Service Discovery in OSGi

Beyond the JVM using Docker, Consul and Registrator

About me

CTO @ Dexels

Architect at Sendrato Wearables

Mildly incoherent at times

Service Discovery in OSGi

Beyond the JVM using Docker, Consul and Registrator

- Service Discovery ‘Theory’
- Pile up technologies
- Demo
- Conclusion
- Wild discussions

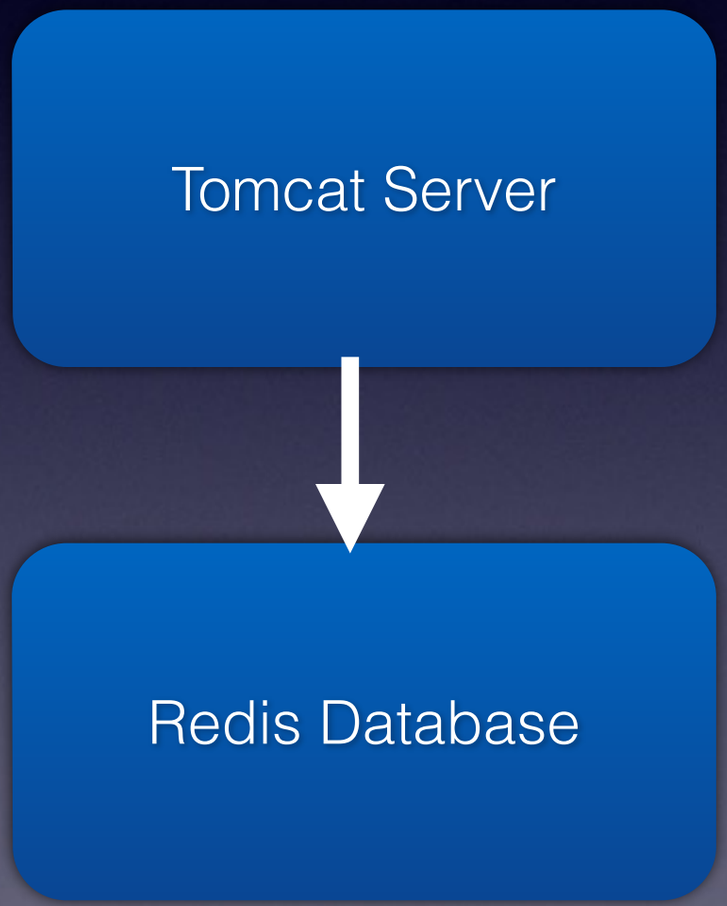
Service Discovery (SOA)

- Can change providers at will
- Access to many 3rd party services
- Services go shopping for other services
- Services can respond to ‘markets’

Service Discovery for normal people

Find system components

Example



Traditional solutions

- Hard code it
- DNS
- Configuration management

Why does that need to change?

- Horizontally scaled architectures
- Cloud infrastructure
- Micro services

How?

- Finding services by exploring systems
- Use a well-known central registry where every service registers

Service Repository

- Distributed key-value stores
- Low volume
- Resilience and availability are most important

Bear with me!

Consul

- Distributed Key/Value configuration store
- Like Etcd or Zookeeper, but opinionated
- Availability over consistency
- Nice web UI
- Service check agent
- DNS and HTTP API

Consul DNS Api

- Use consul as DNS server
- Serves: service.consul domain
- Resolve service with [tags].<name>.service.consul
- Legacy friendly

DNS limitations

- Port is missing (except in SRV records)
- Not designed for dynamic data

Consul HTTP Api

- All Consul functions available
- No out of the box support
- Event notification with long polling

So suppose this thing works…

- We have a reliable, distributed service store

How do we feed it?

Docker containers

- Light-weight virtual machine
- Process with a filesystem and a network
- Universal, polyglot application deployment mechanism

A Docker container from a service perspective

- An image
- An id
- Exposed ports
- Labels

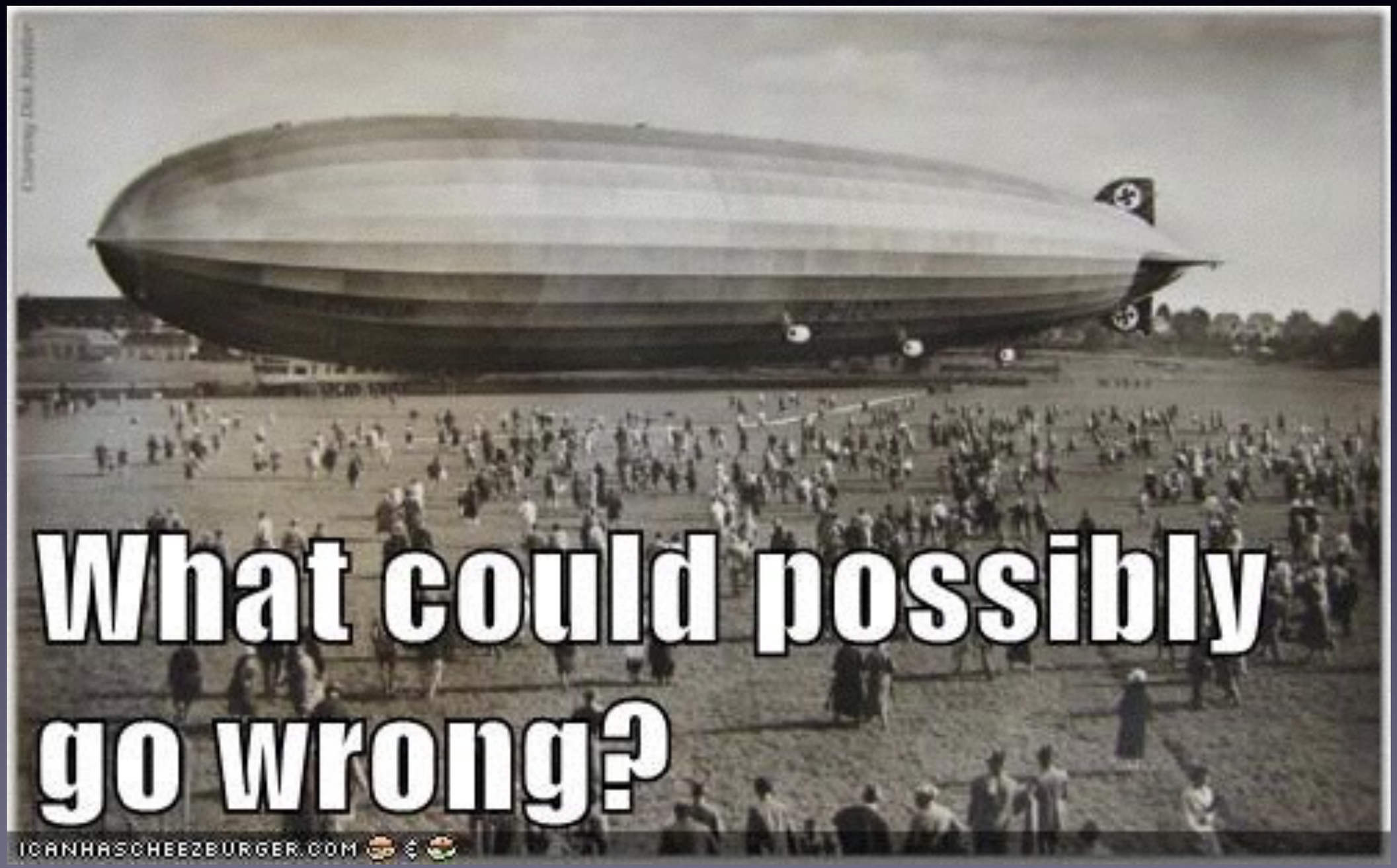
Docker

- Makes reasoning about services much easier

Docker

- Restful HTTP ‘manager’ daemon on each host
- CLI tools use the HTTP interface to the daemon

Demo: Basics



Registrator

- Go based
- Watches Docker daemon using long polling
- Updates a back-end: Etcd, Consul or SkyDns
- https://github.com/gliderlabs/registrator

What data do we have?

HostIp: 192.168.99.102

HostPort: 49212

ContainerPort: 3306

Protocol: TCP

Metadata

- Which service is this (name, tags)
- What protocol? How do I consume the service?

We can add them as docker labels!

Docker labels

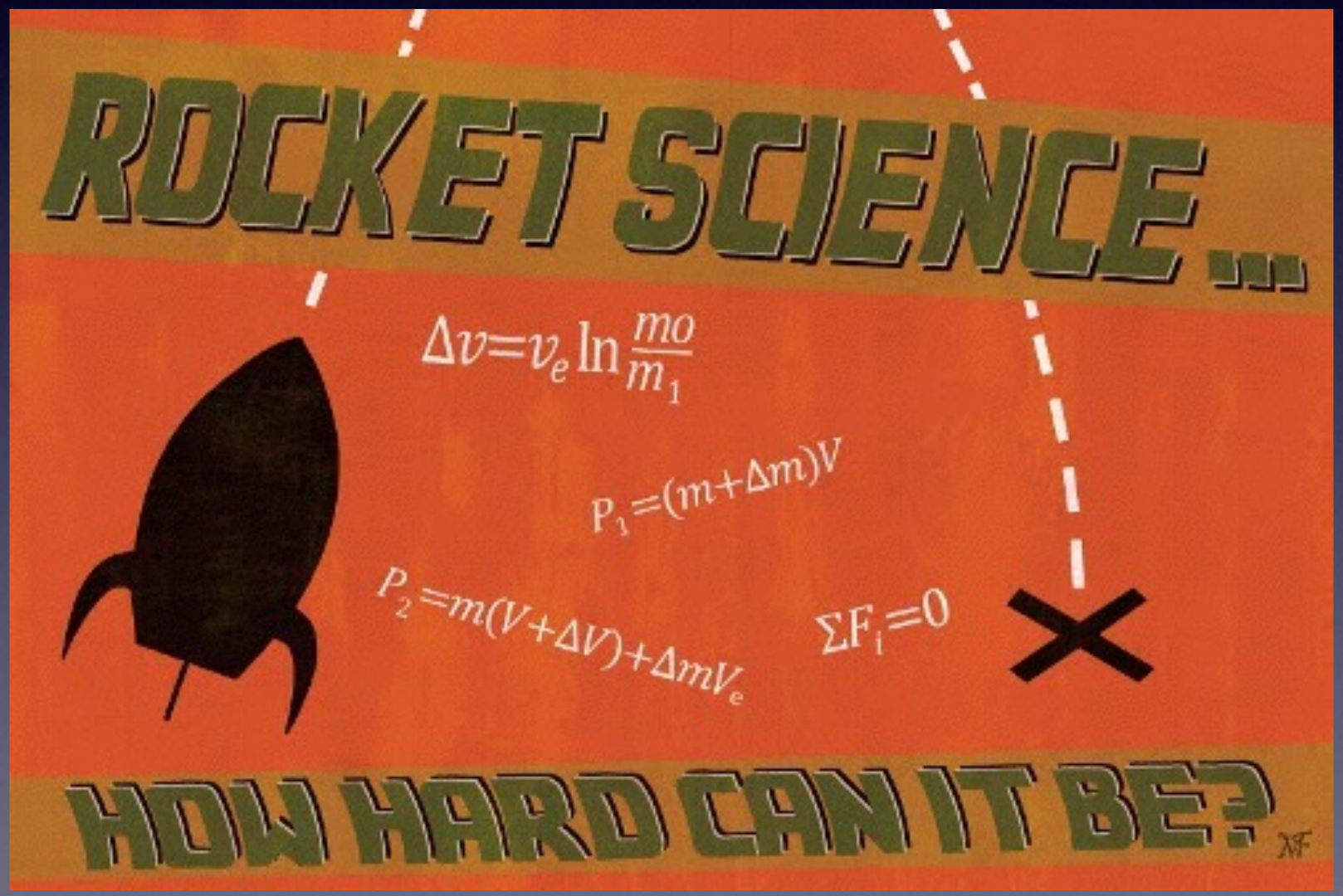
- Completely generic

```
docker run --name some-mysql \
-P \
-l SERVICE_NAME=my_mysql_instance \
-l SERVICE_TYPE=mysql \
-l SERVICE_TAGS=production
-l SERVICE_VERSION=5.5.1 \
--rm \
mysql
```

So suppose this thing works…

- We have a reliable, distributed service store
- We synchronize it with the Docker daemon(s)
- We can access it using an HTTP API
- It notifies us when it changes

This was the easy part



Dynamic Services

- Any service can just appear
- Multiple instances can appear
- Instances can disappear without warning
- Service cascades

OSGi

- Java based dynamic service framework
- Since 1999
- Initially for embedded systems
- Now very popular in cloud deployments

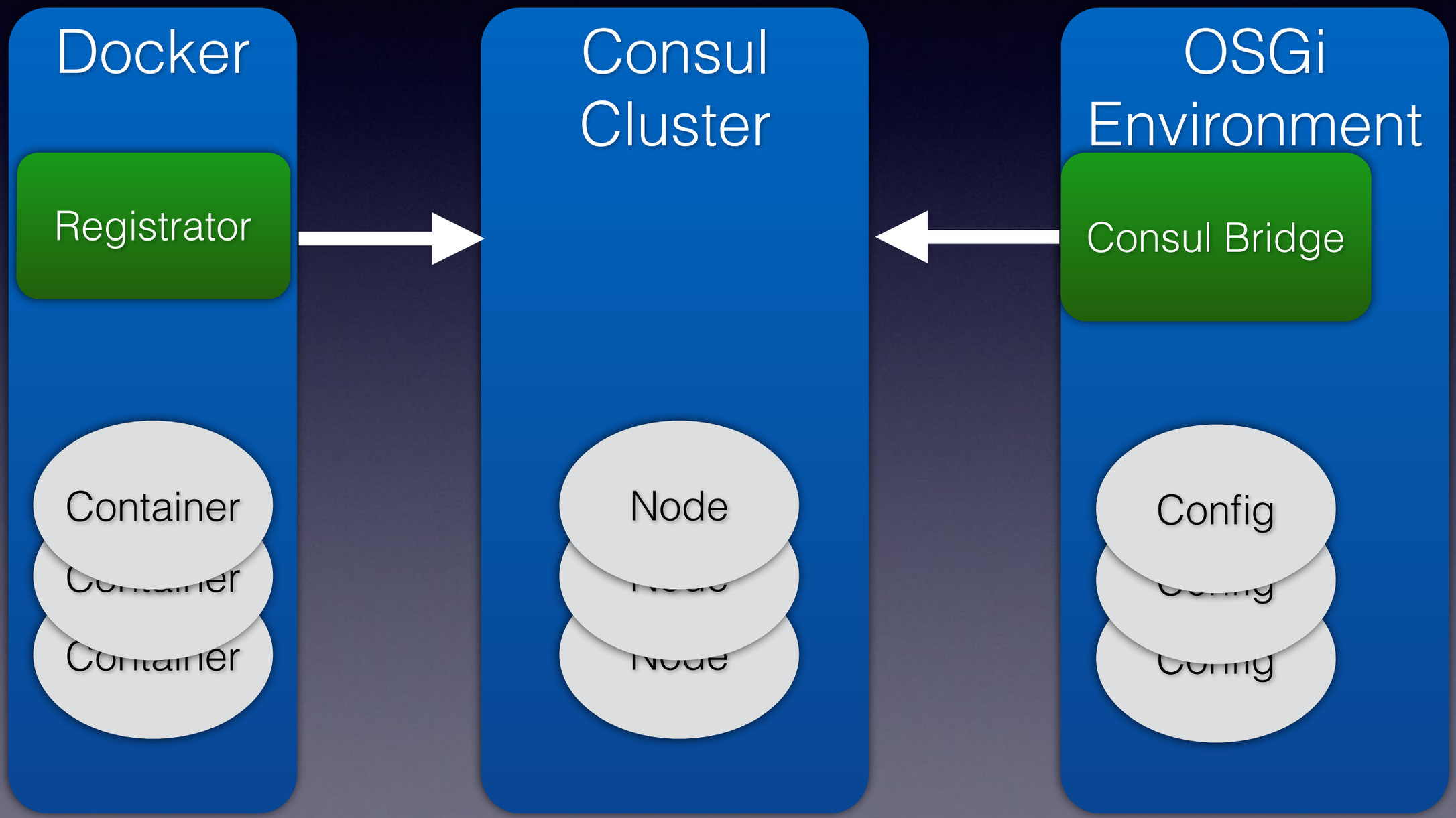
OSGi

- Central ‘Service Bus’
- Any Java object can be registered as a service
- Code can redeploy on the fly
- Services can come and go dynamically
- Services can depend on other services

Consul & OSGi

- Monitor Consul
- Inject services into OSGi

Consul


Docker
Registrator
Container
Container
Container
Consul
Cluster
Node
Node
Node
OSGi
Environment
Consul Bridge
Config
Config
Config

Docker & OSGi

- The Consul ‘monitor’ won’t know what a service is really about
- We need to separate configuration from the ‘driver’

Configuration Admin

- Create Configuration objects based on docker data
- Configuration object = PID + KV
- Leave the actual interpretation to ‘driver bundles’
- ‘Source agnostic’

Example

- Create a HTTP / JSON driver
- Insert documents into Elasticsearch

Final demo!

Service Checks

- Consul can add checks to a service registration
- HTTP API
- Script API
- Deadman switch
- If the check fails it will no longer report that service

Dynamic

- Truly embraces a dynamic system
- Matches the OSGi ‘way’

Really simple

- Registrator is a few hundred lines of Go code
- consul-osgi is a few hundred lines of Java

Scheduler agnostic

- It does not matter who / what started the containers

Unix philosophy

- Do one thing and one thing well
- Easily integrate with other tools

consul-OSGi is a hobby project!

- But turning it into something more would be cool

And I think it can benefit the OSGi community

Future work

- Conventions on metadata would be nice
- Better security model for Docker
- Use other sources of configuration like Kubernetes
- Downloadable drivers using mvn coordinates
- Consul backed load balancer (Vulcand)
- Integrate with secure storage

Thank you!

frank@dexels.com

Twitter @lyaruu

https://github.com/flyaruu/consul-osgi

blog: http://www.codemonkey.nl/