redhat.

Build and run reliable and scalable microservices using Docker, Kubernetes and Netflix OSS

OpenMunich 2016

Andreas Neeb
Chief Architect Financial Services
Red Hat GmbH

11. November 2016

Problem Statement: Running microservices architecures (MSA) at scale isn‘t easy.

Problem Statement: Running microservices architecures (MSA) at scale isn‘t easy.

Some requirements for distributed systems

Development / Deployment:

- Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- Dependency management
- Design for eventual consistency
- Artifact repositories

Runtime:

- Standardization
- Isolation
- Service Discovery
- Load Balancing
- Resiliency
- Health checks & automated recovery
- Distributed logging
- Tracing
- Infrastructure Monitoring

People try to copy Netflix, but they can only copy what they see. They copy the results, not the process.

Adrian Cockcroft, former Chief Cloud Architect, Netflix

Step 1: Docker

Container images are runnable packages that contain your applications and their dependencies. They are lighter than virtual machine images and can be layered with other Container images to re-use common content.

Isolated

Lightweight

Portable

Docker Containers provide standardization, automation and dependency management

Development / Deployment:

- 🐳 Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- 🐳 Dependency management
- Design for eventual consistency
- 🐳 Artifact repositories

Runtime:

- 🐳 Standardization
- 🐳 Isolation
- Service Discovery
- Load Balancing
- Resiliency
- Health checks & automated recovery
- Distributed logging
- Tracing
- Infrastructure Monitoring

Step 2 :Kubernetes

- Container Orchestration
- Bare-metal to multi cloud
- Based on 15 years Container Management at Google
- 100% Open source

„Manage applications, not machines“

OpenShift is Red Hats Kubernetes Distribution (plus much more ...)

Kubernetes adds vital capabilities to deploying, configuring and running MSAs

Development / Deployment:

- Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- Dependency management
- Design for eventual consistency
- Artifact repositories

Runtime:

- Standardization
- Isolation
- Service Discovery
- Load Balancing
- Resiliency
- Health checks & automated recovery
- Distributed logging
- Tracing
- Infrastructure Monitoring

OpenShift enhances Kubernetes with CI/CD, logging and monitoring capabilities

Development / Deployment:

- Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- Dependency management
- Design for eventual consistency
- Artifact repositories

Runtime:

- Standardization
- Isolation
- Service Discovery
- Load Balancing
- Resiliency
- Health checks & automated recovery
- Distributed logging
- Tracing
- Infrastructure Monitoring

Step 3 :Netflix Hystrix

- Latency and fault tolerance via third-party client libraries
- Stop cascading failures in a complex distributed system
- Fail fast and rapidly recover
- Fallback and gracefully degrade when possible
- Near real-time monitoring, alerting, and operational control

Netflix Hystrix adds fault- and latency tolerance plus monitoring

Development / Deployment:

- Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- Dependency management
- Design for eventual consistency
- Artifact repositories

Runtime:

- Standardization
- Isolation
- Service Discovery
- Load Balancing
- Resiliency
- Health checks & automated recovery
- Distributed logging
- Tracing
- Infrastructure Monitoring

Some examples

Development / Deployment:

- Automation
- Continuous Integration / Delivery
- Configuration Management
- Service / API design
- Rigorous Testing
- Dependency management
- Design for eventual consistency
- Artifact repositories

Runtime:

- Standardization
- Isolation
- **Service Discovery**
- **Load Balancing**
- **Resiliency**
- **Health checks & automated recovery**
- Distributed logging
- Tracing
- Infrastructure Monitoring

Service Discovery & Load Balance

Lots of stuff to figure out

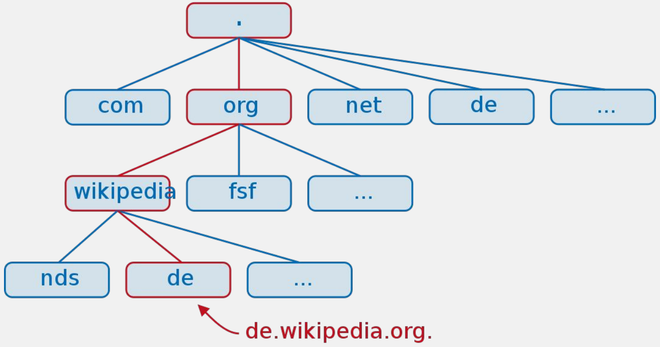
- Discovery Server?
- Consul, Eureka, Zookeeper, Etcd
- Client Libraries?
- Java, Node, Ruby, Go

```
<dependency>
    <groupId>com.netflix.eureka</groupId>
    <artifactId>eureka2-client</artifactId>
    <version>2.0.0-rc.2</version>
         </dependency>
```

What if we just use DNS?

- Comes free with (almost) any OS
- Simply works

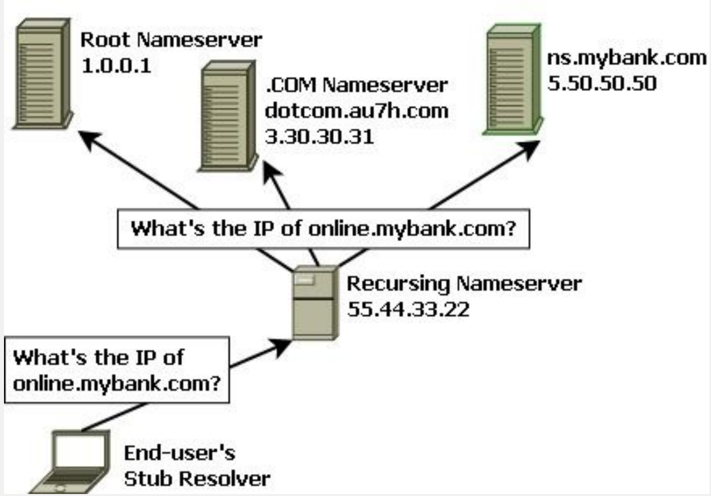
http://myservice



de.wikipedia.org.

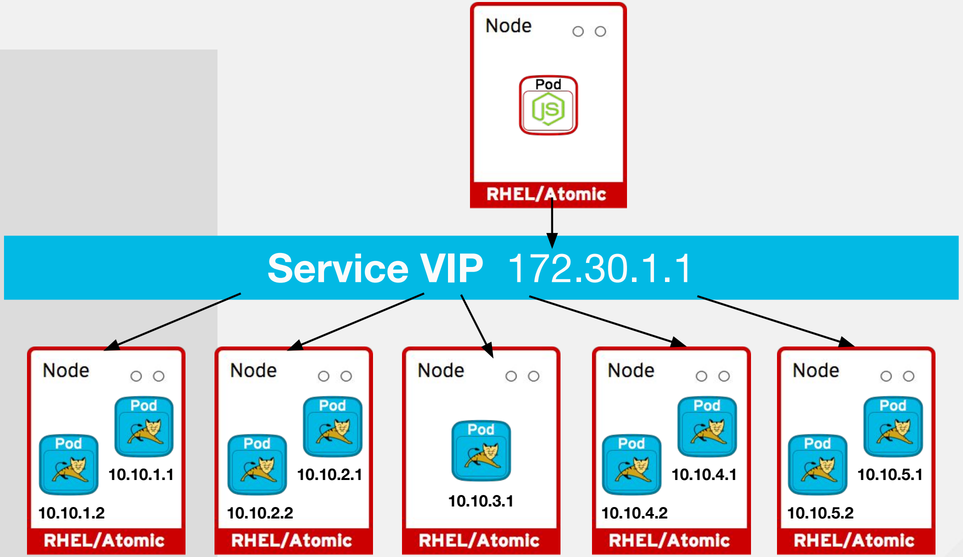
Maybe DNS sucks for elastic discovery?

- Scale out?
- Scale in?
- Caching?



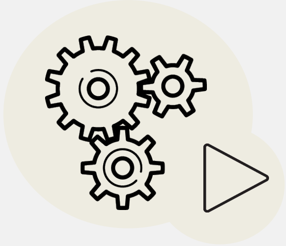
A better way: Kubernetes Services

```
kind: Service
apiVersion: v1
metadata:
    name: myservice
spec:
    selector: myfrontend
        app:
        ports:
            protocol: TCP
            port: 80
            targetPort: 8080
```



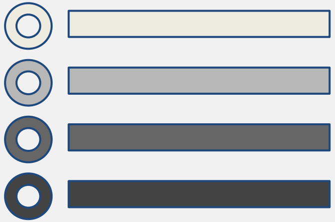
How does it work : Kubernetes Concepts

Pod



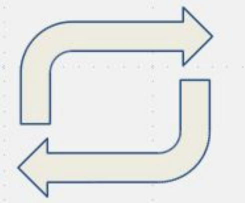
One or More Containers
Shared IP
Shared Storage Volume
Shared Resources
Shared Lifecycle

Label



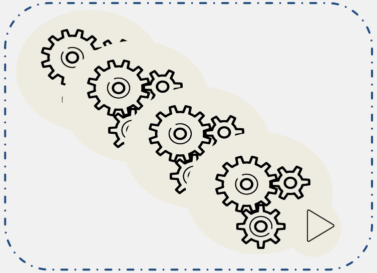
Key/Value pairs associated with Kubernetes objects (e.g. env=production)

Replication Controller



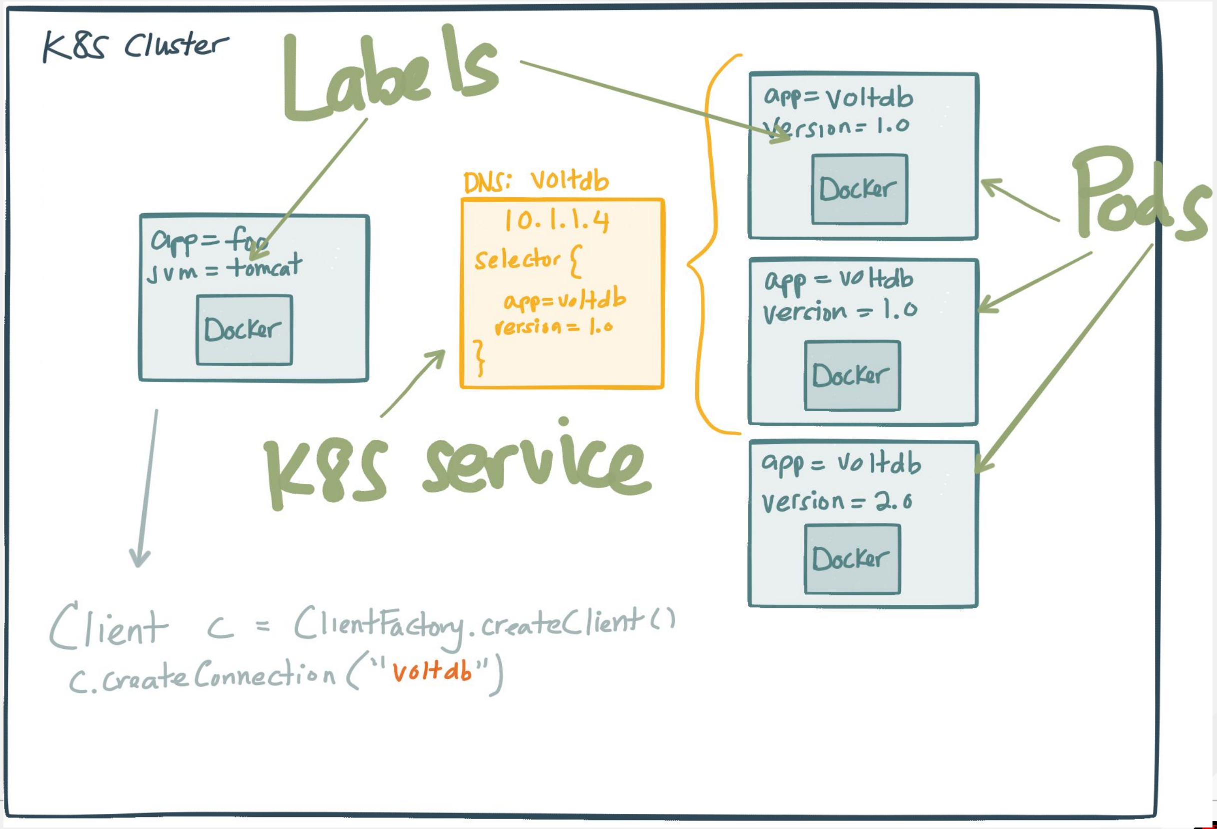
Ensures that a specified number of pod replicas are running at any one time

Service

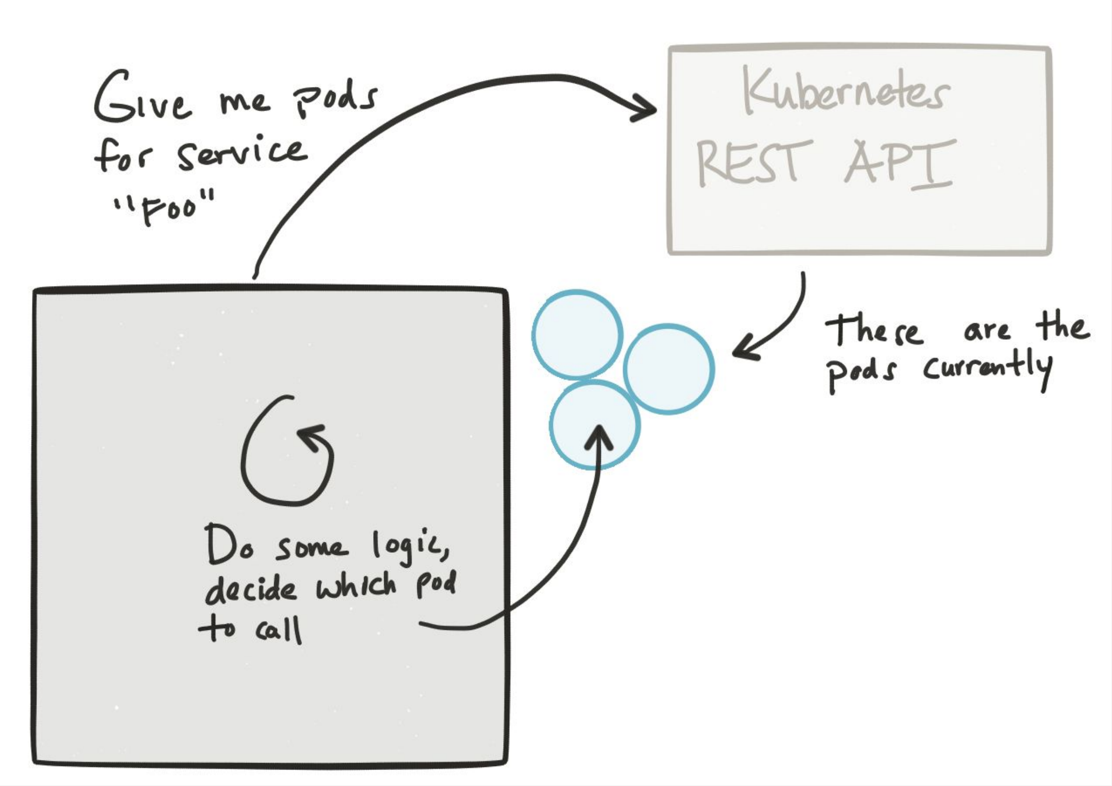


Grouping of pods, act as one, has stable virtual IP and DNS name

Bringing it all together



How about Client-Side Load Balancing?


Give me pods for service "Foo"
Kubernetes REST API
These are the pods currently
Do some logic, decide which pod to call

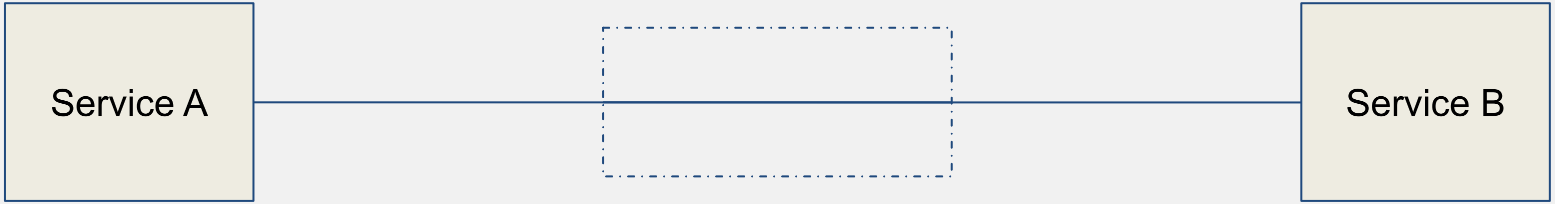
Service Discovery & Load Balancing

- Kubernetes allows for use of simple DNS for Service Discovery--> 95% use case
- Kube Services abstracts application details
- Load Balancing „out of the box“
- Open restAPI enables 5% use cases

Fault-tolerance

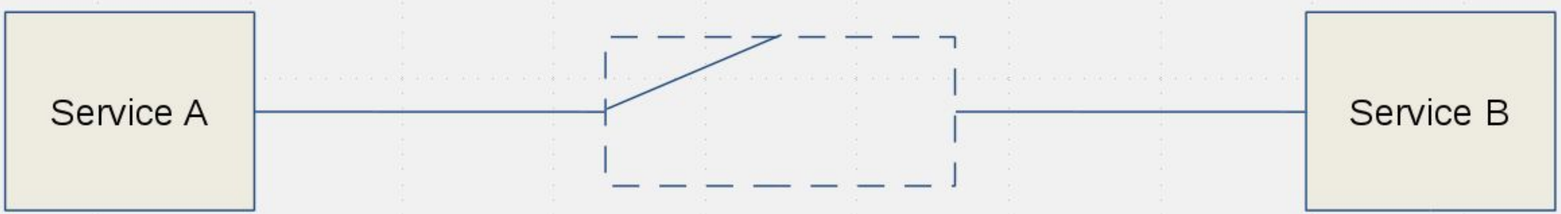
- Health Checks, Auto Recovery -> Kubernetes
- Application Resilience -> Netflix Hystrix / Kubeflix

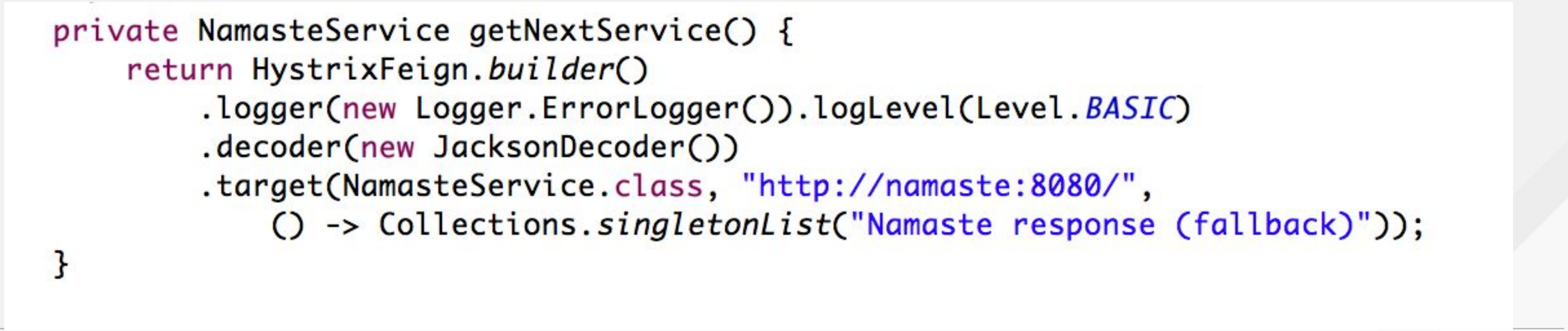
Resilient Application Design / Circuit Breaker



http://martinfowler.com/bliki/CircuitBreaker.html

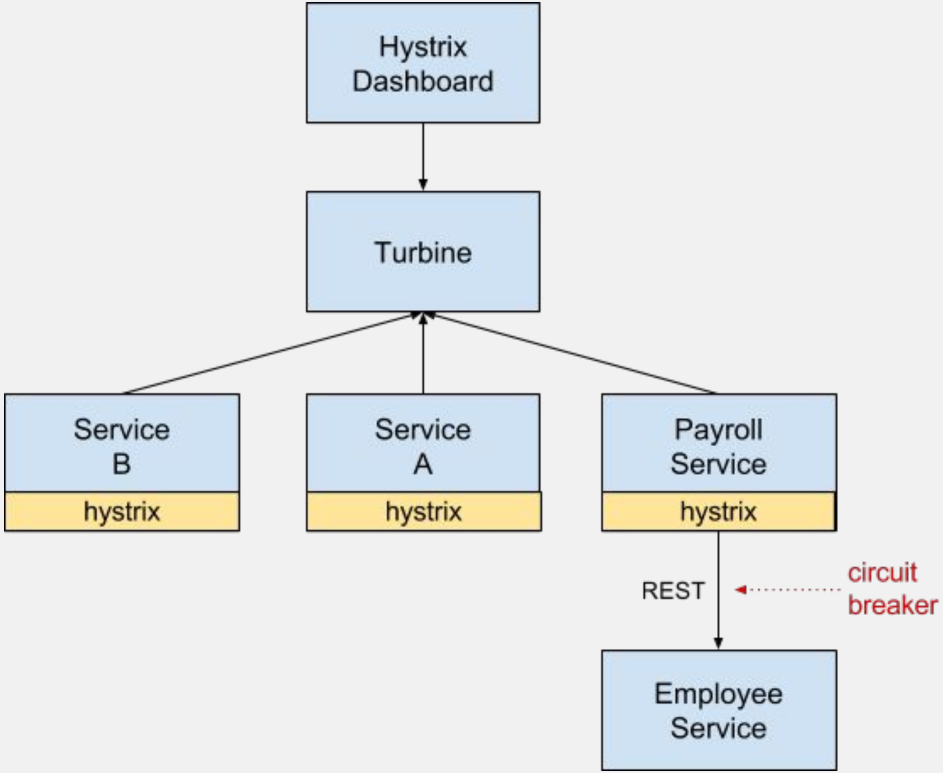
Netflix Hystrix

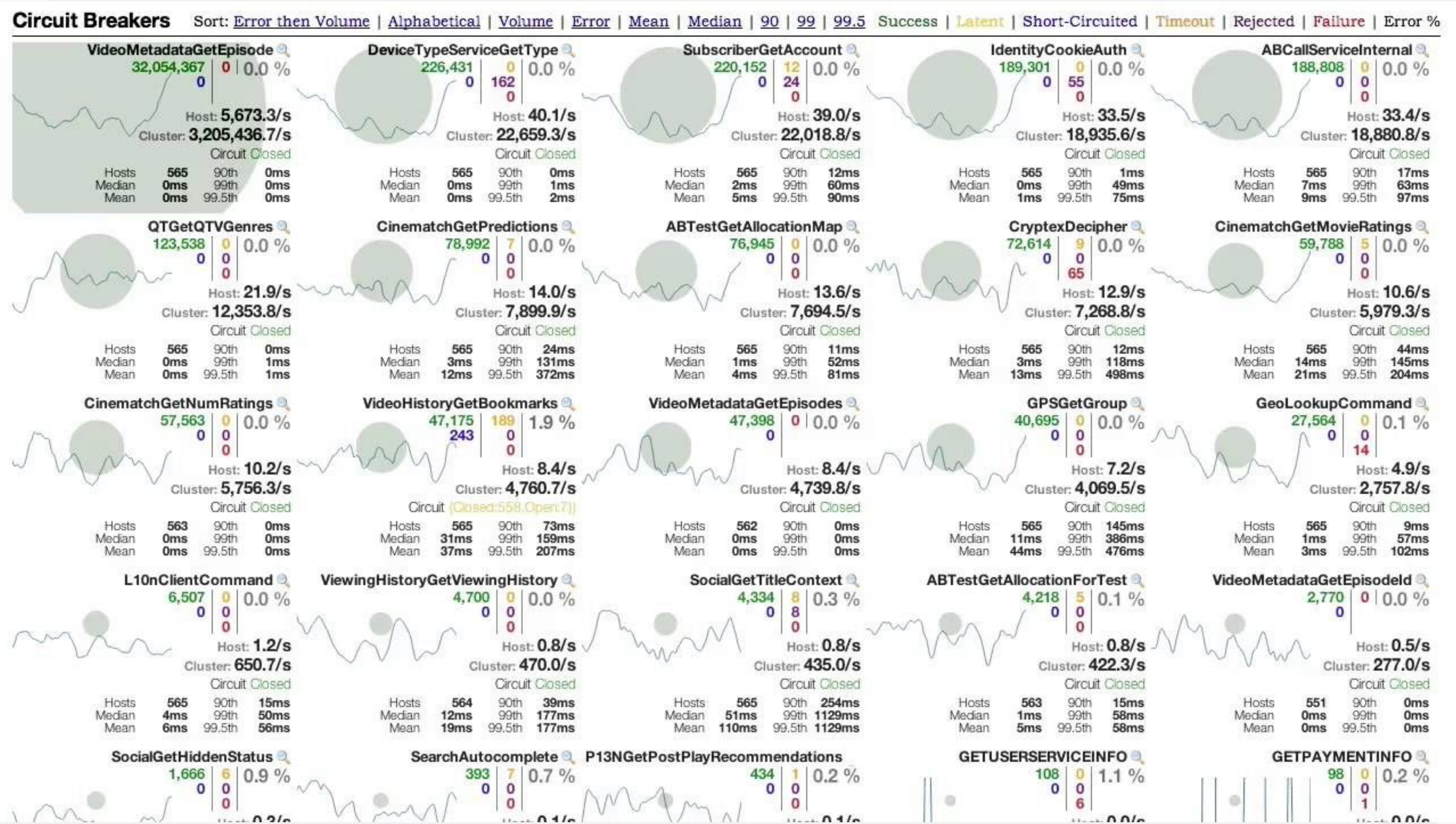




```
private NamasteService getNextService() {
    return HystrixFeign.builder()
        .logger(new Logger.ErrorLogger()).logLevel(Level.BASIC)
        .decoder(new JacksonDecoder())
        .target(NamasteService.class, "http://namaste:8080/",
            () -> Collections.singletonList("Namaste response (fallback)"));
}
```

Hystrix Dashboard & Turbine





Circuit Breakers Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5 Success | Latent | Short-Circuited | Timeout | Rejected | Failure | Error %

VideoMetadataGetEpisode
32,054,367 | 0 | 0.0 %
0
Host: 5,673.3/s
Cluster: 3,205,436.7/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

DeviceTypeServiceGetType
226,431 | 0 | 0.0 %
0 | 162
0
Host: 40.1/s
Cluster: 22,659.3/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 2ms

SubscriberGetAccount
220,152 | 12 | 0.0 %
0 | 24
0
Host: 39.0/s
Cluster: 22,018.8/s
Circuit Closed
Hosts 565 90th 12ms
Median 2ms 99th 60ms
Mean 5ms 99.5th 90ms

IdentityCookieAuth
189,301 | 0 | 0.0 %
0 | 55
0
Host: 33.5/s
Cluster: 18,935.6/s
Circuit Closed
Hosts 565 90th 1ms
Median 0ms 99th 49ms
Mean 1ms 99.5th 75ms

ABCallServiceInternal
188,808 | 0 | 0.0 %
0 | 0
0
Host: 33.4/s
Cluster: 18,880.8/s
Circuit Closed
Hosts 565 90th 17ms
Median 7ms 99th 63ms
Mean 9ms 99.5th 97ms

QTGetQTVGenres
123,538 | 0 | 0.0 %
0 | 0
0
Host: 21.9/s
Cluster: 12,353.8/s
Circuit Closed
Hosts 565 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 1ms

CinematchGetPredictions
78,992 | 7 | 0.0 %
0 | 0
0
Host: 14.0/s
Cluster: 7,899.9/s
Circuit Closed
Hosts 565 90th 24ms
Median 3ms 99th 131ms
Mean 12ms 99.5th 372ms

ABTestGetAllocationMap
76,945 | 0 | 0.0 %
0 | 0
0
Host: 13.6/s
Cluster: 7,694.5/s
Circuit Closed
Hosts 565 90th 11ms
Median 1ms 99th 52ms
Mean 4ms 99.5th 81ms

CryptexDecipher
72,614 | 9 | 0.0 %
0 | 0
65
Host: 12.9/s
Cluster: 7,268.8/s
Circuit Closed
Hosts 565 90th 12ms
Median 3ms 99th 118ms
Mean 13ms 99.5th 498ms

CinematchGetMovieRatings
59,788 | 5 | 0.0 %
0 | 0
0
Host: 10.6/s
Cluster: 5,979.3/s
Circuit Closed
Hosts 565 90th 44ms
Median 14ms 99th 145ms
Mean 21ms 99.5th 204ms

CinematchGetNumRatings
57,563 | 0 | 0.0 %
0 | 0
0
Host: 10.2/s
Cluster: 5,756.3/s
Circuit Closed
Hosts 563 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

VideoHistoryGetBookmarks
47,175 | 189 | 1.9 %
243 | 0
0
Host: 8.4/s
Cluster: 4,760.7/s
Circuit (Closed:558,Open:7))
Hosts 565 90th 73ms
Median 31ms 99th 159ms
Mean 37ms 99.5th 207ms

VideoMetadataGetEpisodes
47,398 | 0 | 0.0 %
0
Host: 8.4/s
Cluster: 4,739.8/s
Circuit Closed
Hosts 562 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

GPSGetGroup
40,695 | 0 | 0.0 %
0 | 0
0
Host: 7.2/s
Cluster: 4,069.5/s
Circuit Closed
Hosts 565 90th 145ms
Median 11ms 99th 386ms
Mean 44ms 99.5th 476ms

GeoLookupCommand
27,564 | 0 | 0.1 %
0 | 0
14
Host: 4.9/s
Cluster: 2,757.8/s
Circuit Closed
Hosts 565 90th 9ms
Median 1ms 99th 57ms
Mean 3ms 99.5th 102ms

L10nClientCommand
6,507 | 0 | 0.0 %
0 | 0
0
Host: 1.2/s
Cluster: 650.7/s
Circuit Closed
Hosts 565 90th 15ms
Median 4ms 99th 50ms
Mean 6ms 99.5th 56ms

ViewingHistoryGetViewingHistory
4,700 | 0 | 0.0 %
0 | 0
0
Host: 0.8/s
Cluster: 470.0/s
Circuit Closed
Hosts 564 90th 39ms
Median 12ms 99th 177ms
Mean 19ms 99.5th 177ms

SocialGetTitleContext
4,334 | 8 | 0.3 %
0 | 8
0
Host: 0.8/s
Cluster: 435.0/s
Circuit Closed
Hosts 565 90th 254ms
Median 51ms 99th 1129ms
Mean 110ms 99.5th 1129ms

ABTestGetAllocationForTest
4,218 | 5 | 0.1 %
0 | 0
0
Host: 0.8/s
Cluster: 422.3/s
Circuit Closed
Hosts 563 90th 15ms
Median 1ms 99th 58ms
Mean 5ms 99.5th 58ms

VideoMetadataGetEpisodeId
2,770 | 0 | 0.0 %
0
Host: 0.5/s
Cluster: 277.0/s
Circuit Closed
Hosts 551 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

SocialGetHiddenStatus
1,666 | 6 | 0.9 %
0 | 0
0
Host: 0.3/s

SearchAutocomplete
393 | 7 | 0.7 %
0 | 0
0
Host: 0.1/s

P13NGetPostPlayRecommendations
434 | 1 | 0.2 %
0 | 0
0
Host: 0.1/s

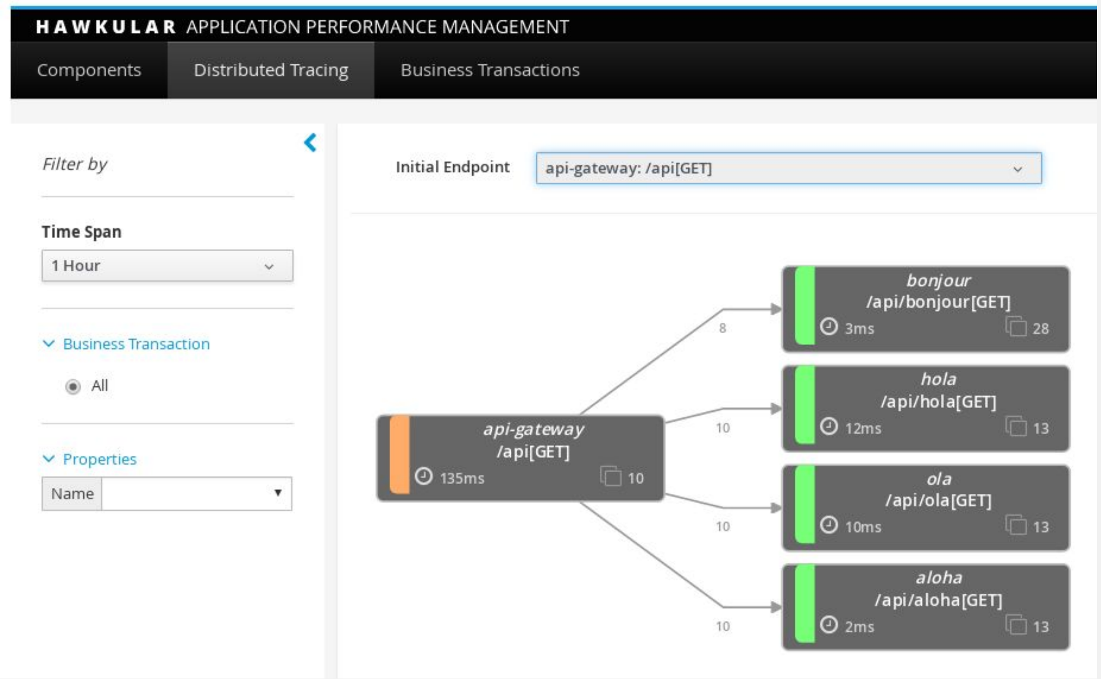
GETUSERSERVICEINFO
108 | 0 | 1.1 %
0 | 0
6
Host: 0.0/s

GETPAYMENTINFO
98 | 0 | 0.2 %
0 | 0
1
Host: 0.0/s

Fault-tolerance

- Kubenetes offers self healing
- Application readiness and health checks „teach“ the platform about your application
- Hystrix allows to build resilient applications on top of a resilient platform
- Combining Kubenetes primitives (labels) and Hystrix allows for easy monitoring

Distributed Tracing: Zipkin + Hawkular APM



Time Span: 1 Hour

Name:

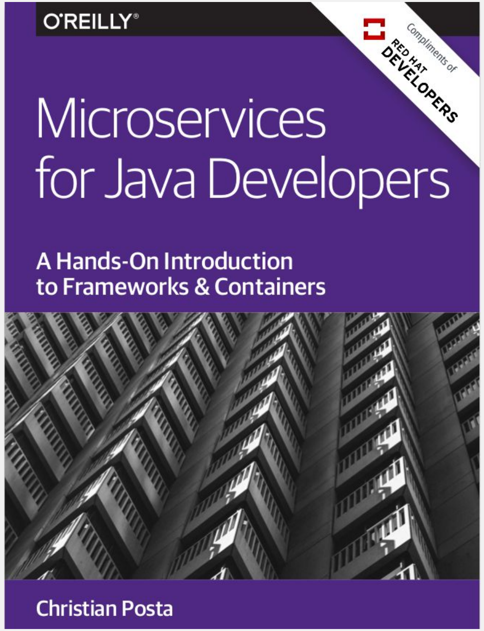
Initial Endpoint: api-gateway: /api[GET]

Summary

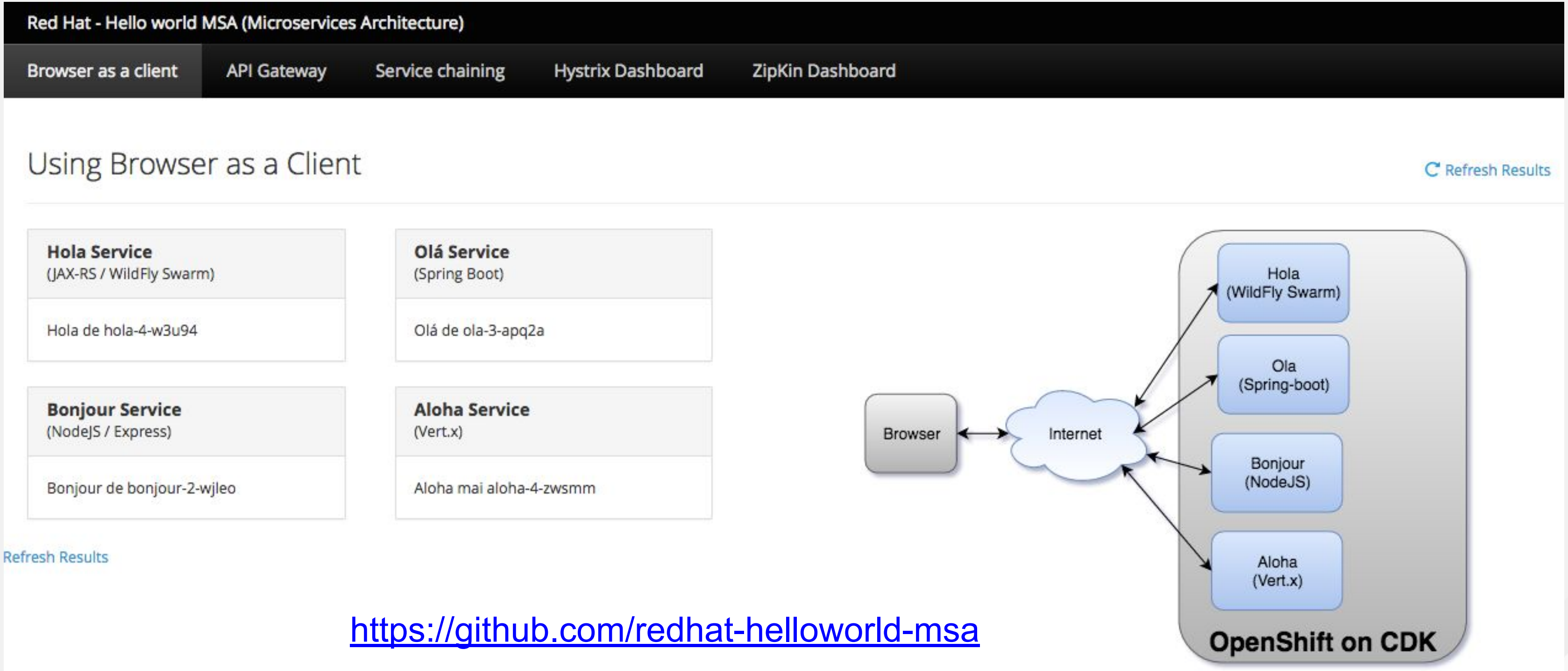
- Yes, distributed systems are inherently complex, but
- There are standard solutions available
- Docker, Kubernetes, OpenShift, Netflix OSS
- A smart platform helps keeing your apps and services simple and focused on what they're supposed to do
- Open Source is at the heart of successfull MSA implementations
- KISS: Keep it simple (95% use case!)

References

http://developers.redhat.com
http://openshift.com
http://blog.openshift.com
https://github.com/redhat-helloworld-msa
https://github.com/fabric8io/kubeflix
https://github.com/fabric8io/spring-cloud-kubernetes


O'REILLY
Compliments of
RED HAT DEVELOPERS
Microservices for Java Developers
A Hands-On Introduction to Frameworks & Containers
Christian Posta

“Show me the code” -- Demo Time!



Red Hat - Hello world MSA (Microservices Architecture)

Using Browser as a Client

Hola Service
(JAX-RS / WildFly Swarm)

Hola de hola-4-w3u94

Olá Service
(Spring Boot)

Olá de ola-3-apq2a

Bonjour Service
(NodeJS / Express)

Bonjour de bonjour-2-wjleo

Aloha Service
(Vert.x)

Aloha mai aloha-4-zwsmm

https://github.com/redhat-helloworld-msa

OpenMunich 2016

redhat.

THANK YOU

- plus.google.com/+RedHat
- facebook.com/redhatinc
- linkedin.com/company/red-hat
- twitter.com/RedHatNews
- youtube.com/user/RedHatVideos