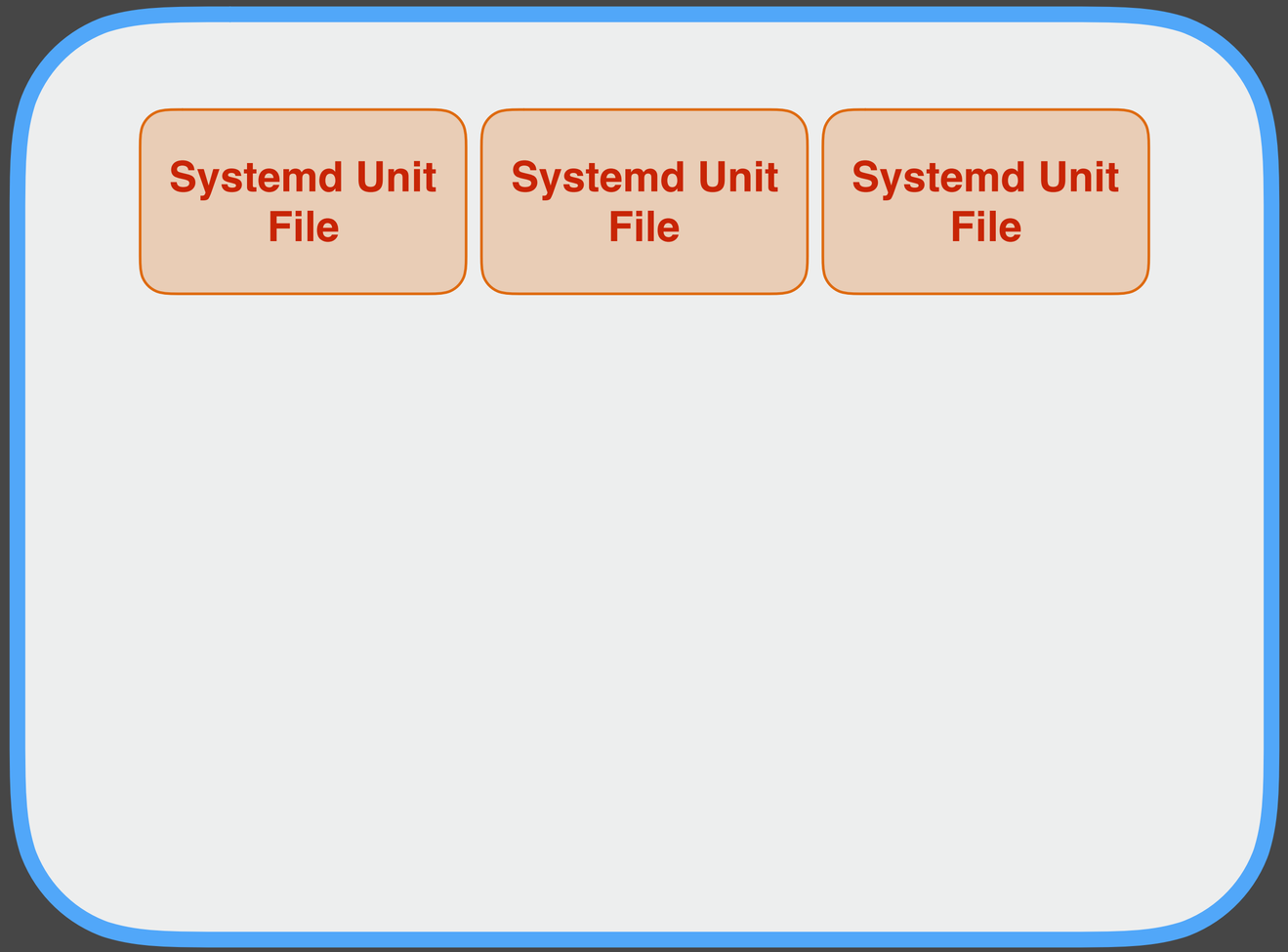
```
[Unit]
Description=My Advanced Service
After=etcd.service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill apache1
ExecStartPre=-/usr/bin/docker rm apache1
ExecStartPre=/usr/bin/docker pull coreos/apache

ExecStart=/usr/bin/docker run --name apache1 -p 8081:80 coreos/apache /usr/sb
-D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/10.10.10.123:8081 run

ExecStop=/usr/bin/docker stop apache1
ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/10.10.10.123:8081

[X-Fleet]
Conflicts=%p@*.service
```

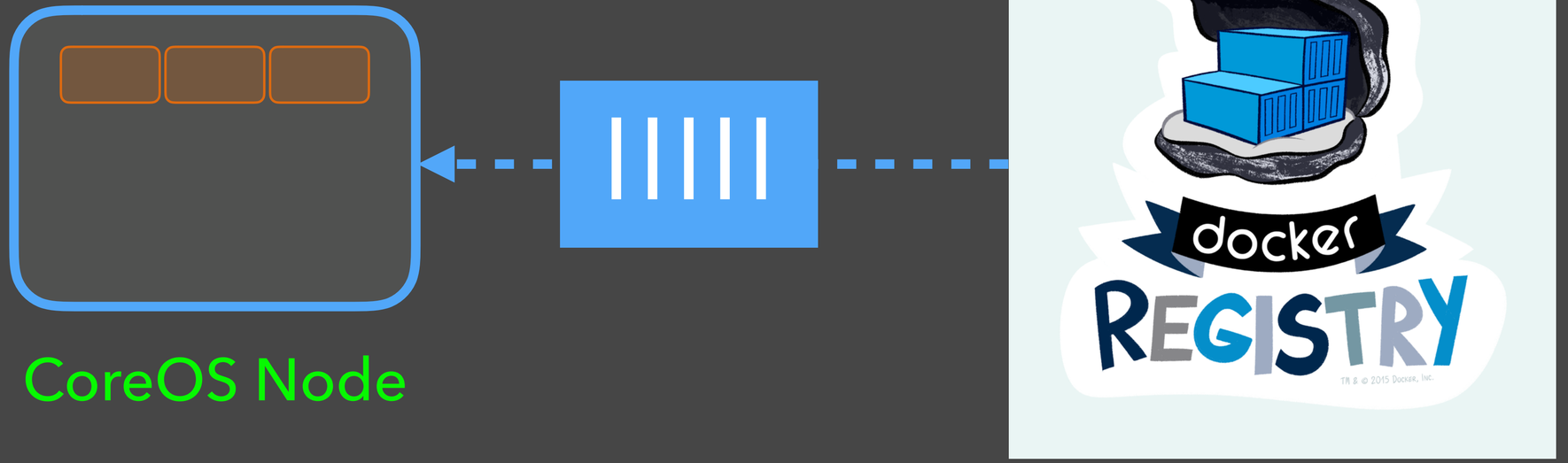
Option Name	Description
`MachineID`	`Require the unit be scheduled to the machine identified by the given string`
`MachineOf`	Limit eligible machines to the one that hosts a specific unit
`MachineMetadata`	Limit eligible machines to those with this specific metadata
`Conflicts`	Prevent a unit from being collocated with other specific units

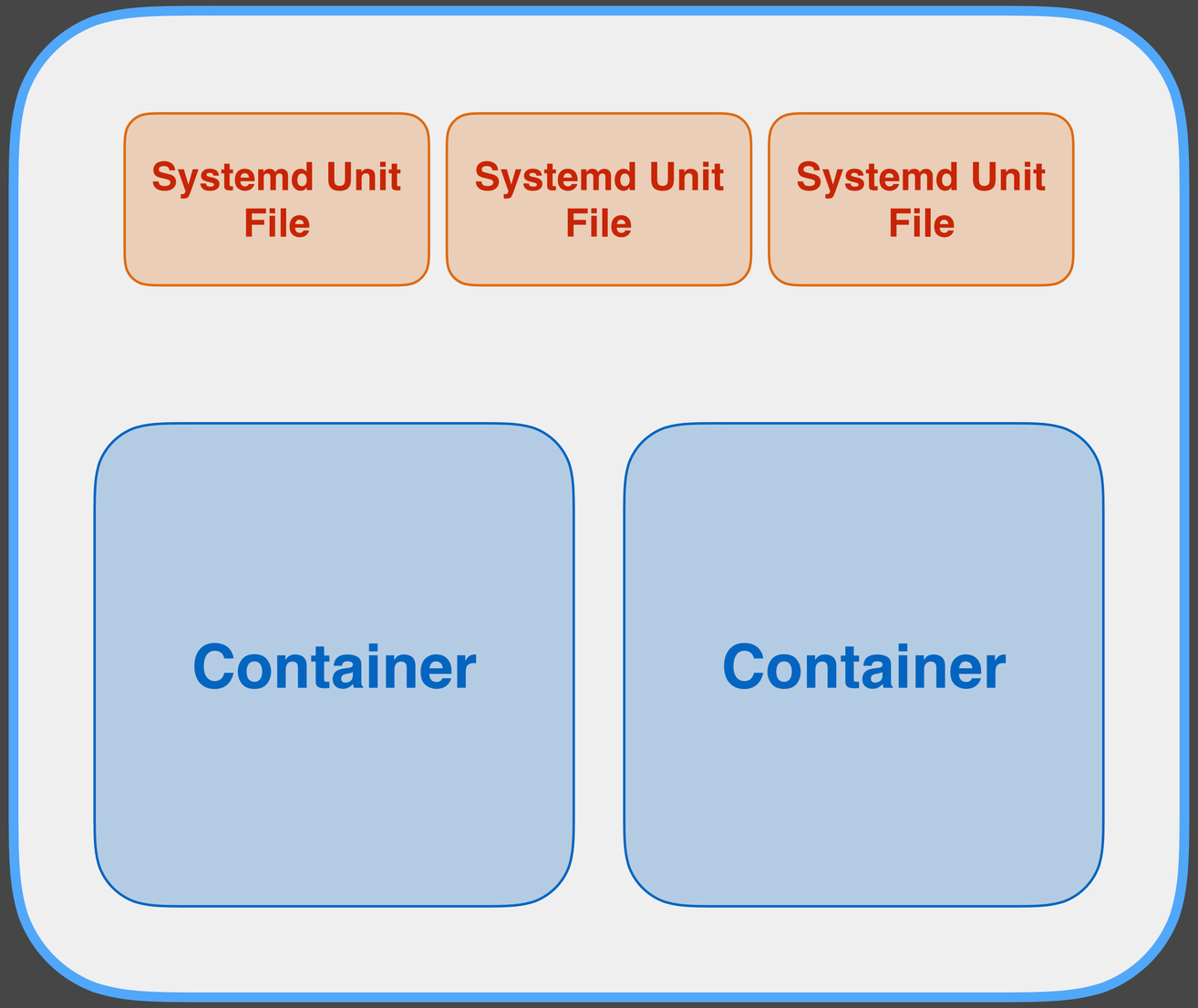
Application Deploys

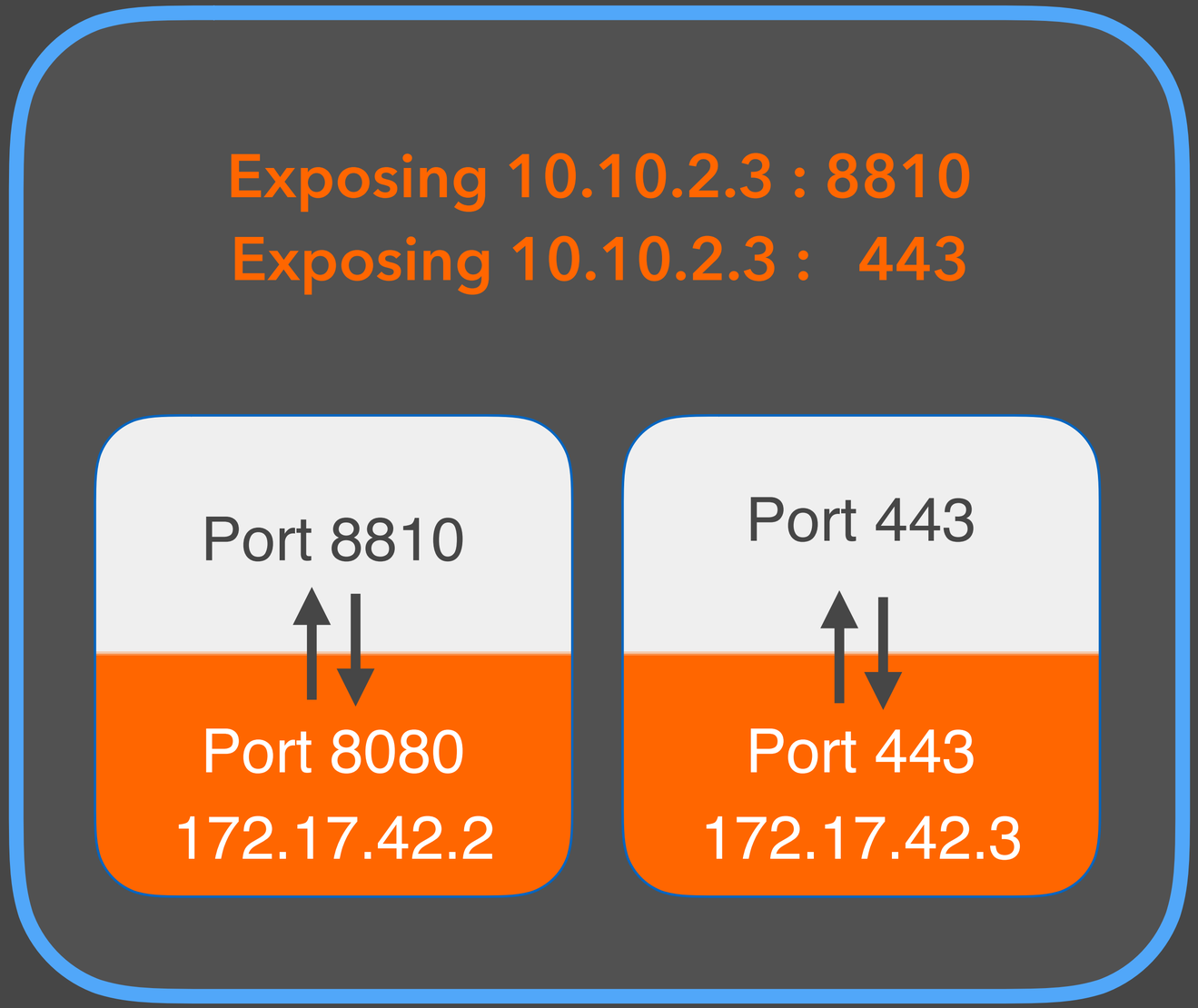
PUT **json version of unit file to fleet API**

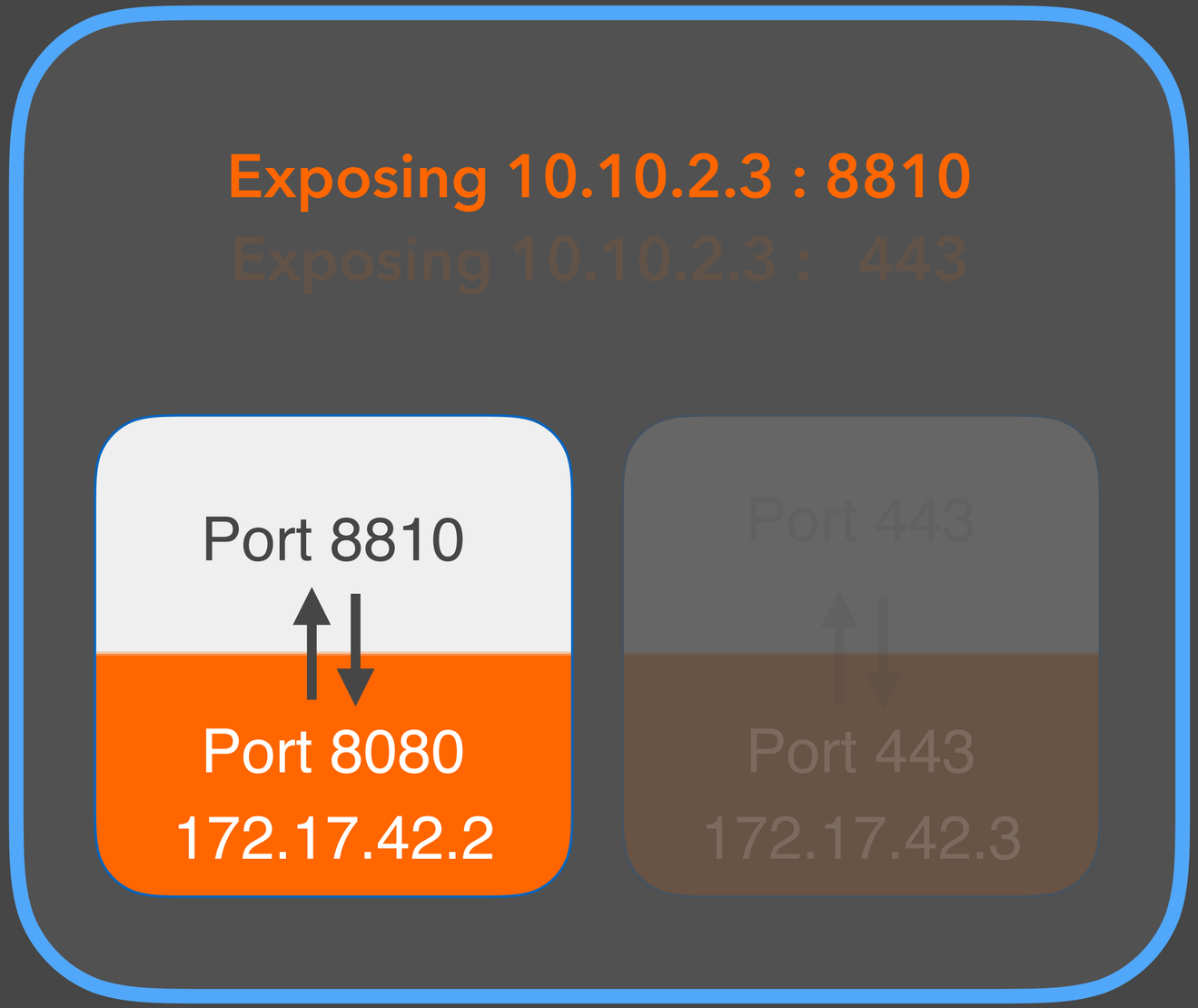
```
curl -s -X PUT -d @unit.json -H 'Content-Type:
application/json' coreos.demo.jpw.com:49153/fleet/v1/
units/mycoolapp@12.service
```

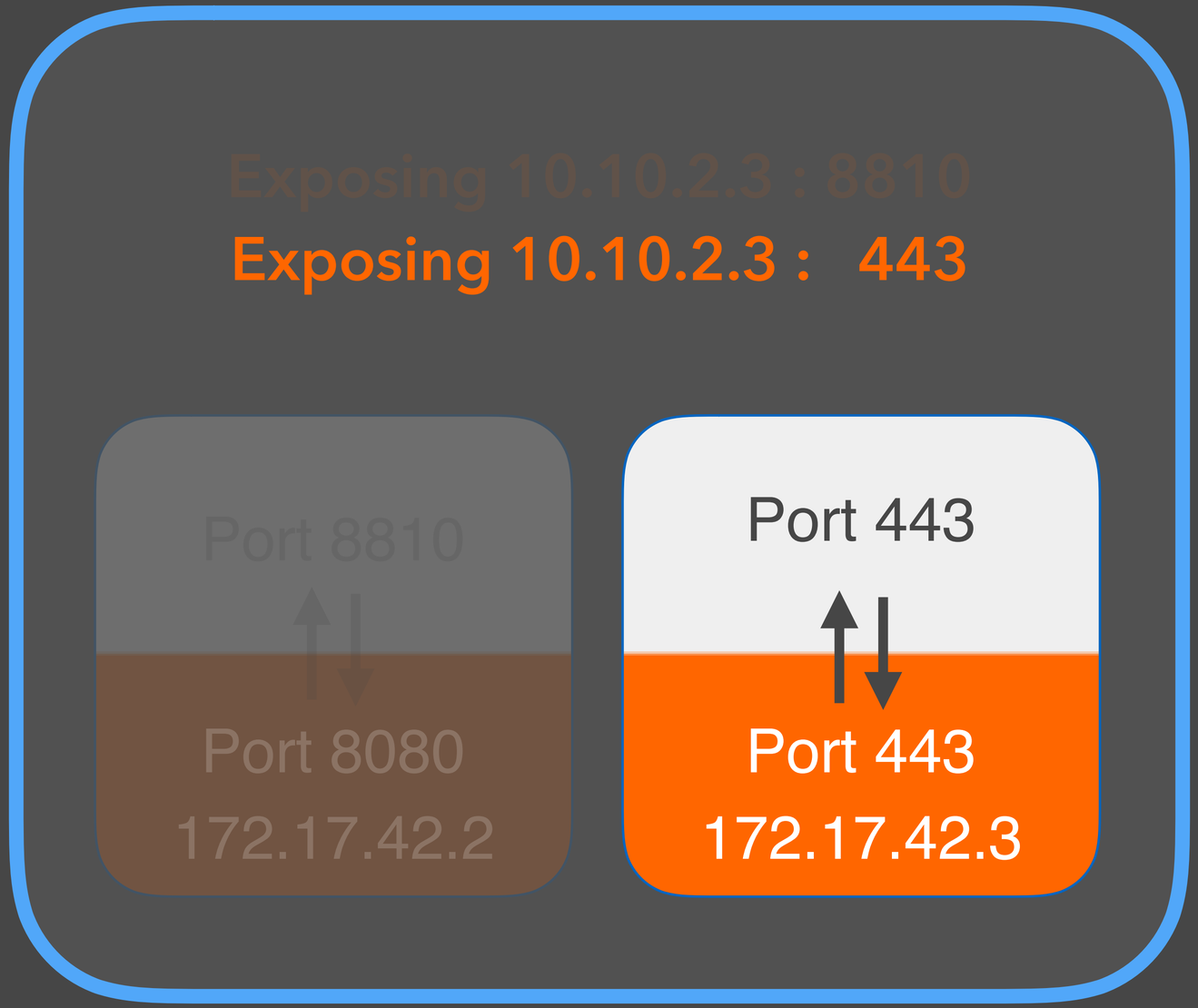

Systemd Unit File
Systemd Unit File
Systemd Unit File
CoreOS Node

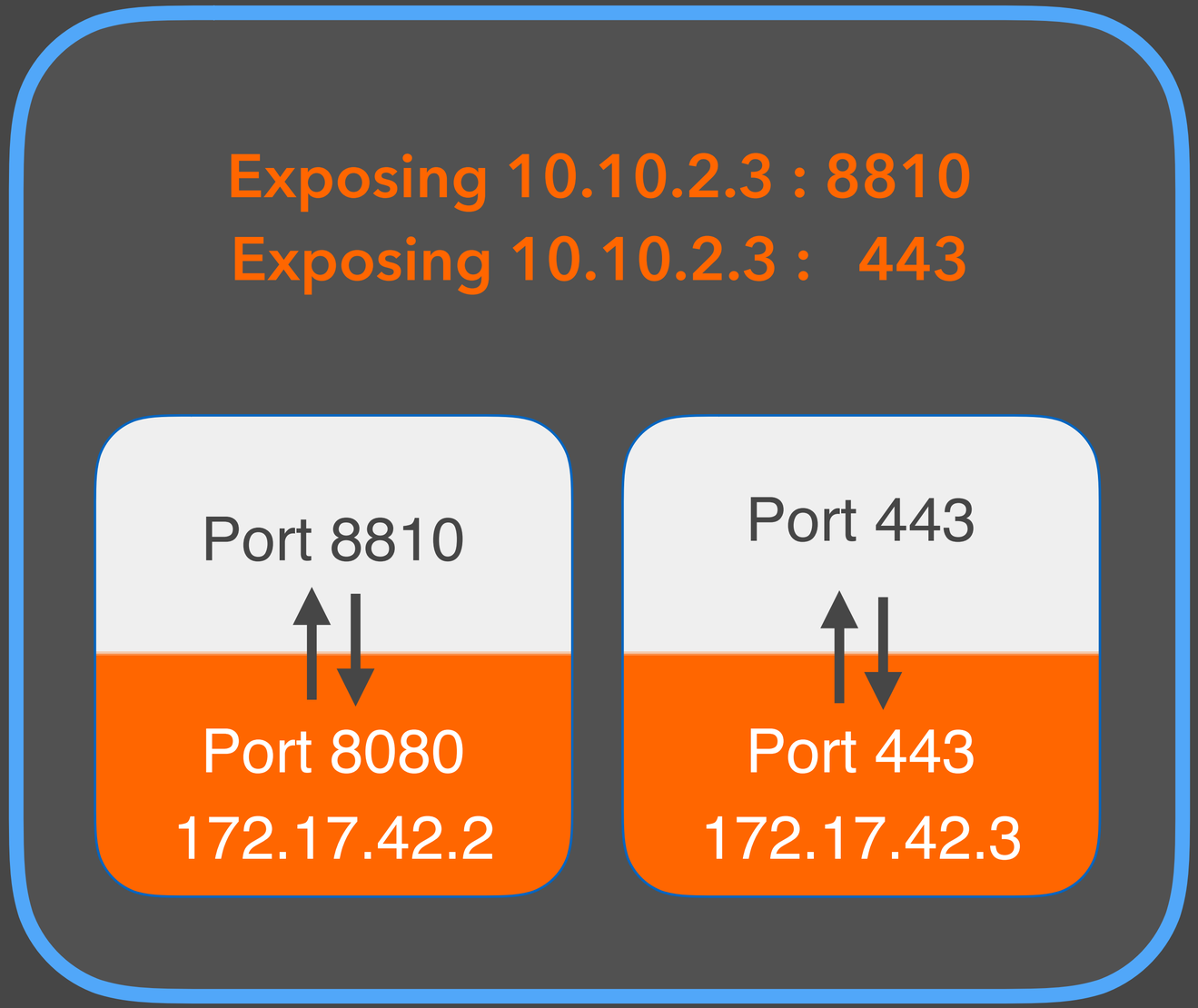

CoreOS Node
docker
REGISTRY


Systemd Unit File
Systemd Unit File
Systemd Unit File
Container
Container
CoreOS Node


Exposing 10.10.2.3 : 8810
Exposing 10.10.2.3 : 443
Port 8810
Port 8080
172.17.42.2
Port 443
Port 443
172.17.42.3
CoreOS Node | IP 10.10.2.3


Exposing 10.10.2.3 : 8810
Exposing 10.10.2.3 : 443
Port 8810
Port 8080
172.17.42.2
Port 443
Port 443
172.17.42.3
CoreOS Node | IP 10.10.2.3


Exposing 10.10.2.3 : 8810
Exposing 10.10.2.3 : 443
Port 8810
Port 8080
172.17.42.2
Port 443
Port 443
172.17.42.3
CoreOS Node | IP 10.10.2.3


Exposing 10.10.2.3 : 8810
Exposing 10.10.2.3 : 443
Port 8810
Port 8080
172.17.42.2
Port 443
Port 443
172.17.42.3
CoreOS Node | IP 10.10.2.3

Service Discovery and Configuration

```
# etcdctl ls /services/quad
```

```
# etcdctl ls /services/quad
/services/quad/10.253.156.164
/services/quad/10.253.156.165
```

```
# etcdctl ls /services/quad
/services/quad/10.253.156.164
/services/quad/10.253.156.165

# etcdctl get /services/quad/10.253.156.165
```

```
# etcdctl ls /services/quad
/services/quad/10.253.156.164
/services/quad/10.253.156.165

# etcdctl get /services/quad/10.253.156.165
10.253.156.165:8810
```

Hey Jim,
what is a sidekick container?

A container that performs other duties that are related to our main container but shouldn't be directly built into that application

Nginx Presence Unit File

```
[Unit]
BindsTo=nginx-1.0.0@%i.service

[Service]
EnvironmentFile=/etc/environment
ExecStartPre=-/usr/bin/docker kill nginx-presence-%i
ExecStartPre=-/usr/bin/docker rm nginx-presence-%i
ExecStart=/usr/bin/docker run --rm --name nginx-presence-%i -e AWS_ACCESS_KEY=foo -e AWS_SECRET_KE
AWS_REGION=us-east-1 -e ELB_NAME=coreos-appname -e PROXY_HOST=10.10.11.11 -e PROXY_PORT=3128 jpweb
presence
ExecStop=/usr/bin/docker stop nginx-presence-%i

[X-Fleet]
MachineOf=nginx-1.0.0-secure@%i.service
```

Nginx Presence Unit File

```
[Unit]
BindsTo=nginx-1.0.0@%i.service

[Service]
EnvironmentFile=/etc/environment
ExecStartPre=-/usr/bin/docker kill nginx-presence-%i
ExecStartPre=-/usr/bin/docker rm nginx-presence-%i
ExecStart=/usr/bin/docker run --rm --name nginx-presence-%i -e AWS_ACCESS_KEY=foo -e AWS_SECRET_KE
AWS_REGION=us-east-1 -e ELB_NAME=coreos-appname -e PROXY_HOST=10.10.11.11 -e PROXY_PORT=3128 jpweb
presence
ExecStop=/usr/bin/docker stop nginx-presence-%i

[X-Fleet]
MachineOf=nginx-1.0.0-secure@%i.service
```

Nginx Presence Unit File

```
[Unit]
BindsTo=nginx-1.0.0@%i.service

[Service]
EnvironmentFile=/etc/environment
ExecStartPre=-/usr/bin/docker kill nginx-presence-%i
ExecStartPre=-/usr/bin/docker rm nginx-presence-%i
ExecStart=/usr/bin/docker run --rm --name nginx-presence-%i -e AWS_ACCESS_KEY=foo -e AWS_SECRET_KE
AWS_REGION=us-east-1 -e ELB_NAME=coreos-appname -e PROXY_HOST=10.10.11.11 -e PROXY_PORT=3128 jpweb
presence
ExecStop=/usr/bin/docker stop nginx-presence-%i

[X-Fleet]
MachineOf=nginx-1.0.0-secure@%i.service
```

SDN

Software-defined Networking

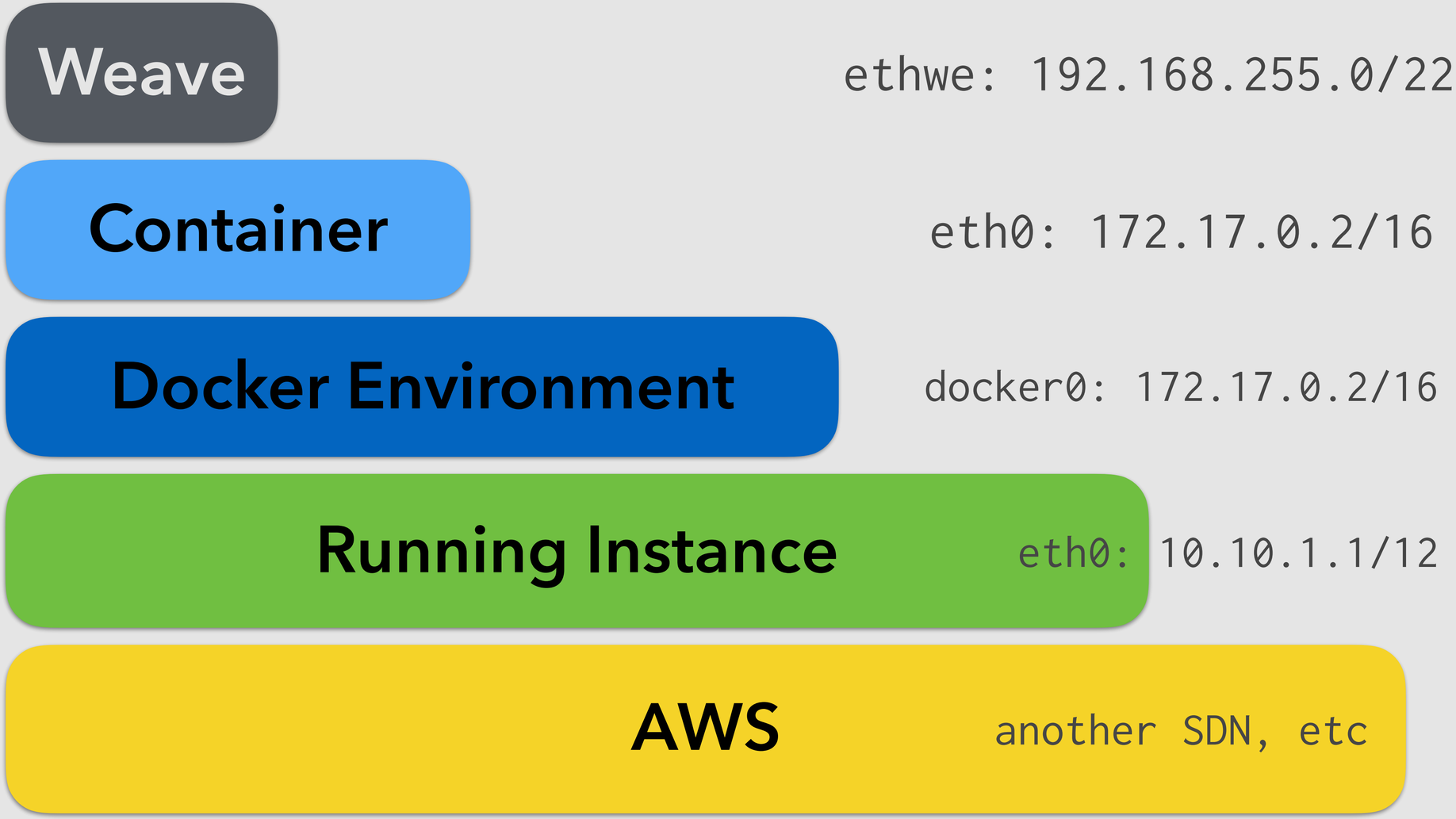
- Containers IP address allocation across cluster
- Dynamic dns across cluster.
- Encrypted connections
- Easy install short up and running time

- Container networking that runs in a container?


YO DAWG! I HEARD YOU LIKE NETWORKS
SO I PUT AN SDN IN A CONTAINER, IN A VM, THAT IS APART OF AN SDN, ON A HOST, ON A NETWORK, ON THE INTERNET
memegenerator.net

This time, yeah really

(on aws anyway)


Weave
ethwe: 192.168.255.0/22
Container
eth0: 172.17.0.2/16
Docker Environment
docker0: 172.17.0.2/16
Running Instance
eth0: 10.10.1.1/12
AWS
another SDN, etc

Lessons Learned

Reference Links

- https://coreos.com
- https://www.docker.com
- https://coreos.com/etcd/docs/latest/
- https://github.com/kelseyhightower/confd
- https://raft.github.io
- @jpw
- https://github.com/jpweber