Linux Containers & Docker

December 12, 2016 @ Urbino

Speakers: *Antonio Esposito*, *Michele Sorcinelli*

Reference teachers: *Andrea Seraghiti*, *Alessandro Bogliolo*

github.com/kobe25/docker-tutorial

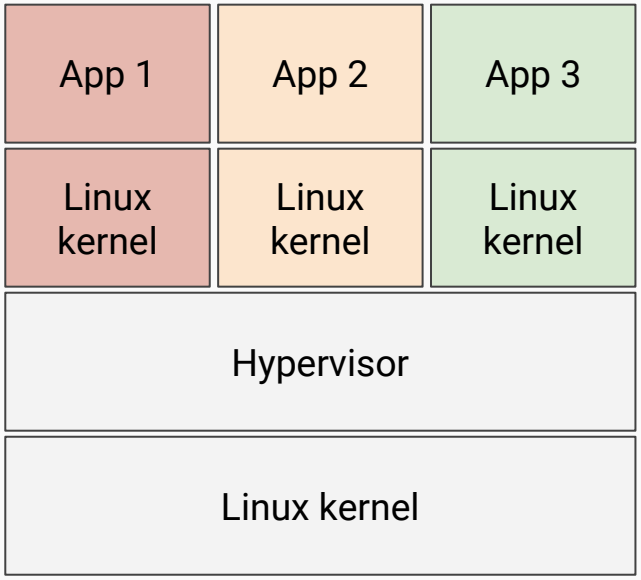
Code & Slides here!

Background

Causes of **high friction in bootstrapping** a project:

- obsolete documentation
- implicit tasks or dependency versions
- too coupled to a specific system
- tricky configuration

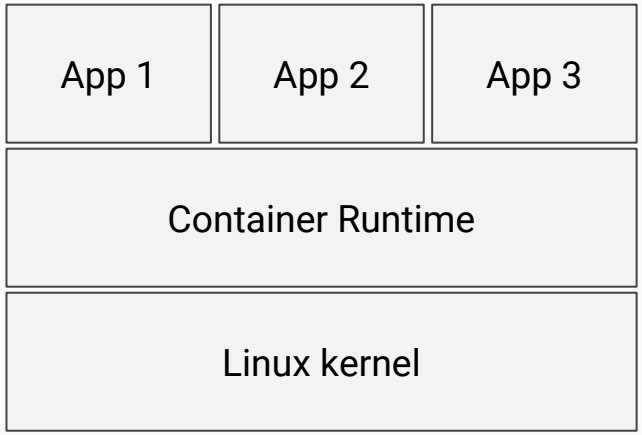
Virtual Machines



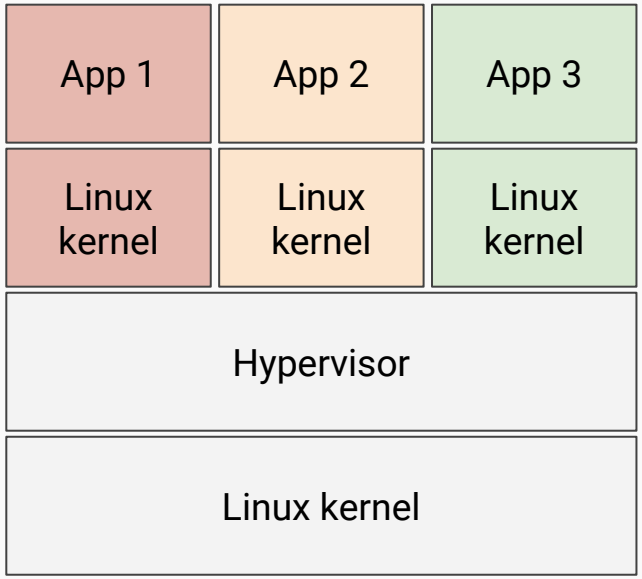
- reproducible environment with “configuration management” tools
- great isolation
- full operating system virtualization
- resource overhead

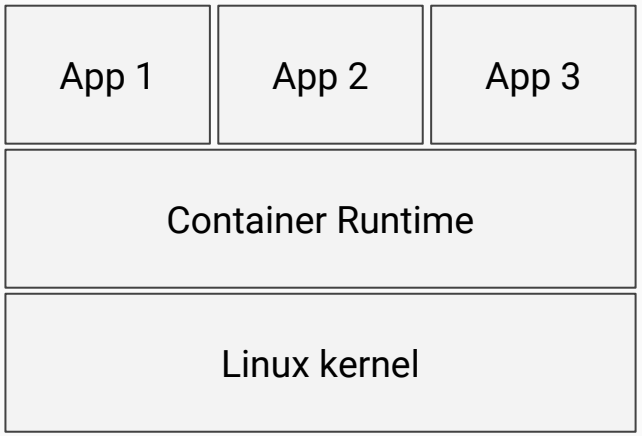
Containers

- native reproducible environment
- good isolation
- only application “virtualization”
- no resource overhead



Virtual Machines vs Containers





History of Containers

- 2006: cgroups (control groups)
- ...
- **2013: docker**
- 2014: rkt, kubernetes
- Today: Google runs ~2 billions containers every week

Docker: just a “container runtime”

- Provides a **user-friendly interface** to Linux kernel isolation features
- Native on Linux systems, quite integrated with OS X and Windows
- How To Install: https://www.docker.com/products/overview

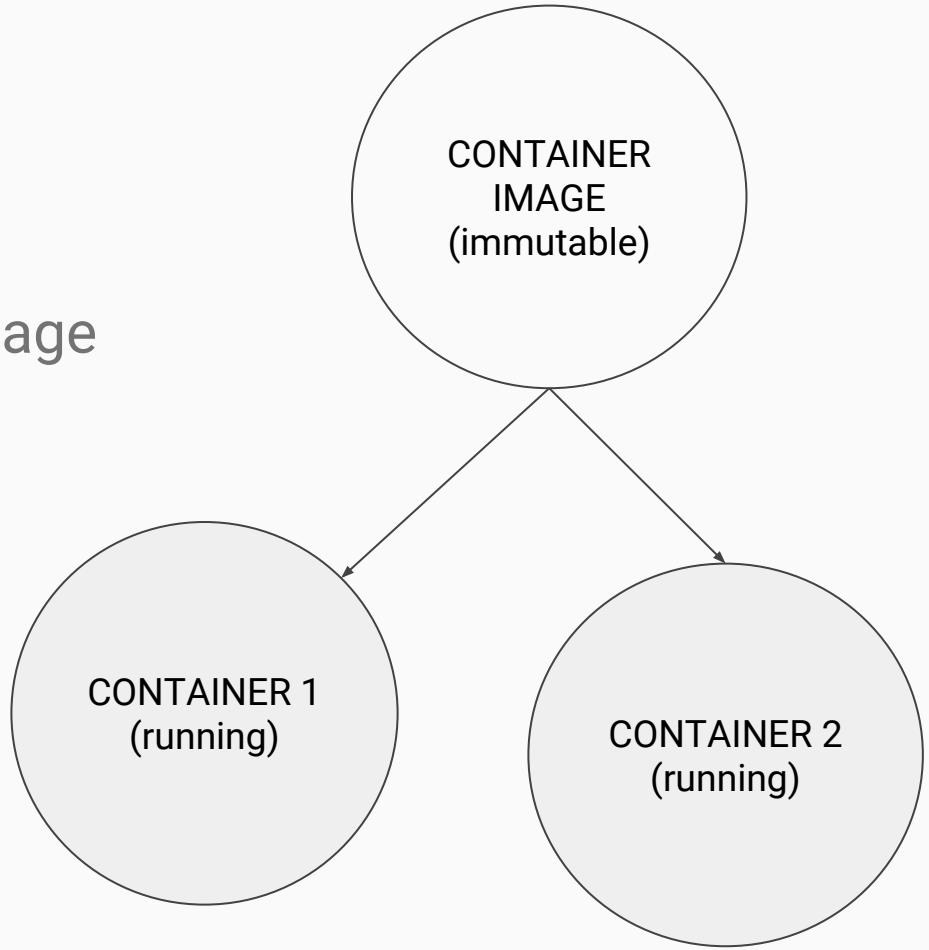
Isolation

Every container has its own:

- **process** and **memory** namespaces
- **file system** namespace
- **IP address** and networking

Container Images vs Containers

1. Build a container image
    - Images are **IMMUTABLE**
2. Run 1+ container(s) of this image
    - Containers are **RUNNING**



Containers in practice!

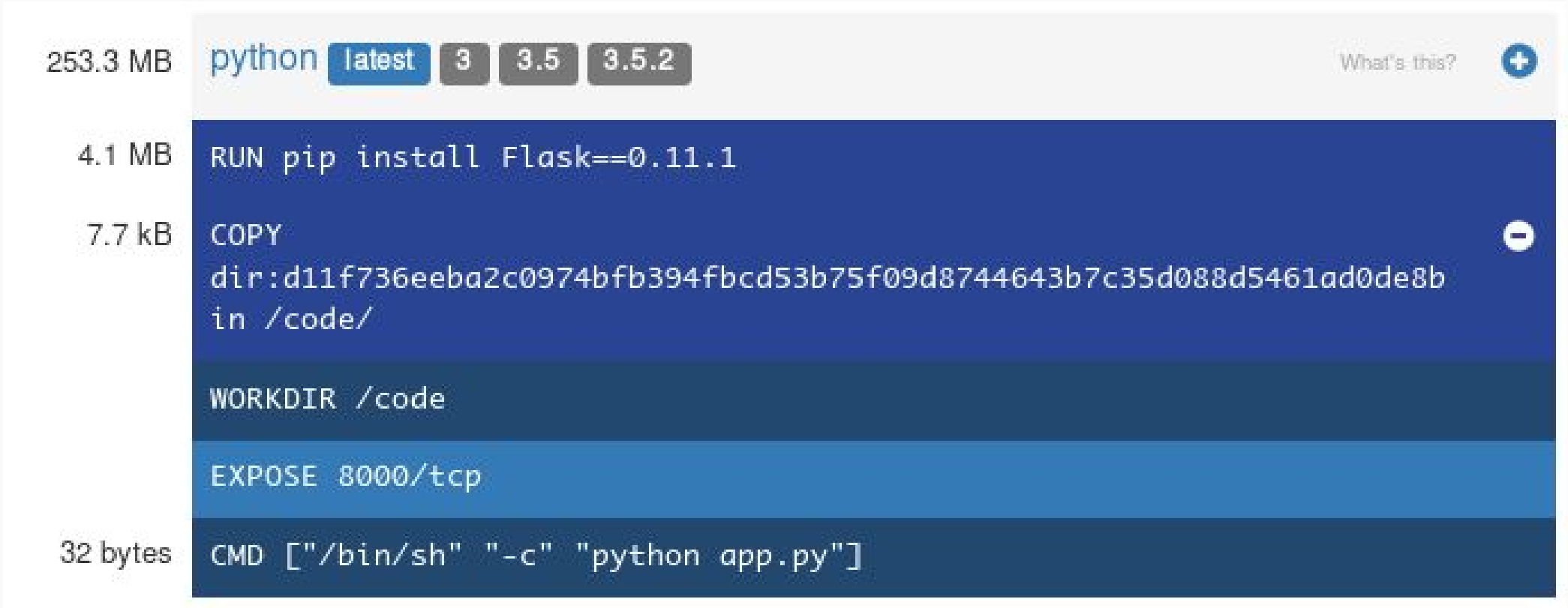
app.py

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return '<h1>Hello, World!</h1>'

app.run(host='0.0.0.0', port=8000)
```

Dockerfile

```
FROM python:3.5

RUN pip install Flask==0.11.1

COPY ./ /code/

WORKDIR /code

EXPOSE 8000

CMD python app.py
```

Dockerfile


253.3 MB
python
latest
3
3.5
3.5.2
What's this?
4.1 MB
RUN pip install Flask==0.11.1
7.7 kB
COPY dir:d11f736eeba2c0974bfb394fbcd53b75f09d8744643b7c35d088d5461ad0de8b in /code/
WORKDIR /code
EXPOSE 8000/tcp
32 bytes
CMD ["/bin/sh" "-c" "python app.py"]

`docker` command line

Show all Docker commands:

```
$ docker help
```

Show container images and container status:

```
$ docker images
```

```
$ docker ps
```

`docker` command line

Pull a Docker container image:

```
$ docker pull python

$ docker pull python:3.5

$ docker pull kobe25/docker-tutorial:latest

$ docker pull quay.io/coreos/etcd:latest
```

`docker` command line

Build a container image:

```
$ docker build -t app-image .
```

Create and start a container:

```
$ docker run --name app app-image
```

```
$ docker ps
```

`docker` command line

Manage containers:

```
$ docker stop app

$ docker start app

$ docker rm -f app
```

Other commands: `restart`, `kill`, `inspect`..

Permissions and Volumes

- container user != host user
- default container user is root
- container root user can do privilege escalation

Good practices:

- use a non-root user
- host UID == container UID (because of file system permission on volume mounts)

Networking

- PTP interface between host and container
- Dedicated bridges between containers (namespacing)
- DNS inside containers (not host)
  - resolving
  - ping

Resource Constraints

Limit CPU usage:

```
$ docker run --cpu-shares=20 mysql
```

Limit memory usage:

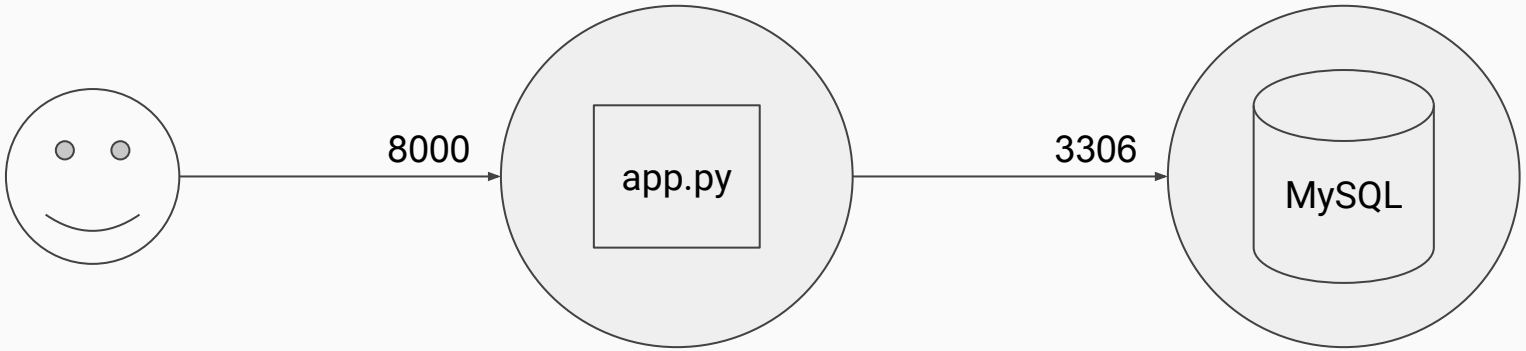
```
$ docker run --memory=1024 mysql
```

Others: network I/O, disk I/O…

Multi-Container Applications

Docker Compose

- Enables **multi-container applications**
- Offers a user-friendly **configuration file**



docker-compose.yml

```
version: '2'
services:
  db:
    image: mongo
  app:
    build: ./
    volumes: [‘.:/code’]
    ports: [‘127.0.0.1:8000:8000’]
    depends_on: [‘db’]
```

`docker-compose` command line

Build multiple container images and run a multi-container application:

```
$ docker-compose up -d
```

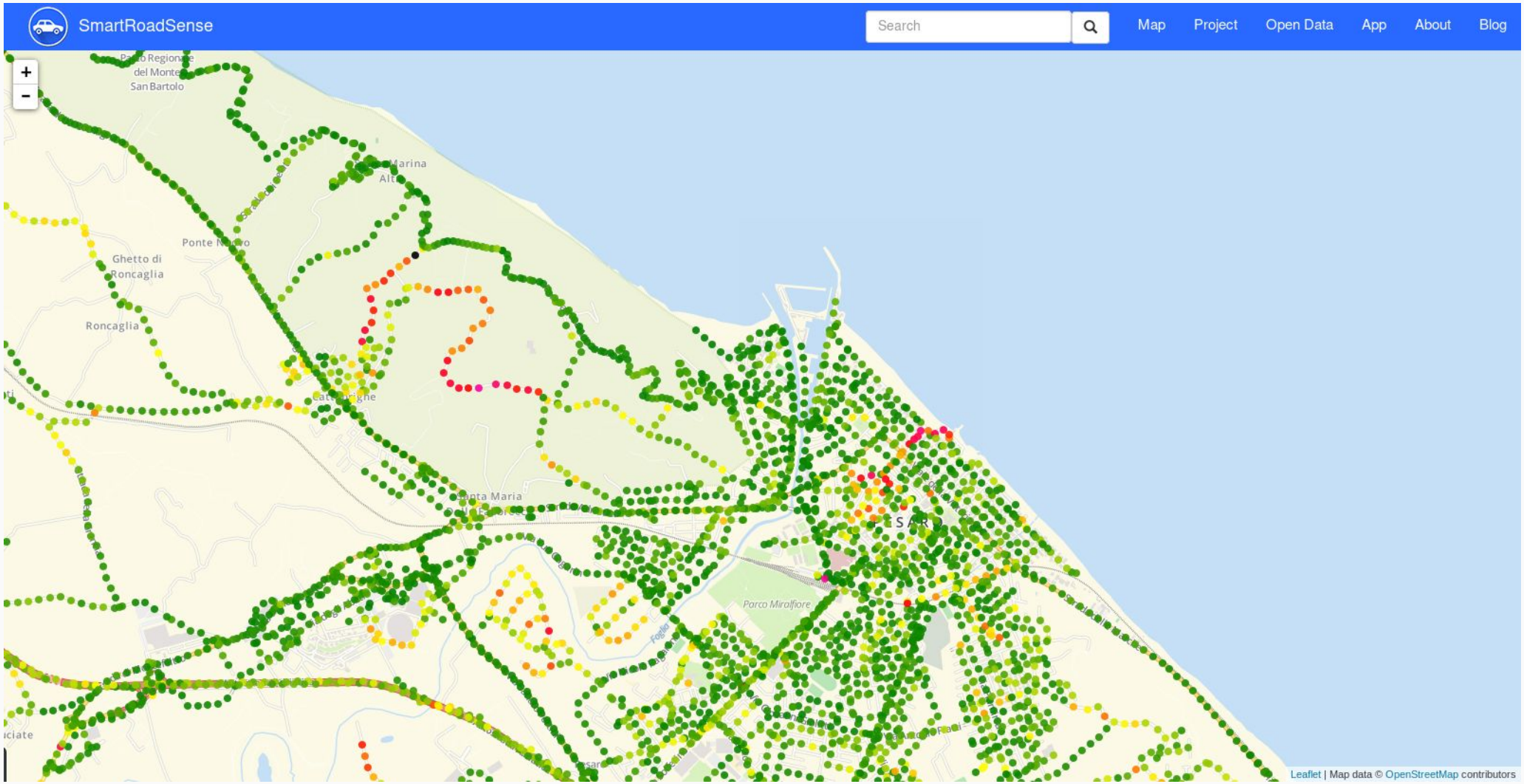
Check current status:

```
$ docker-compose ps
```

Documentation

- Docker Engine (or simply Docker): https://docs.docker.com/engine/
- Docker Compose: https://docs.docker.com/compose/

SmartRoadSense.it



SmartRoadSense: a real use case

Persistent Services:

- 3 PostgreSQL/PostGIS
- 1 MySQL
- 5 PHP
- 1 Node.js
- 1 Nginx
- 1 Apache

Scheduled Jobs:

- 2 PHP
- 1 Go
- 1 Bash

On demand:

- 3 database clients
- 1 Node.js

Challenges in Production

- Developers deploy 1+ application(s) to production
- High Availability requires a cluster with 3+ servers
- Optimizing resources

Container “Production Ecosystem”

CoreOS is a GNU/Linux distribution optimized for containers

- low overhead, high density

Kubernetes/OpenShift is a Platform as a Service (PaaS)

- automating deployment, scaling, and management of containerized applications in server clusters

Cloud Computing

Software as a Service (SaaS)
Applications for End Users

Infrastructure as a Service (IaaS)
RAW Compute, Network and Storage

Software as a Service (SaaS)
Applications for End Users

Platform as a Service (PaaS)
Deploying Applications by Developers

Infrastructure as a Service (IaaS)
RAW Compute, Network and Storage

Thanks for Your Attention !

QUESTIONS?