Real virtual environments without virtualenv

Mihai Iachimovschi

@mishunika
mihai.iachimovschi@gmail.com

What’s inside?

What’s inside?

- Consistency

What’s inside?

- Consistency
- Diversity

What’s inside?

- Consistency
- Diversity
- Isolation

Building the dev env

Installing dependencies

- pip
- or the OS packaging tool
- or even easy_install?

Building the dev env (cont)

How?

- everything globally
- or inside a virtualenv
- or using conda
- Vagrant!?

Deploying

Deploying

Where?

Deploying

Where?

- PaaS




Deploying

Where?

- PaaS




- IaaS


Deploying

Where?

- PaaS

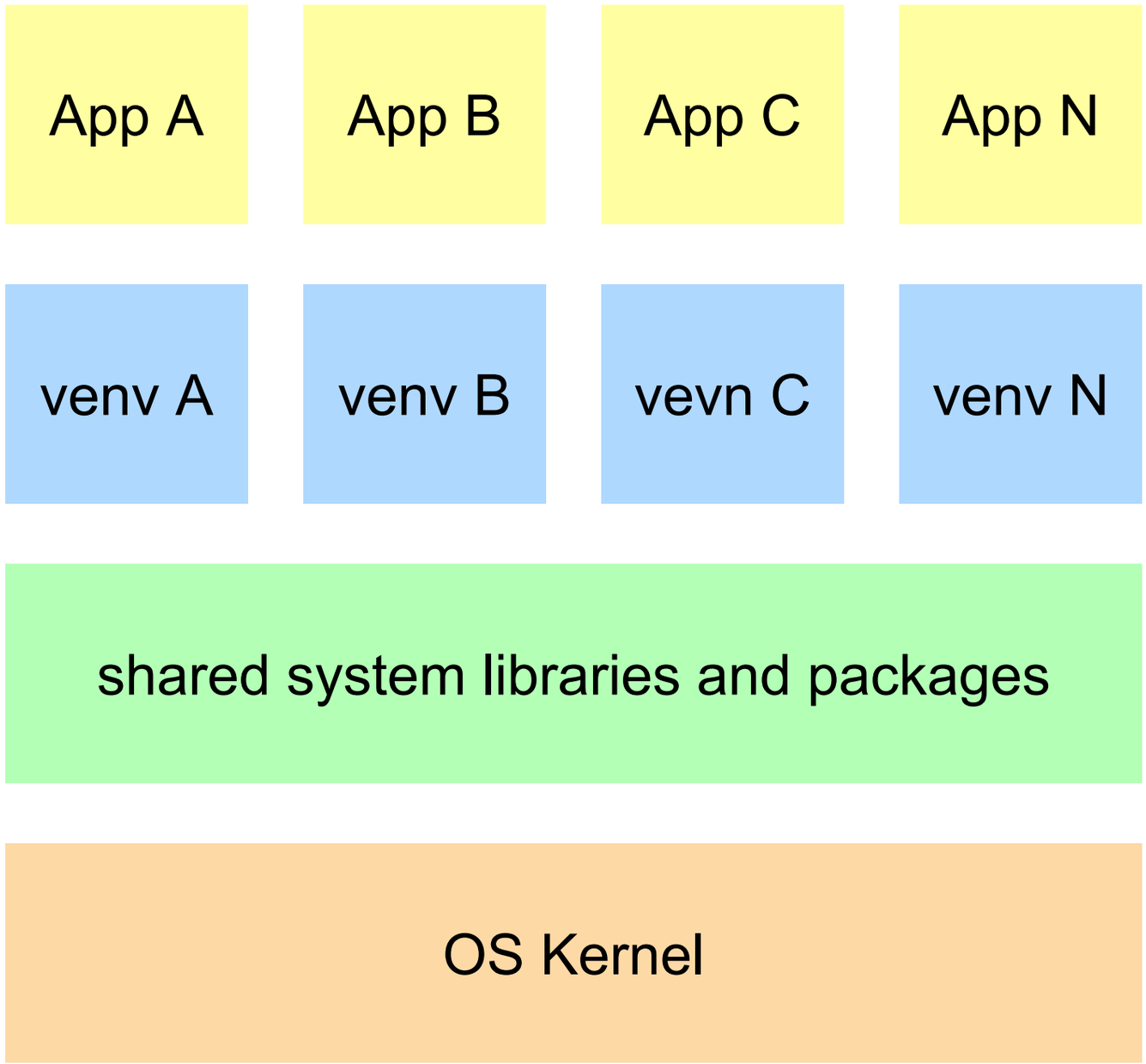



- IaaS



- your own server

Virtualenvs



When something breaks…

¯_(ツ)_/¯

IT WORKS

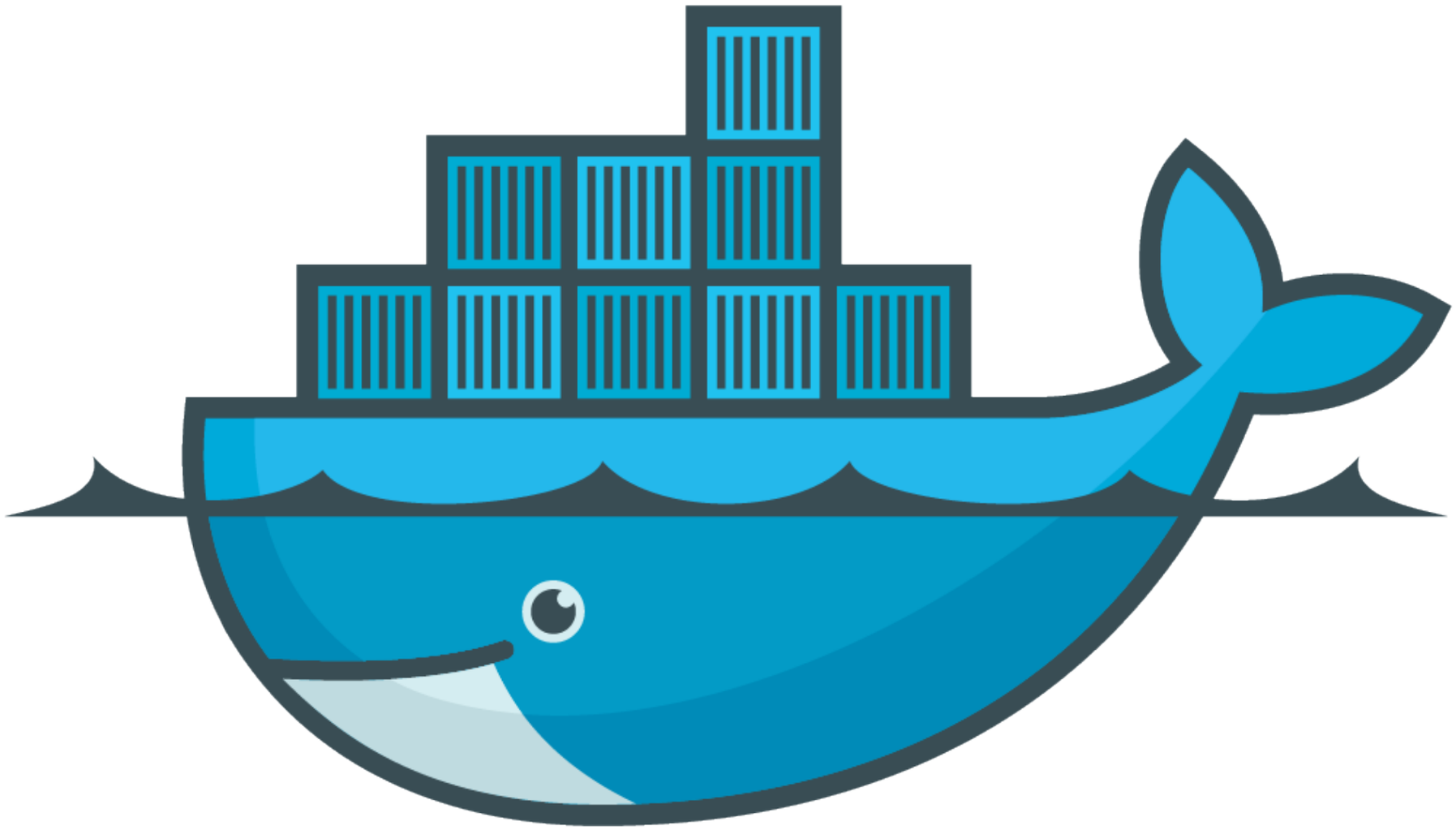
on my machine

Imagine…

Pack it!

Let’s see how we can **containerize** it.

Docker



“Container? It’s a VM!”

“Container? It’s a VM!”

It’s not!

“Container? It’s a VM!”

It’s not!

It’s just a process

“Container? It’s a VM!”

It’s not!

It’s just a process

It shares the system kernel

“Container? It’s a VM!”

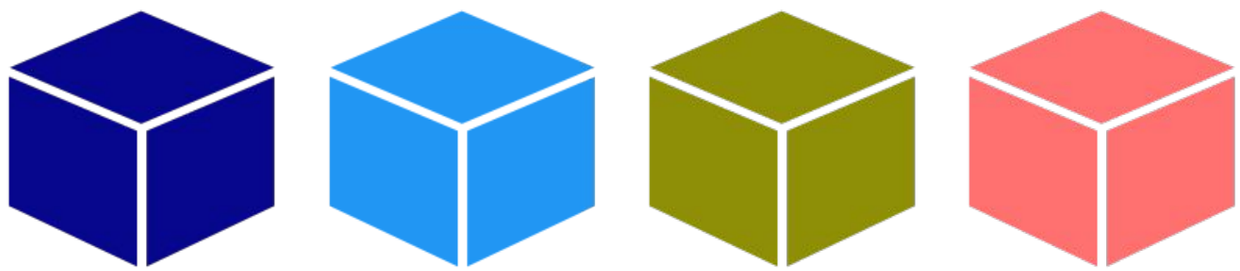
It’s not!

It’s just a process

It shares the system kernel

It is running in it’s own namespace

Containers

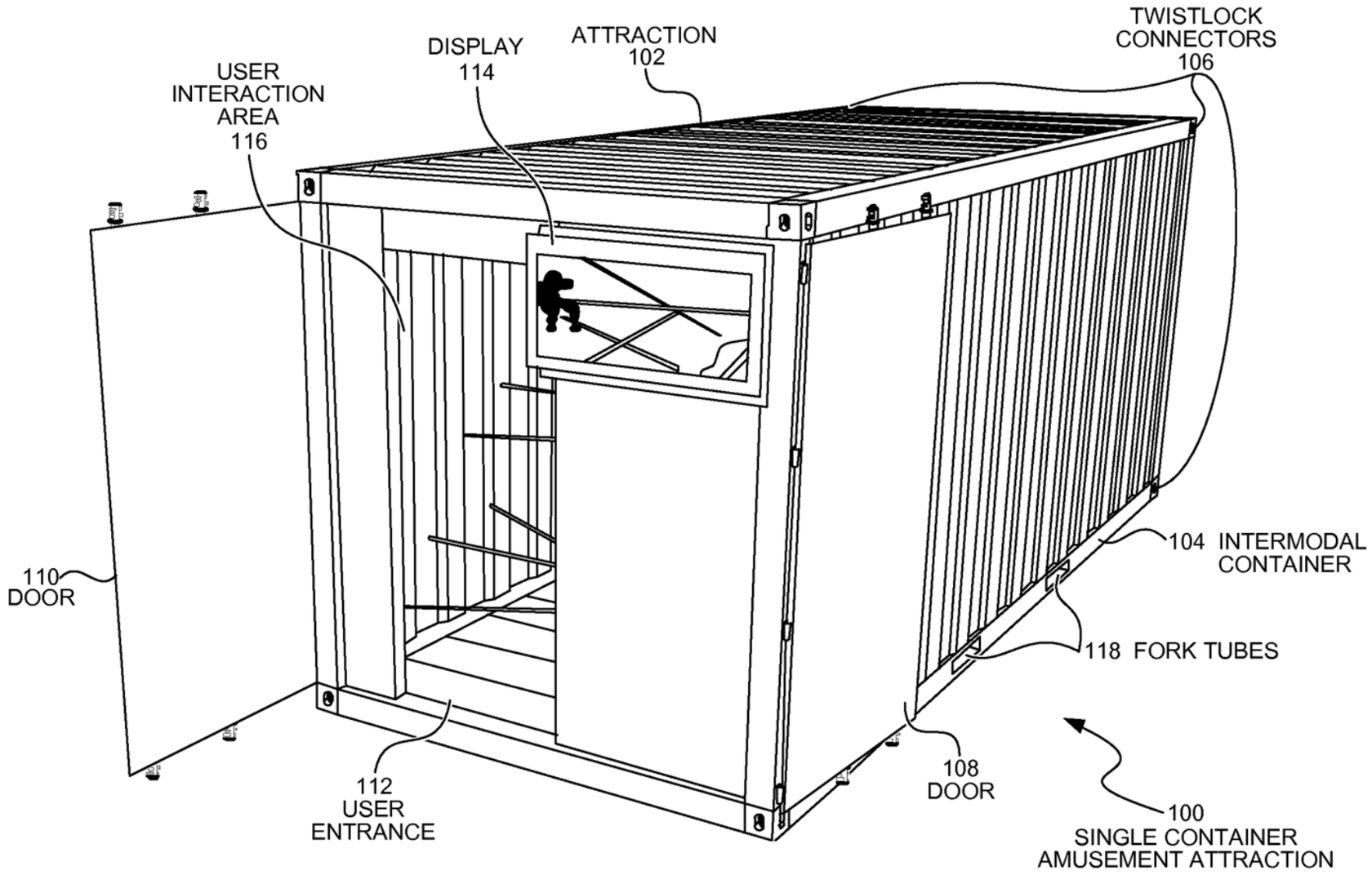


Docker Engine

shared system libraries and packages

OS Kernel

Container anatomy



Container anatomy (cont.)

Container anatomy (cont.)

cgroups

Container anatomy (cont.)

cgroups

namespaces

What's inside?

- System dependencies
- Separate libs & bins
- Limited endpoints

Docker images

Docker images

Self contained

Docker images

Self contained

Isolated

Docker images

Self contained

Isolated

Immutable

Docker images

Self contained

Isolated

Immutable

Portable

Docker containers

Isolated process

Disposable

Running over the image FS layers (AUFS)

AUFS

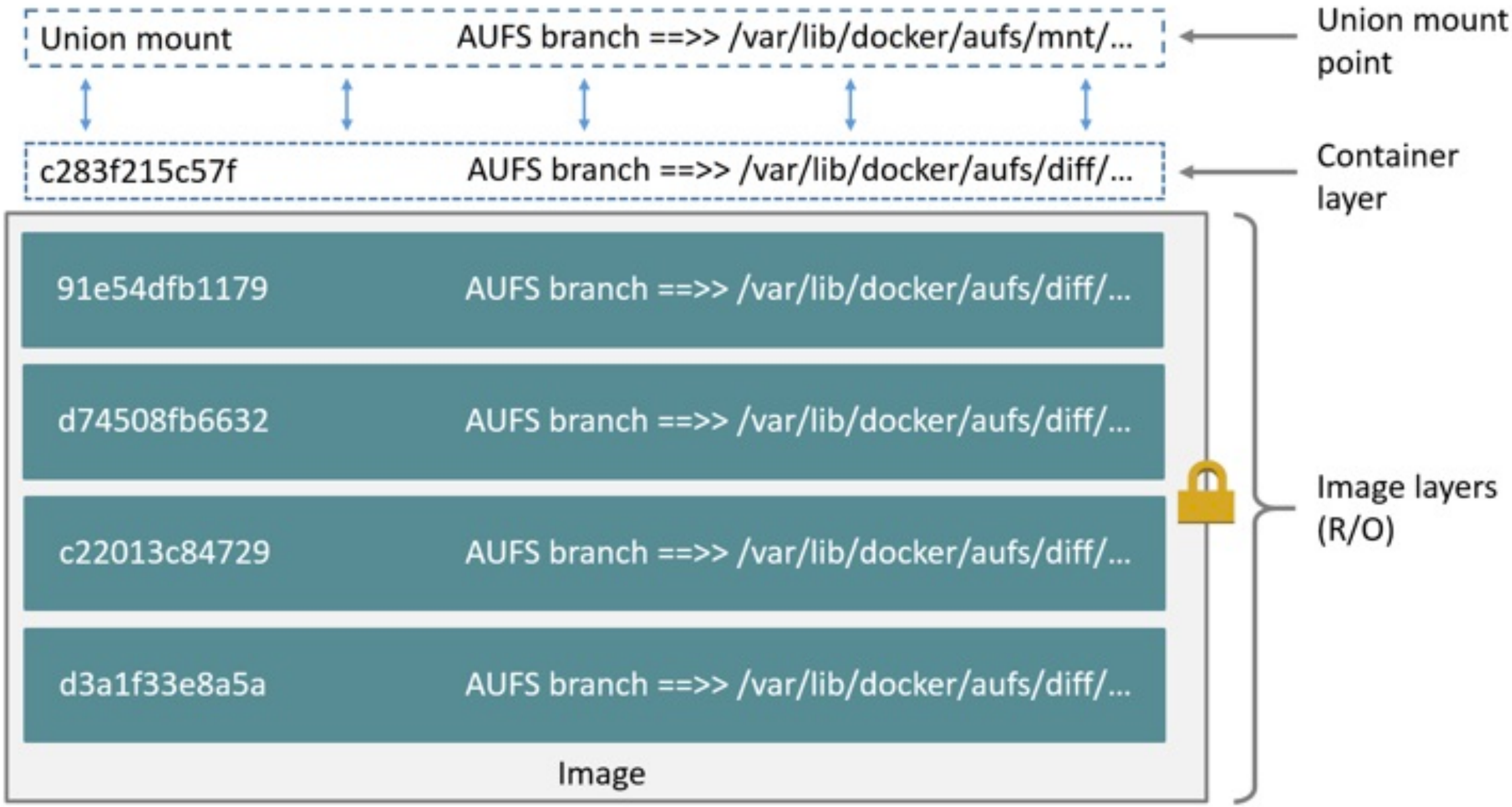


image: docker.org

Stacking = cumulative

How do we delete files?

Not re-writing them.

How we delete?

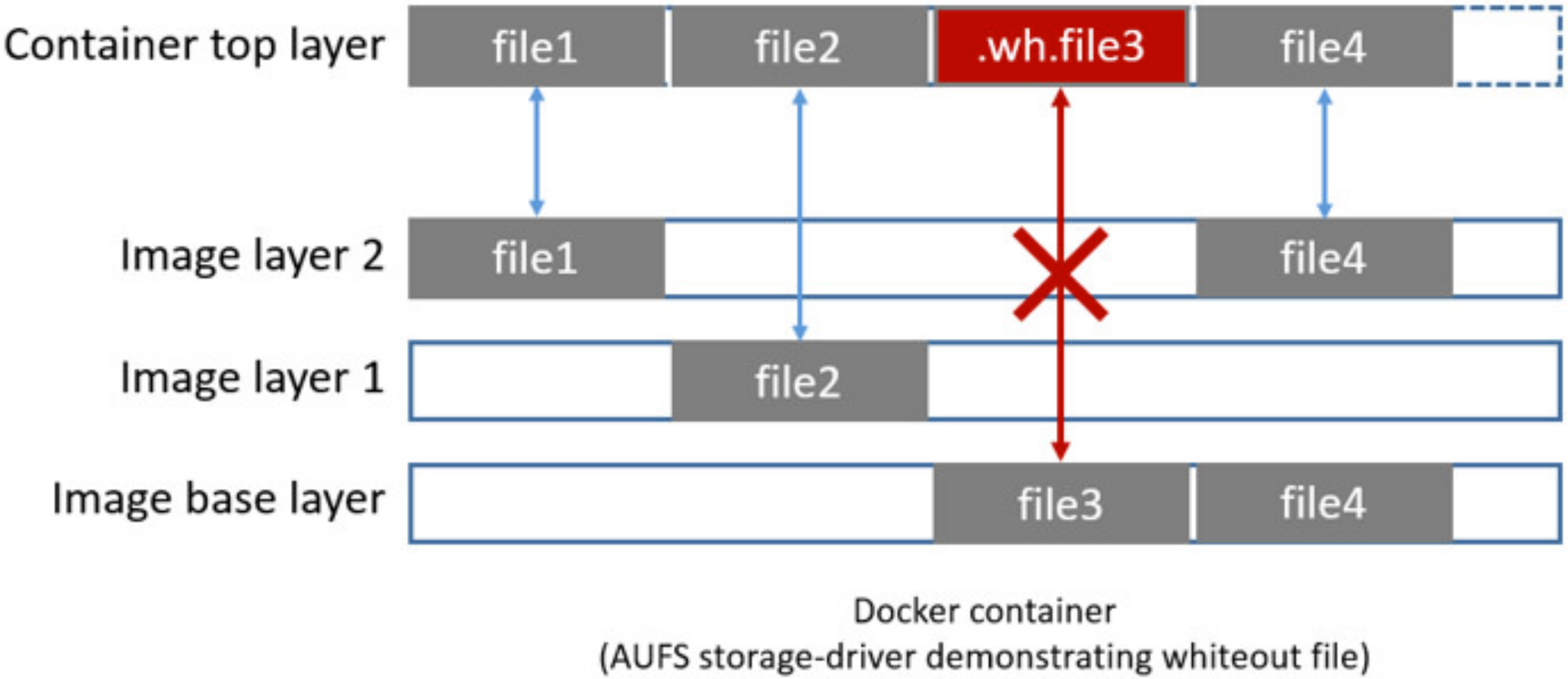


image: docker.org

Stacking = cumulative

~~How do we delete files?~~

~~Not re-writing them.~~

Beware!

It is still **cumulative**,
it'll increase the size of the final image.

“My data is not readonly!”

“My data is not readonly!”

How to handle the actual app data?

“My data is not readonly!”

How to handle the actual app data?

Volumes: Saving the state!

Building images

Building images

- Raw product (source code)

Building images

- Raw product (source code)
- Cooking recipe (Dockerfile)

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Dockerfile

```
FROM python:3.5
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

We'll go through

- Building
- Running
- Gathering logs
- Sneaking into a container

Build

```
docker build –t mihai/django-app .
```

Build

```
docker build -t mihai/django-app .
```

Build

```
docker build –t mihai/django-app .
```

Build

```
docker build -t mihai/django-app .
```

Build

```
docker build –t mihai/django-app .
```

Run

```
docker run --publish 8000:8000 \
    mihai/django-app \
    python manage.py \
    runserver 0.0.0.0:8000
```

Run

```
docker run --publish 8000:8000 \
        mihai/django-app \
        python manage.py \
        runserver 0.0.0.0:8000
```

Run

```
docker run --publish 8000:8000 \
    mihai/django-app \
    python manage.py \
    runserver 0.0.0.0:8000
```

Run

```
docker run --publish 8000:8000 \
    mihai/django-app \
    python manage.py \
    runserver 0.0.0.0:8000
```

Run

```
docker run --publish 8000:8000 \
           mihai/django-app \
 python manage.py \
 runserver 0.0.0.0:8000
```

Run

```
docker run --publish 8000:8000 \
    mihai/django-app \
    python manage.py \
    runserver 0.0.0.0:8000
```

Run (detached)

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run (detached)

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run (detached)

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run (detached)

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run +volumes

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           --volume $(pwd):/code
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run +volumes

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           --volume $(pwd):/code
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Run +volumes

```
docker run --publish 8000:8000 \
           --detach
           --name my_django_app
           --volume $(pwd):/code
           mihai/django-app \
           python manage.py \
           runserver 0.0.0.0:8000
```

Gather logs

```
docker logs -f my_django_app
```

Gather logs

```
docker logs -f my_django_app
```

Gather logs

```
docker logs -f my_django_app
```

Gather logs

```
docker logs -f my_django_app
```

Gather logs

```
docker logs -f my_django_app
```

Get inside the container

```
docker exec -it my_django_app sh
```

Get inside the container

```
docker exec -it my_django_app sh
```

Get inside the container

```
docker exec -it my_django_app sh
```

Get inside the container

```
docker exec -it my_django_app sh
```

Get inside the container

```
docker exec -it my_django_app sh
```

Get inside the container

```
docker exec -it my_django_app sh
```

So now we have it

Isolated

Has its dependencies bundled

It’s disposable

But that’s not enough

But that’s not enough

But that’s not enough

How about external dependencies?

But that’s not enough

How about external dependencies?

How about creating a full prod-like environment?

But that’s not enough

How about external dependencies?

How about creating a full prod-like environment?

No problem!
We just need more containers…

Stack them, it’s easy!

Orchestration

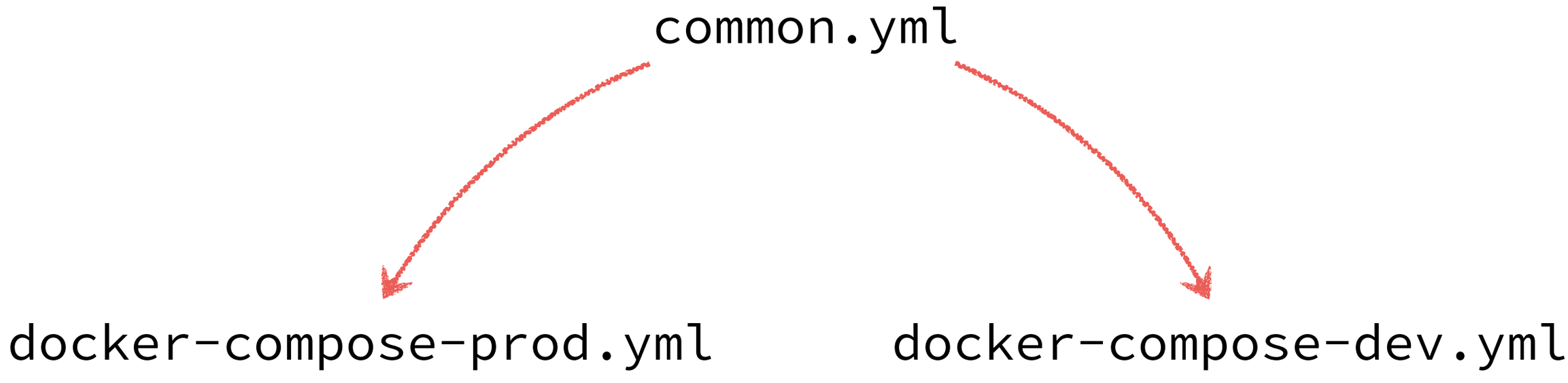
docker-compose.yml

```
version: '2'
services:
  db:
    image: postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
```

Run

```
docker-compose up
```

extended compose yml



Production compose

```
version: '2'
services:
  ...
  web:
    extends:
      file: common.yml
      service: web
    command: uwsgi --ini uwsgi.ini
    depends_on:
      - db
      - nginx
```

Production?

- Service discovery (Orchestration)
- Secrets
- Logging

Service discovery

- Compose
- Swarm
- Consul
- etcd
- ZooKeeper

Secrets

- env vars
- volumes with secrets
- docker-vault
- keywhiz

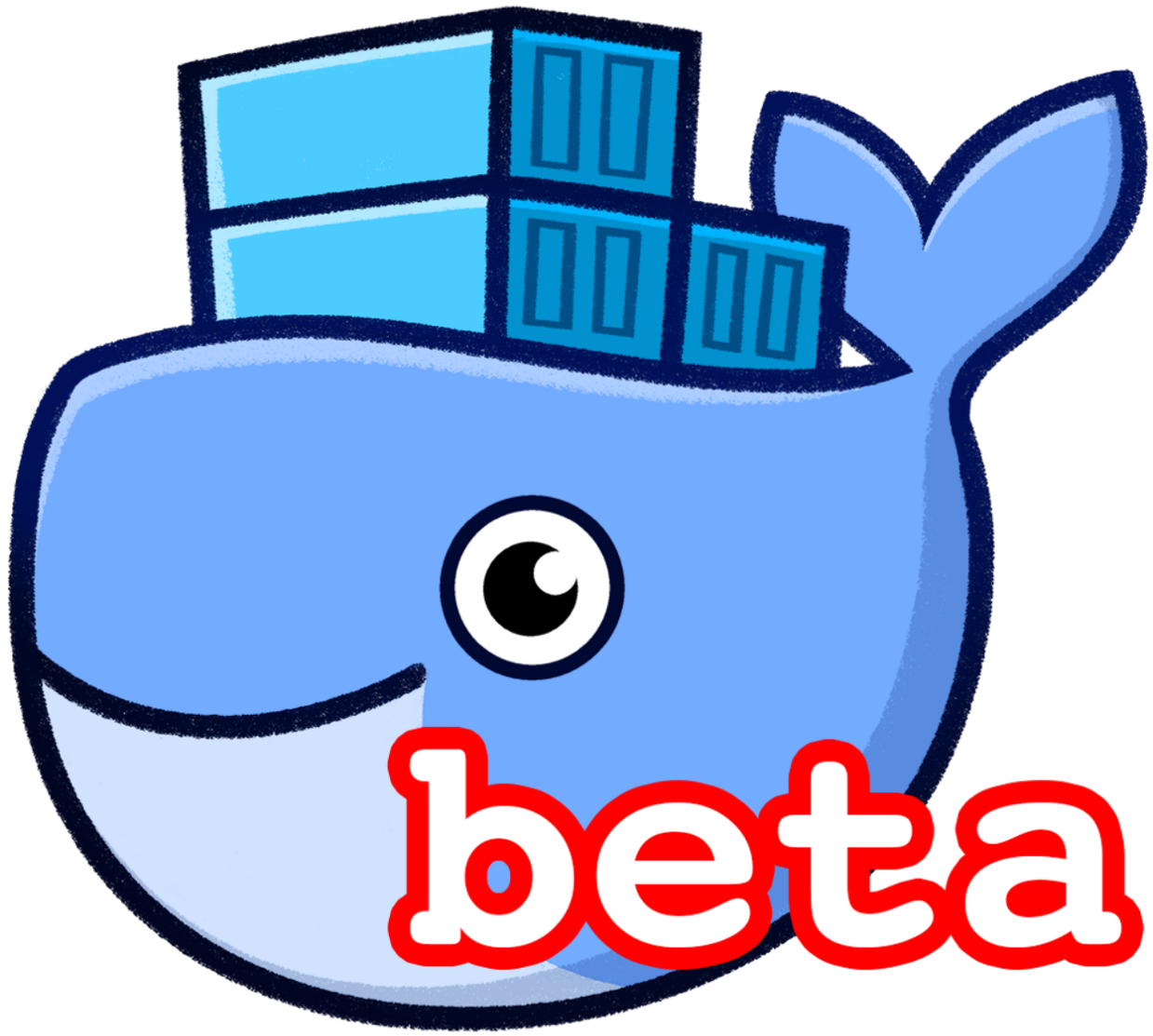
NB: Kubernetes has it built in

Logging

- stderr
- syslog
- journald
- json
- other custom logging drivers

Docker for Mac & Windows

- Mac: xhyve
- Win: Hyper-V



Summing up!

Environments:

- uniform (dev, prod)
- self-contained
- properly isolated
- immutable.

Fin.

Thank **you**.