A microservice architecture with Docker

Arnaud Cogoluègnes, Zenika
Context

(health) insurance company

existing web applications

monolithic relational database

most of the business logic in the database
Context

problem: difficult to re-use business logic

solution: expose high business value web services
Big picture

- Configuration server
- API Gateway
- Backend
- Frontend
- Client (browser)
- Service registry

Mesos Cluster
Configuration server

Git repo

Get properties/YAML from repo.

Configuration server

API Gateway

Service registry

Mesos Cluster

Backend

Frontend

All applications then use the configuration server through HTTP.
Docker primer

Binary isolation
Resource isolation
Unified packaging
Service registry

Applications registers themselves in the service registry.
API gateway

Routes client calls to services. Gets services locations from the service registry.

Configuration server → API Gateway → Service registry

Mesos Cluster

Backend

Frontend
Mesos cluster

- Mesos cluster
  - Slave
  - Configuration server
  - API Gateway
  - Service registry
  - Zookeeper: For leader election and storage.
  - Scheduler: Handles applications launching.
  - Marathon master(s): Hosts applications.
Mesos 2-tier resource scheduling
Multiple environments

Keep configuration server and Mesos cluster. Add API Gateway and Service Registry instances for each environment.
Architecture

HTTPS → HAProxy  → API Gateway  → Web Service 1

HTTP → HAProxy  → API Gateway  → Web Service 2

virtual IP (keepalived)

“Static” architecture

Mesos
<table>
<thead>
<tr>
<th>What</th>
<th>Tool</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>CentOS, Debian</td>
<td>Ubuntu, Fedora, CoreOS, Rancher OS</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>Ansible</td>
<td>Chef, Puppet, Salt</td>
</tr>
<tr>
<td>Configuration Server</td>
<td>Spring Cloud Configuration Server</td>
<td>etcd, Consul, Zookeeper, Netflix Archaius</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>Netflix Eureka</td>
<td>Consul, etcd, Zookeeper</td>
</tr>
<tr>
<td>Dynamic Load Balancing</td>
<td>Netflix Zuul</td>
<td>HAProxy, Traefik</td>
</tr>
<tr>
<td>Orchestration</td>
<td>Mesos &amp; Marathon</td>
<td>Rancher, Fleet, Kubernetes</td>
</tr>
<tr>
<td>Centralized Logging</td>
<td>Graylog</td>
<td>ELK</td>
</tr>
</tbody>
</table>
Disruptive?

stateless: no web session, no state on Mesos slaves

logs: no file, GELF + UDP

backup: none for Mesos masters and slaves

redundancy: no hardware redundancy for Mesos slaves

environments: dev and production apps on same hosts
Not so disruptive

OS: nothing fancy

Docker: only for stateless applications

storage: in traditional VM, not in container
Docker in production

One packaging to rule them all

Hard to measure daemon memory consumption

Use direct LVM on CentOS

Use a (internal) Docker Registry ASAP
Wrapping up

Microservices are difficult
Docker in production is not so difficult
Embrace DevOps practices
Adopt what you can control