



RED HAT
OPEN SOURCE DAY

Europe, Middle East & Africa



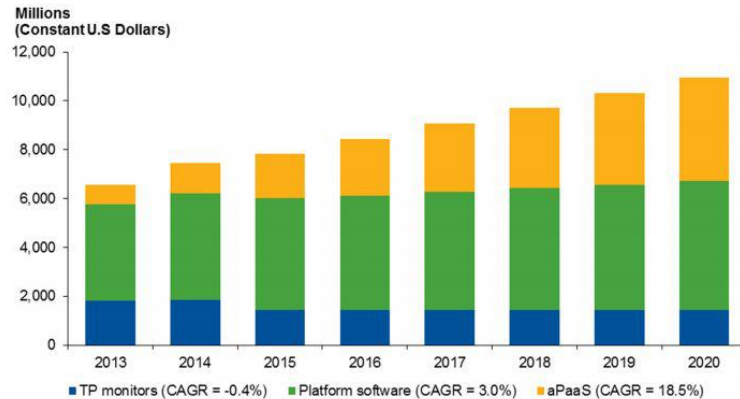
MICROSERVICES 2.0 & RHOAR

Cloudify your applications: Microservices and beyond

Ugo Landini
Solution Architect

Samuele Dell'Angelo
Solution Architect

State of the Market



TP = transaction processing; CAGR = compound annual growth rate; aPaaS = application platform as a service

Source: Gartner (November 2016)

2015 AP Revenue (Gartner, Nov. 2016) :

- Oracle -4.5%
- IBM -9.5%
- Red Hat +33.3%
- Amazon +50.6%
- Pivotal +22.7%

“Resist the temptation to simply lift and shift Java EE applications from closed-source to open-source application servers for modest license savings. If you are contemplating porting an application, consider rearchitecting it to be cloud-native and moving it to aPaaS - presuming that business drivers warrant the investment.”

Gartner (November 2016)

ThoughtWorks®

TECHNOLOGY
RADAR

● HOLD

45.Application Servers new

46.OSGi

47.SPDY new

*“Most teams we work with favor bundling an embedded http server within your web application. There are plenty of options available: Jetty, SimpleWeb, Webbit and Owin Self-Host amongst others. Easier automation, easier deployment and a reduction in the amount of infrastructure you have to manage lead us to **recommend embedded servers over application servers for future projects**”*

ThoughtWorks Technology Radar, May 2015

MICROSERVICES 101



Microservices defined

“... is an approach to developing a single application as a **suite of small services**, each running in its **own process** and communicating with **lightweight mechanisms**, often an HTTP resource API. These services are built around **business capabilities** and **independently deployable** by **fully automated deployment** machinery. There is a **bare minimum of centralized management** of these services, which may be written in **different programming languages** and use **different data storage technologies**.”

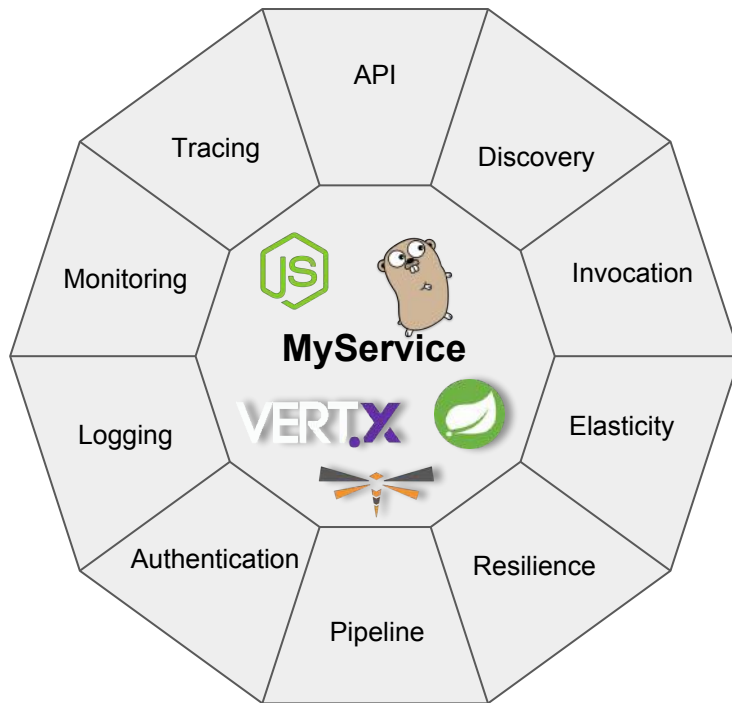
Martin Fowler

<http://martinfowler.com/articles/microservices.html>

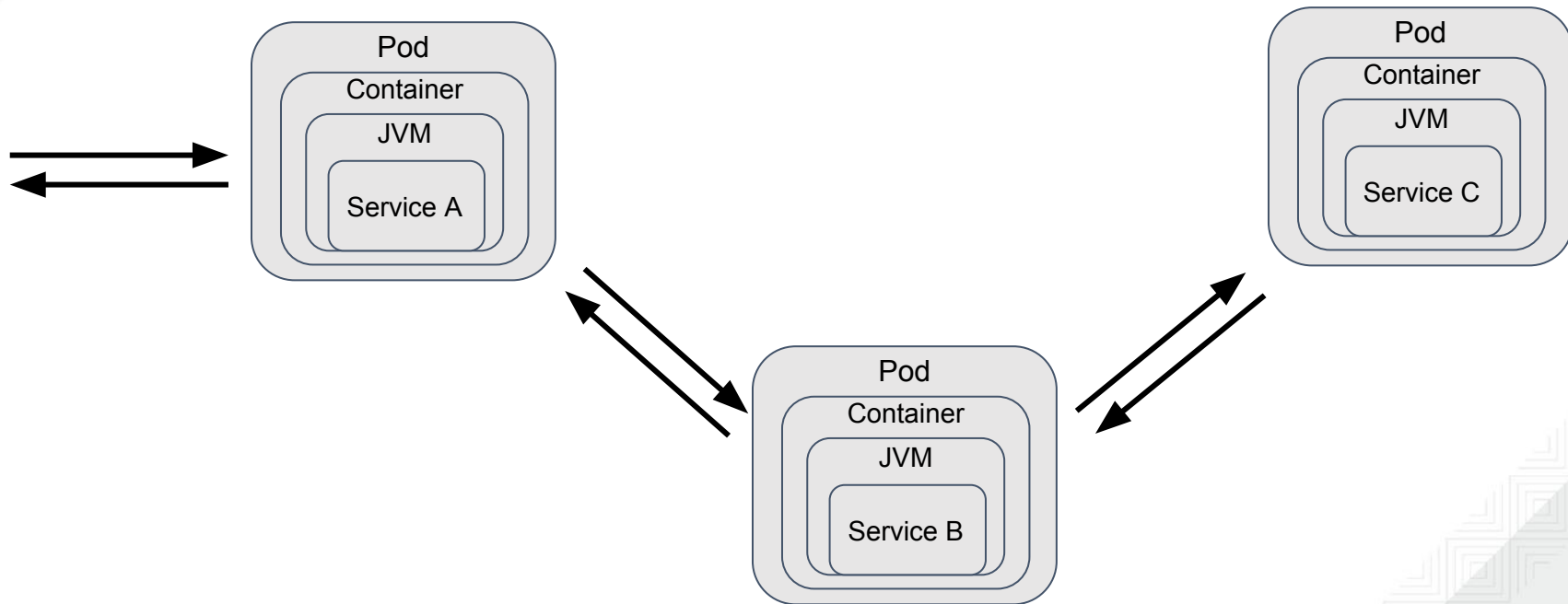
Microservices 101

- Small single-purpose services driven from **DDD (Domain Driven Design)** or practical decomposition of an existing application or existing SOA-style **mini-services**
- **Combined** to form a system or application
- **Independently deployable** (replaceable)
- **Independently scalable**
- **Antifragile** - increased robustness and resilience under pressure
- **Fully automated** software delivery
- **Polyglot** (language and framework independence)
- **API / Contract Focused**
- Typically **event-driven**
- **Decentralized** data management

Microservices 101



Microservices == Distributed Computing



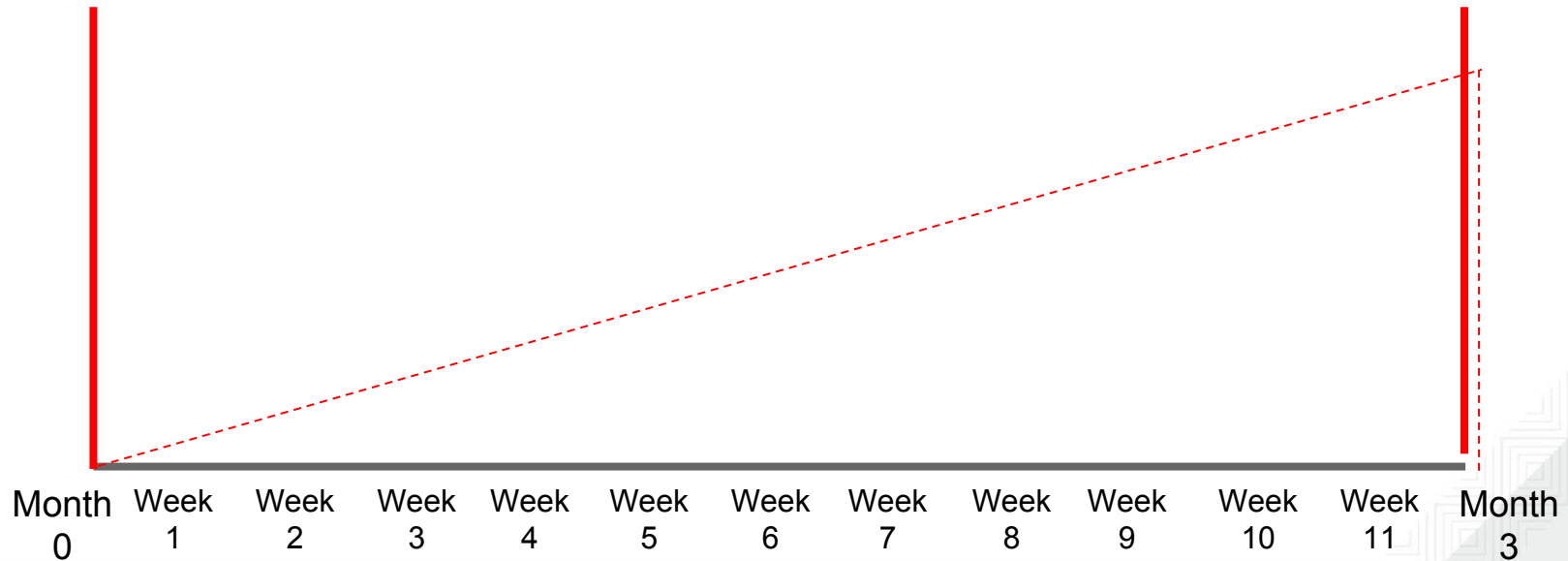
Wait, but weren't we already doing this distributed stuff...

- ... what about CORBA?
- ... and RMI?
- ... EJB?
- ... SOA?

What's the difference?

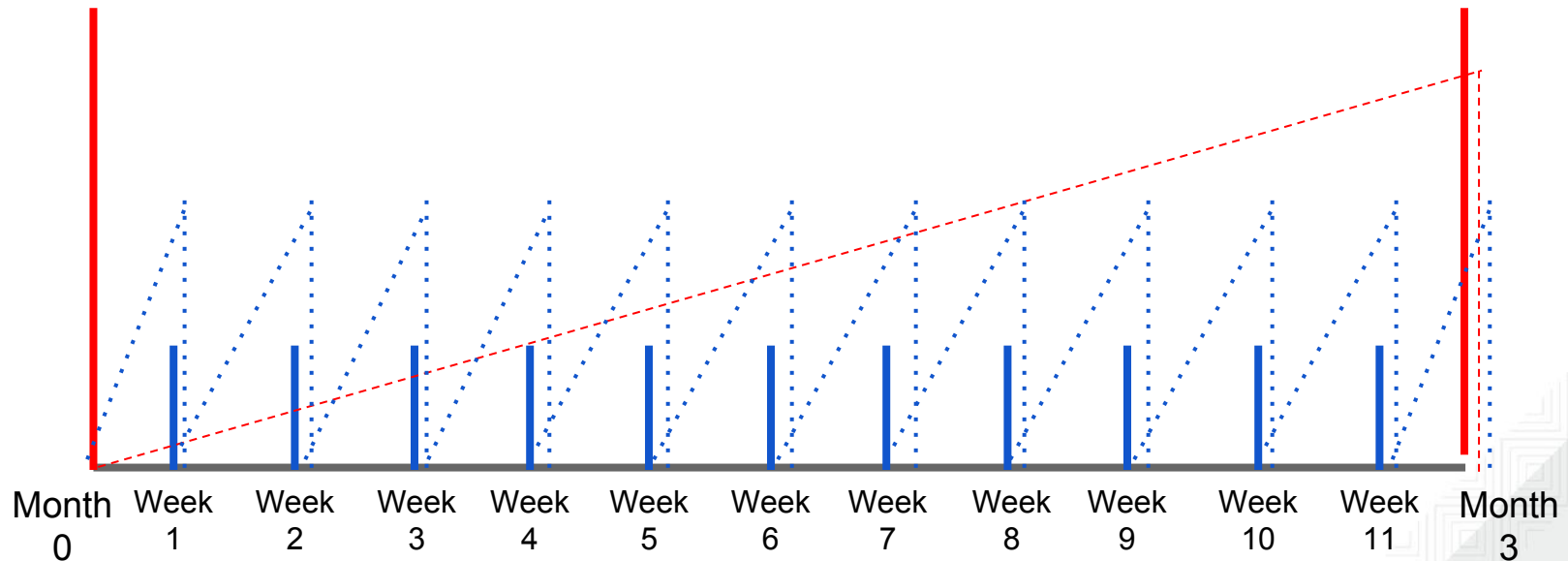
Maturing the Application LifeCycle

Monolith Java EE Lifecycle



Maturing the Application LifeCycle

Monolith Java EE Lifecycle
Fast Moving Java EE Monolith

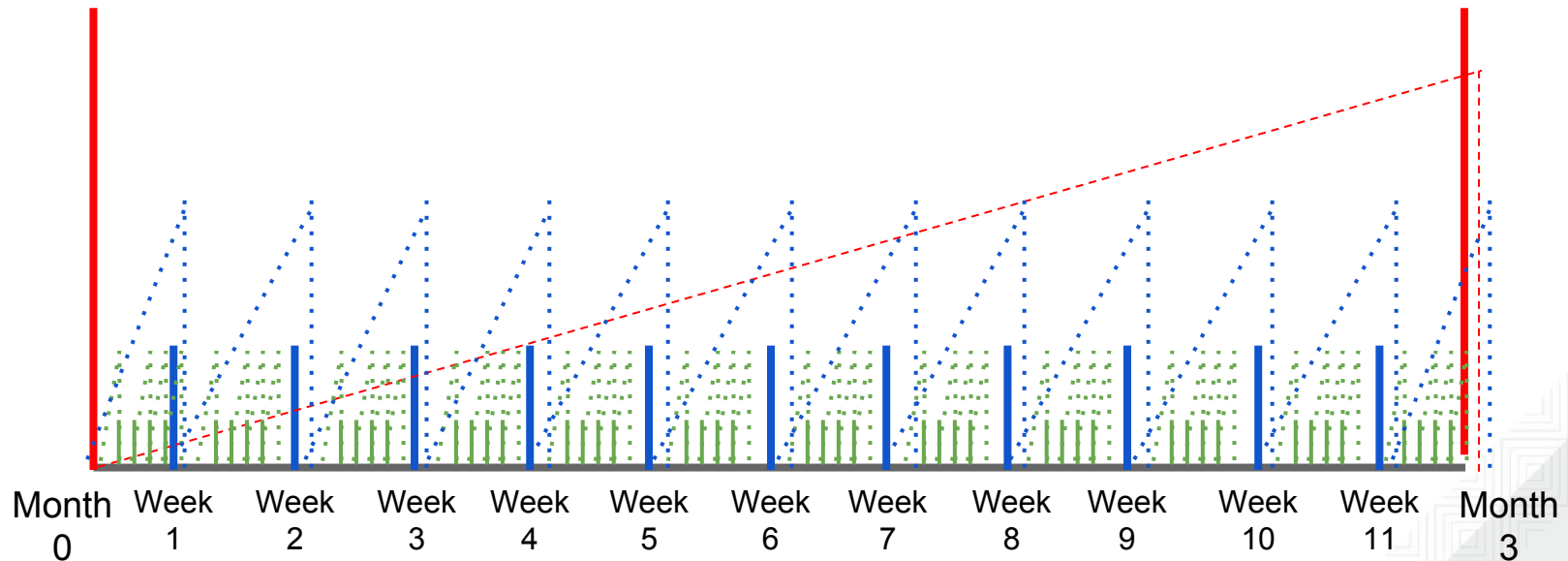


Maturing the Application LifeCycle

Monolith Java EE Lifecycle

Fast Moving Java EE Monolith

Java EE Microservices



What's the difference?

- Same ideas, new technologies (which will evolve in the future)
- But now, we should be able to bring a new feature in **production** in a few minutes

Microservices: the Good, the Bad...

The Good

- Domain-Driven Design
- Low coupling, high cohesion
- APIs and contracts
- Agile software development
- Full lifecycle automation
- Dev and Ops working together
- Common packaging / container format
- Rethinking Data



Microservices: the Good, the Bad...

The Bad

- Too much Dogma / CS purity
- Tradeoff between Agility & Operational Complexity
- Magnificent Monoliths and Stupendous SOA are not necessarily bad
- Microservices / Unicorn Envy
- Not all organizations can afford the **skills** and talent required to be successful
- Maintaining **data consistency** is hard in distributed systems



Microservices: the Good, the Bad...

The Ugly

- Building large scale distributed systems is **really** hard
- **Monitoring / APM** tools need to catch up
- Heterogeneity (languages, frameworks, data stores)
- Event-based, asynchronous, reactive programming is still in it's infancy and skills are **rare**
- **CAP: Consistency, Availability, Partition Tolerance**
? – choose two



Microservices Recommendations

- Understand and state your goals
- Understand the **tradeoffs**
- Start with People, Process and Culture
 - Agile Dev / DevOps is a prerequisite
- Invest in **automation** (provisioning, CI/CD, etc.)
- Start **small**
 - Small non-mission-critical green-field
 - Decomposition of existing monolith
- Get help - eg. **Red Hat Innovation Labs**

Java Microservices Platform (2014)



Config Server



ZIPKIN



NETFLIX Ribbon



HYSTRIX
DEFEND YOUR APP

CLOUD FOUNDRY

Why these components?



Eureka is the Service Registry where the clients lookup for service locations a.k.a Service Discovery



Config Server

Config Server externalized the Configuration



Ribbon is the client side Load Balancer



Hystrix is the Circuit Breaker



Zipkin is the Distributed Tracer



Zuul is the smart proxy purely based on Java

Why these components?



Eureka is the Service Registry where the clients lookup for service locations a.k.a Service Discovery



Config Server externalized the Configuration



Ribbon is the client side Load Balancer



Hystrix is the Circuit Breaker



Zipkin is the Distributed Tracer



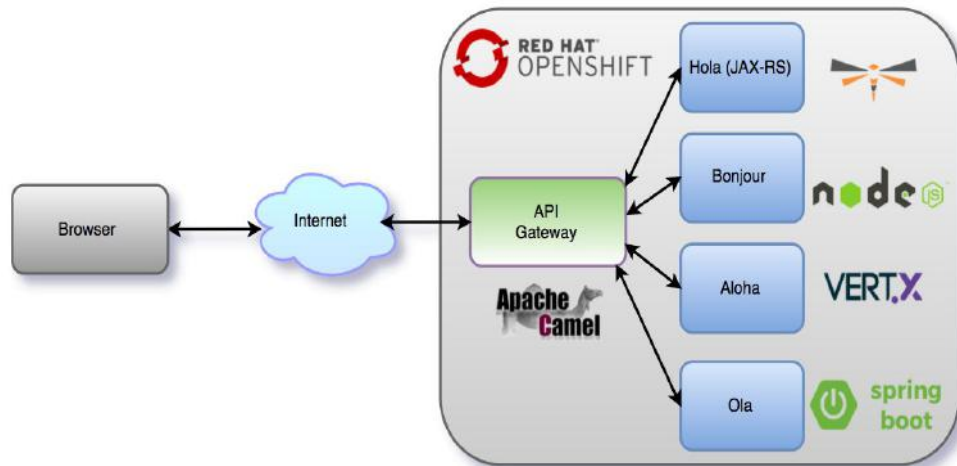
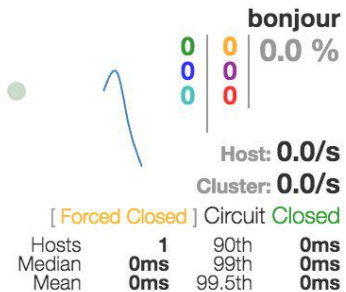
Zuul is the smart proxy purely based on Java



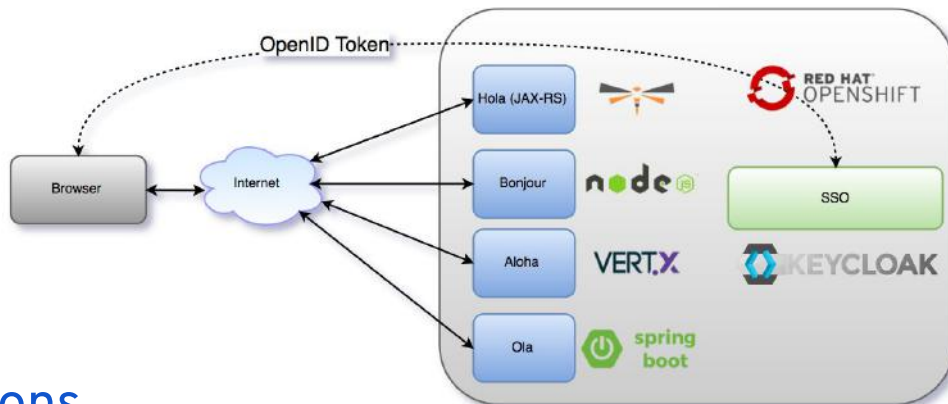
HYSTRIX

DEFEND YOUR APP

Success | Short-Circuited | Bad Request | Timeout | Rejected | Failure | Error %

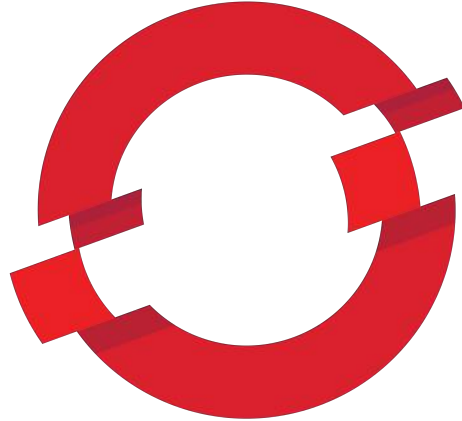


2.0



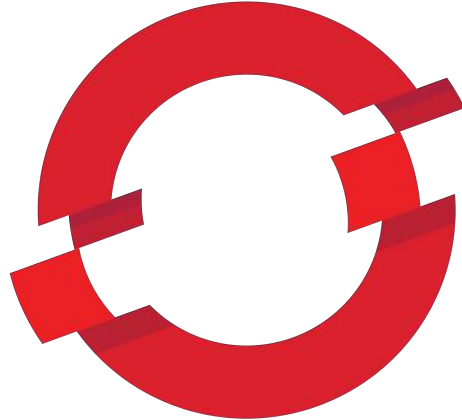
bit.ly/msa-instructions

Better Microservices Platform (2016/2017)



OPENSIFT

Even Better Microservices Platform (2018)



OPENSIFT

Istio



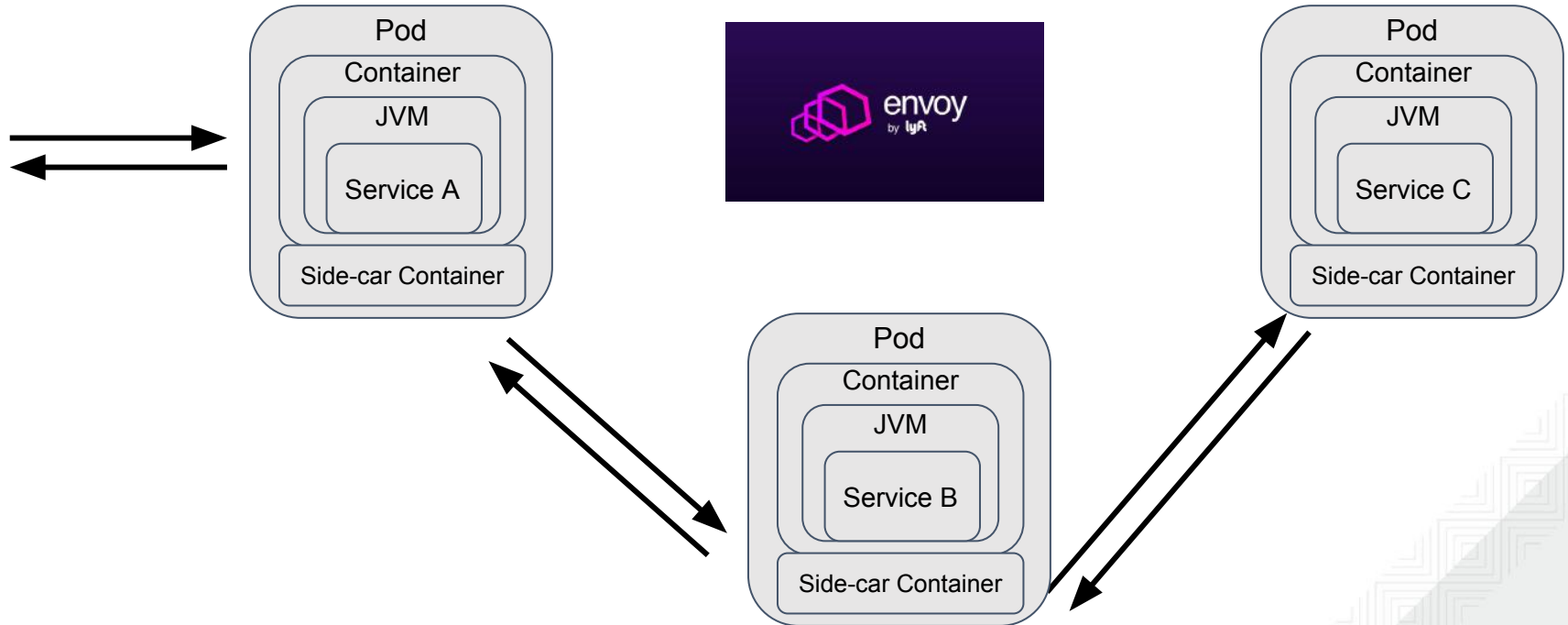
Istio - Sail

(Kubernetes - Helmsman or ship's pilot)

Sidecar?



Pods with 2 containers!



Infrastructure cluttering your code?

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zuul</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```



Istio

Intelligent Routing and Load Balancing

Control traffic between services with dynamic route configuration.

Conduct A/B tests, release canaries, and gradually upgrade versions using red/black deployments.

Resilience Across Languages and Platforms

Increase reliability by shielding applications from flaky networks and cascading failures in adverse conditions.

Telemetry and Reporting

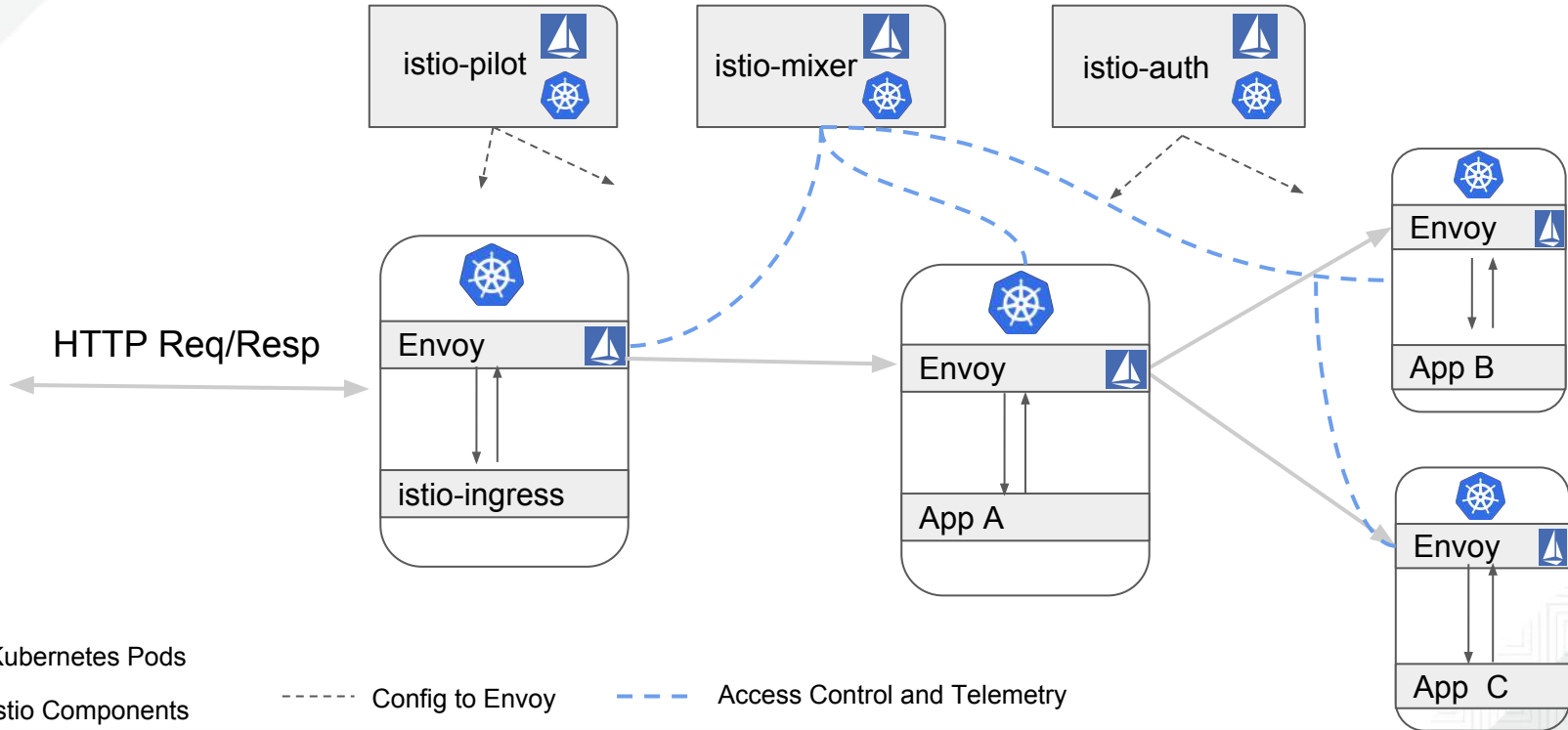
Understand the dependencies between services, the nature and flow of traffic between them, and quickly identify issues with distributed tracing.


Policy Enforcement

Apply organizational policy to the interaction between services, ensure access policies are enforced and resources are fairly distributed among consumers.

Istio Service Mesh

Currently upstream only



 Kubernetes Pods

 Istio Components

----- Config to Envoy

----- Access Control and Telemetry

Istio Components

- Control Plane
 - Istio-Pilot - istioctl, API, config
 - Istio-Mixer - Quota, Telemetry, Rate Limiting, ACL
 - Istio-Auth - TLS and Certificates
- Data Plane
 - Envoy proxy deployed as “side-cars” with applications

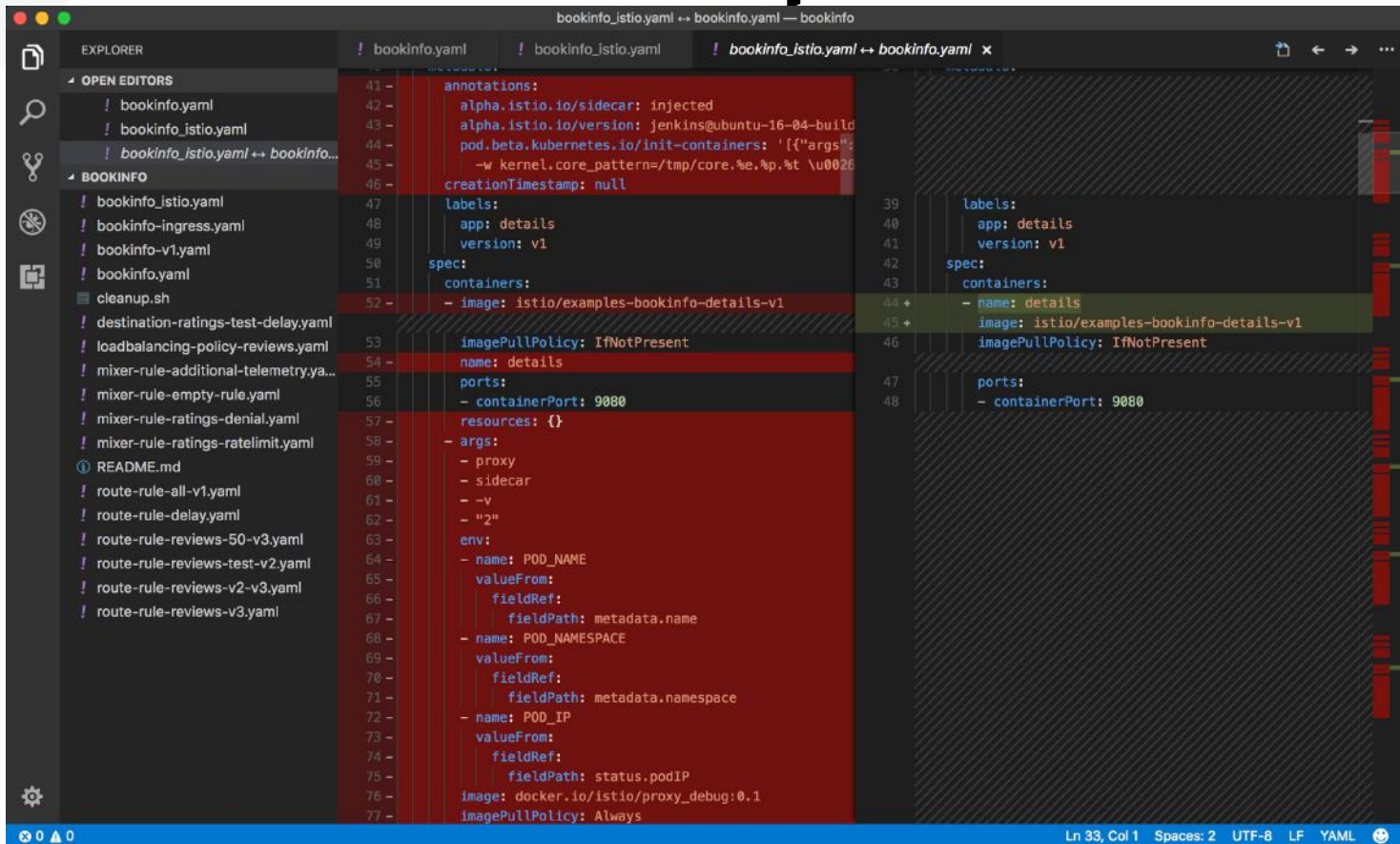
Circuit Breakers

Before Istio	After Istio
Boiler plate code	No code related to circuit breaking mixed with business logic
Multiple libraries and dependencies e.g. Hystrix	No libraries
Separate dashboard to collect circuit breaker e.g. Hystrix Turbine	All metrics can be collected and displayed in Grafana without extra bit of code
	Define circuit breakers using Kubernetes Tags

Tracing

Before Istio	After Istio
Boiler plate code	No code related to tracing mixed with business logic
Multiple libraries and dependencies e.g. Zipkin	No libraries

All in one place



The screenshot shows a code editor with a diff view. The left pane shows the original file, and the right pane shows the modified file. The diff highlights changes in a red background. The Explorer sidebar on the left lists various files, including 'bookinfo_istio.yaml' and 'bookinfo.yaml'. The status bar at the bottom indicates 'Ln 33, Col 1 Spaces: 2 UTF-8 LF YAML'.

```
41 - annotations:
42 -   alpha.istio.io/timeout: injected
43 -   alpha.istio.io/version: jenkins@ubuntu-16-04-build
44 -   pod.beta.kubernetes.io/init-containers: '[{"args":
45 -     -w kernel.core_pattern=/tmp/core.%.%.%.%t \u0026
46 -   creationTimestamp: null
47 -   labels:
48 -     app: details
49 -     version: v1
50 -   spec:
51 -     containers:
52 -     - image: istio/examples-bookinfo-details-v1
53 -     imagePullPolicy: IfNotPresent
54 -     name: details
55 -     ports:
56 -     - containerPort: 9080
57 -     resources: {}
58 -     args:
59 -     - proxy
60 -     - sidecar
61 -     - -v
62 -     - "2"
63 -     env:
64 -     - name: POD_NAME
65 -       valueFrom:
66 -         fieldRef:
67 -           fieldPath: metadata.name
68 -     - name: POD_NAMESPACE
69 -       valueFrom:
70 -         fieldRef:
71 -           fieldPath: metadata.namespace
72 -     - name: POD_IP
73 -       valueFrom:
74 -         fieldRef:
75 -           fieldPath: status.podIP
76 -     image: docker.io/istio/proxy_debug:0.1
77 -     imagePullPolicy: Always
```

```
39 labels:
40   app: details
41   version: v1
42 spec:
43   containers:
44 +   - name: details
45 +     image: istio/examples-bookinfo-details-v1
46     imagePullPolicy: IfNotPresent
47   ports:
48     - containerPort: 9080
```

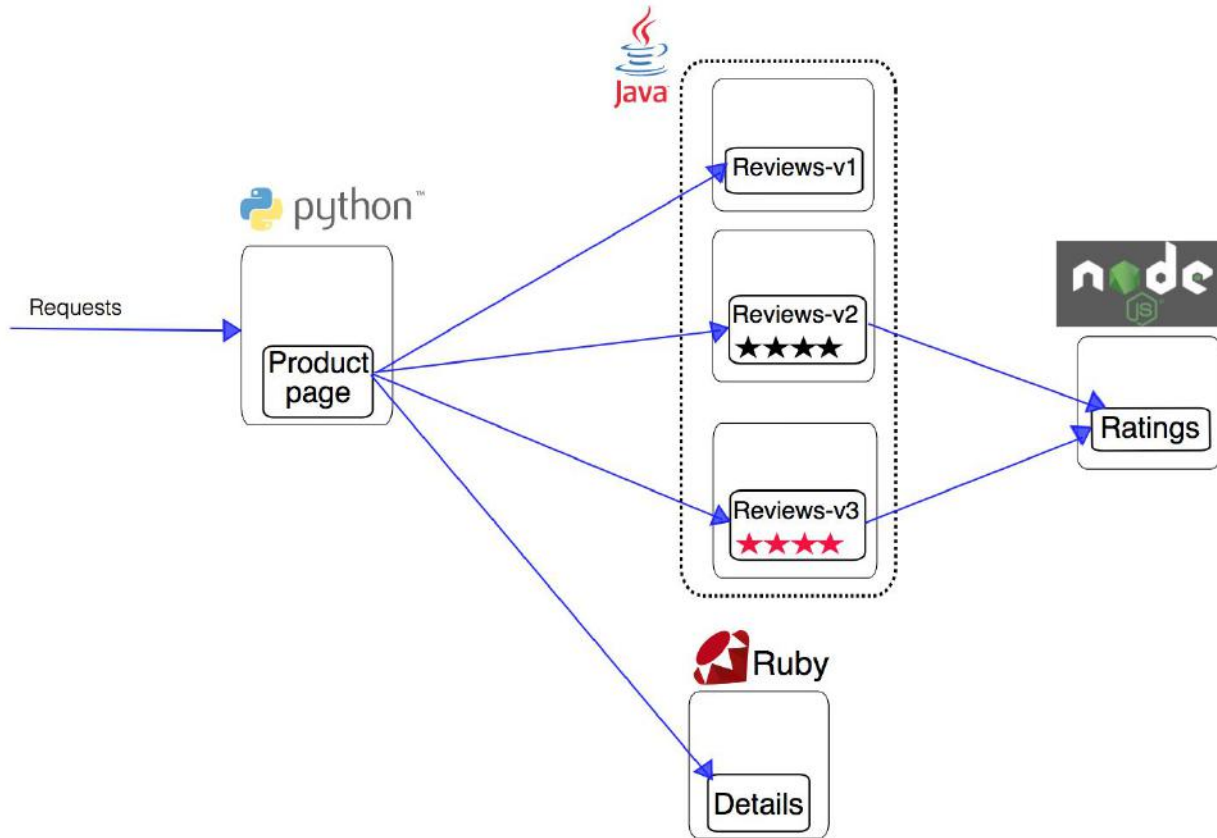

How to use it

Routes and commands injected via CLI or API:

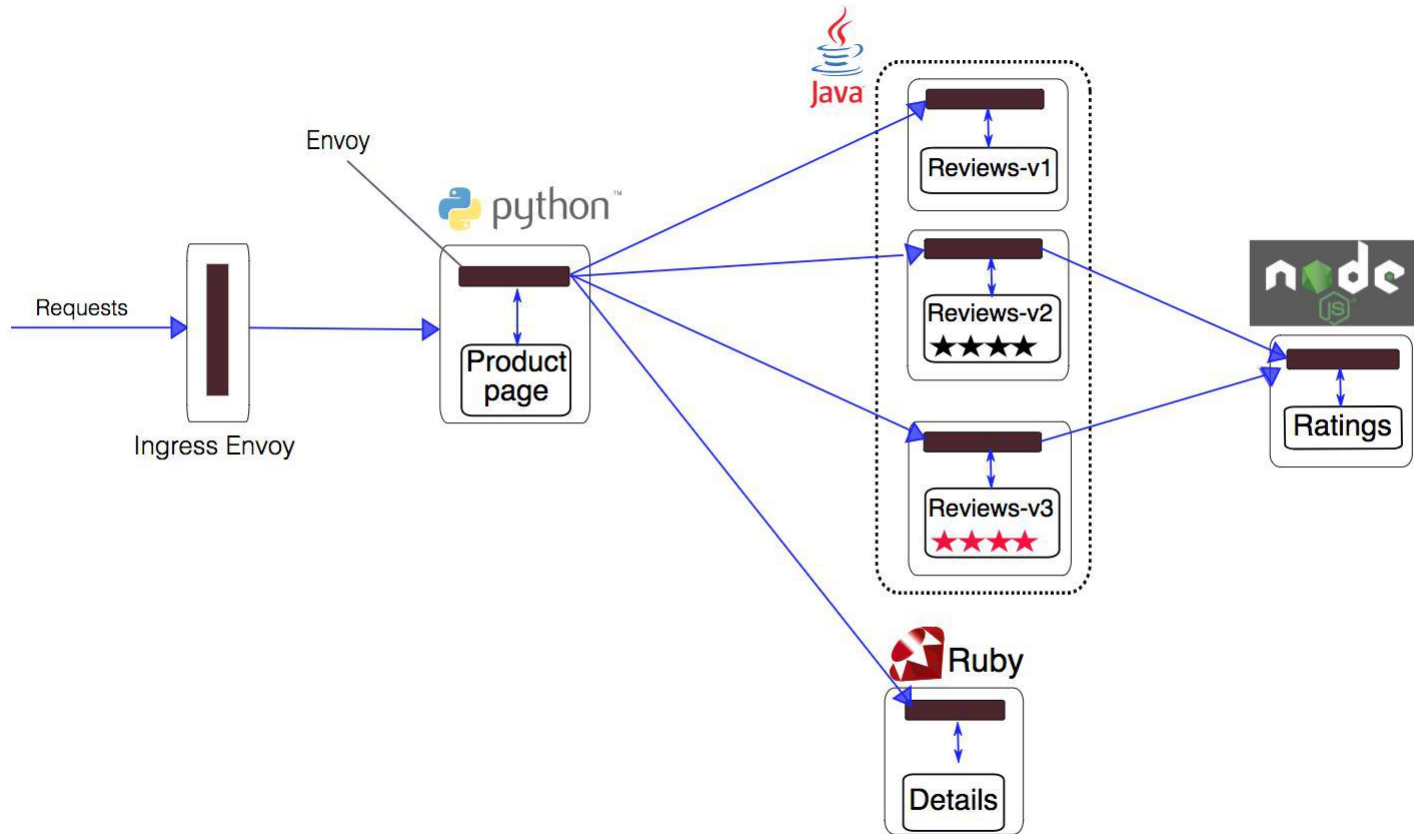
```
apiVersion: config.istio.io/v1alpha2
kind: RouteRule
metadata:
  name: reviews-test-v2
spec:
  destination:
    name: reviews
  precedence: 2
  match:
    request:
      headers:
        cookie:
          regex: "^(*?;)?(user=jason)(;.*)?$"
  route:
  - labels:
    version: v2
```

DEMO





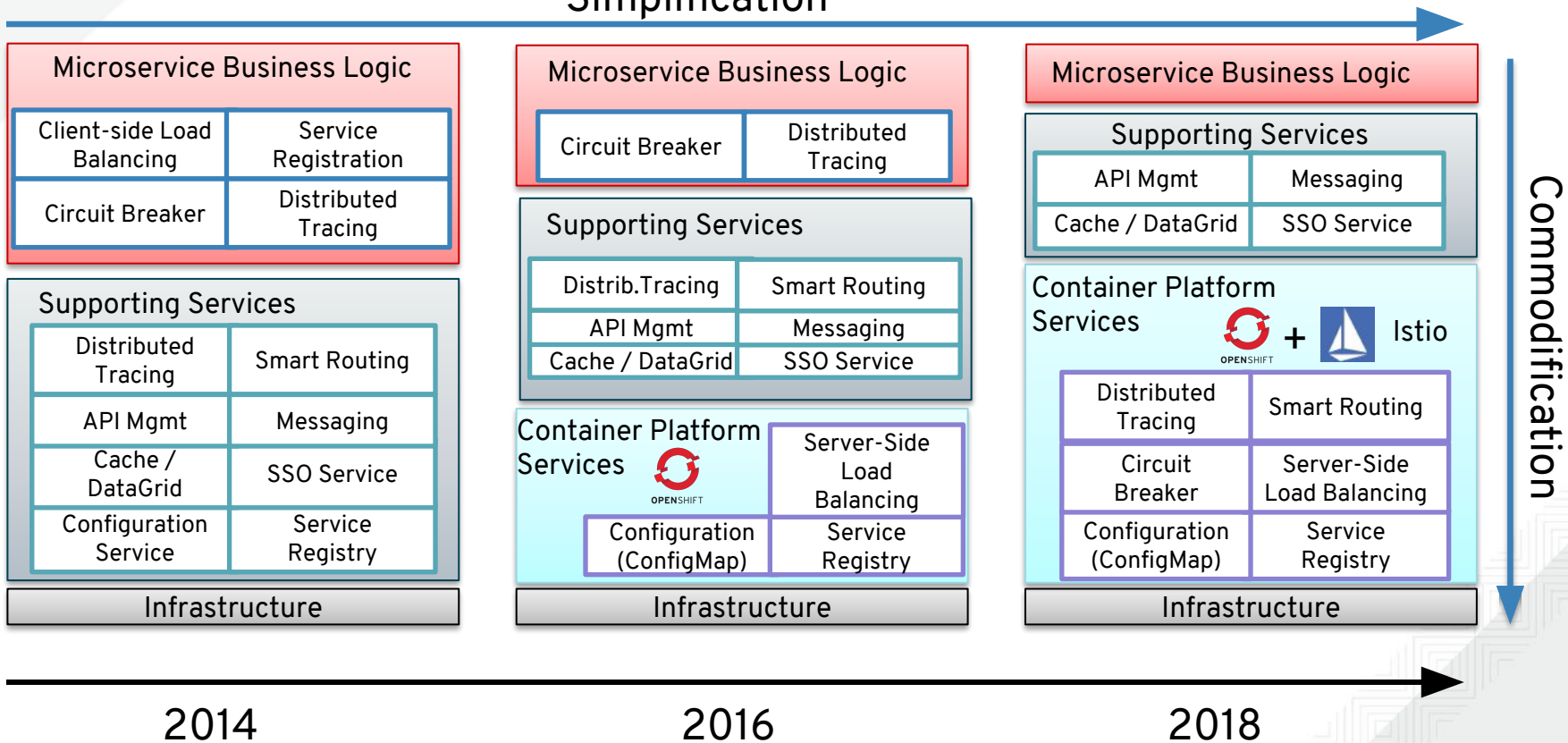
BookInfo Application without Istio



BookInfo Application

Evolution of Microservices

Simplification



2014

2016

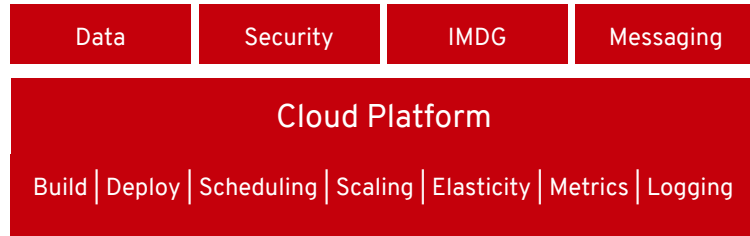
2018

Ok, but that's all about MSA infrastructure...

OpenShift is the best Container Platform, it will solve for you many of the problems at an infrastructure level without cluttering your code but...

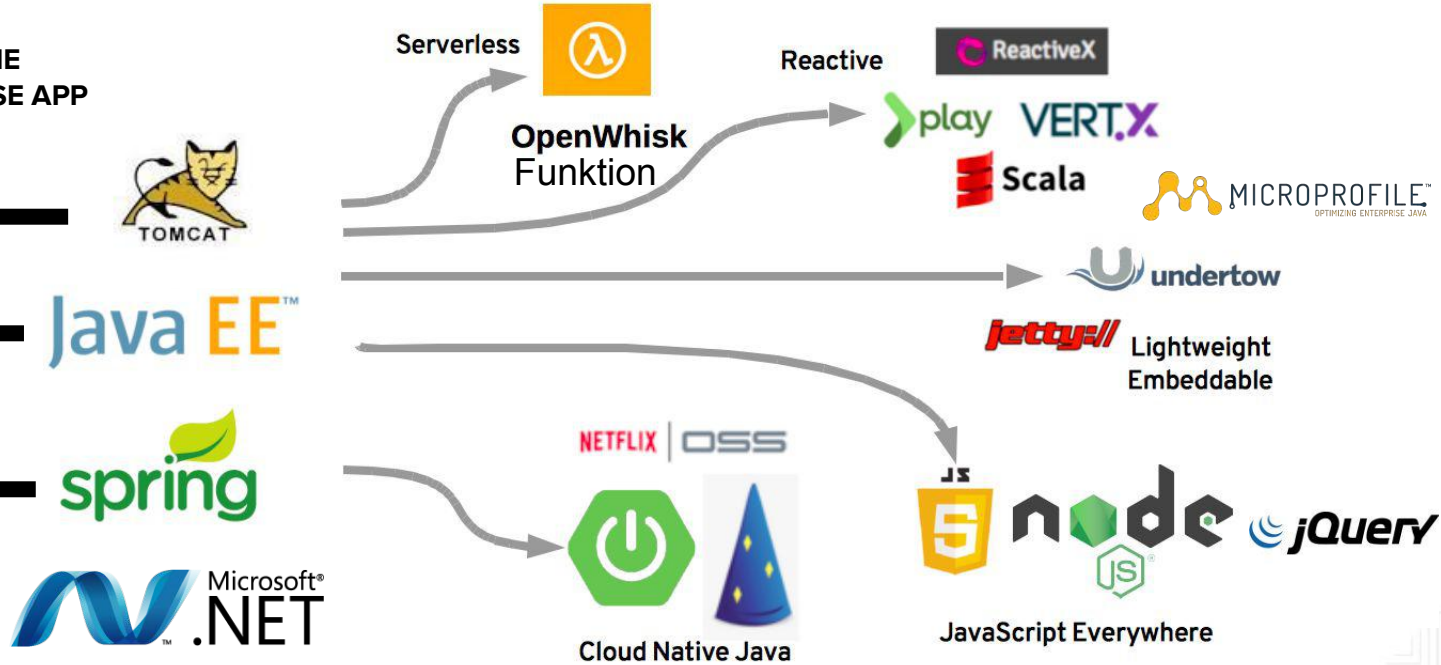
... we still need to *code* our microservices!

The App Server 2014/...



Where developers are going

50% OF THE
ENTERPRISE APP
MARKET

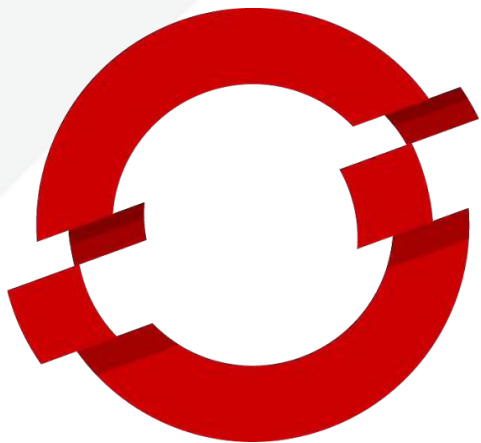


Martin Fowler: monolith first!

*“Almost all the successful **microservice** stories have **started with a monolith that got too big** and was broken up”*

*“Almost all the cases where I've heard of a **system** that was **built as a microservice** system from scratch, it has ended up in **serious trouble.**”*



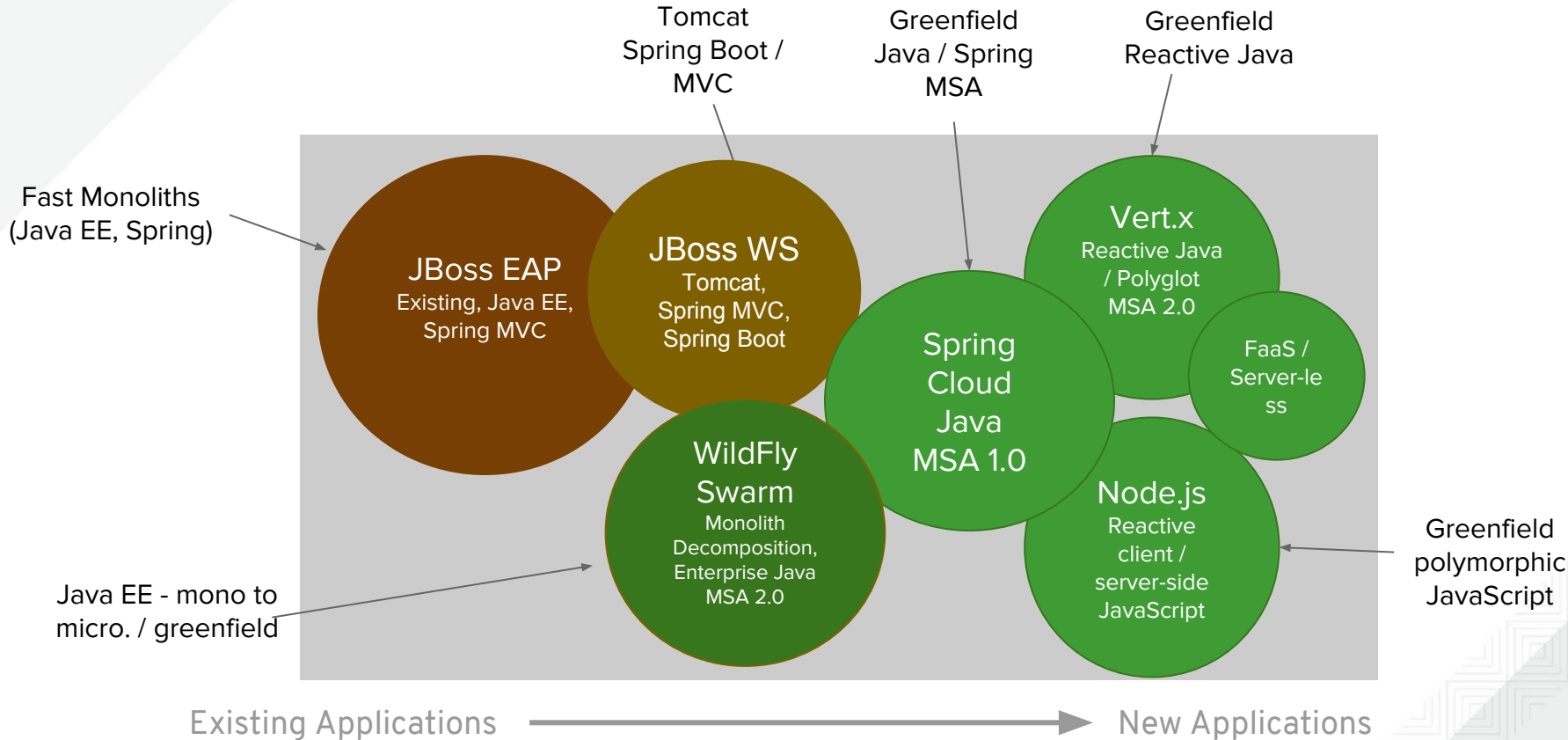


RED HAT® OPENSSHIFT Application Runtimes

Modern, cloud-native application runtimes and an opinionated developer experience for organizations that are moving beyond 3-tier architectures and embracing cloud-native application development.

RHOAR: OpenShift Application Runtimes

- **Multiple runtime options**
 - JBoss EAP - existing Java EE / Spring apps.
 - WildFly Swarm / MicroProfile - Java EE centric MSA
 - Spring Boot / Cloud - Spring centric MSA
 - Vert.x - greenfield reactive Java
 - Node.js - greenfield reactive JavaScript
- **OpenShift - Public, Dedicated Public & Enterprise**
- **Tightly integrated with OpenShift & Kubernetes**
- **Tightly Integrated with Red Hat Developer SaaS**
- **3rd-party Integrations - eg. Netflix Ribbon, Hystrix, etc.**
- **Opinionated DevX starting with launch.openshift.io**



Ok, so it's (also) about being lighter?

Theoretically, yes. But, beware:

- A simple ReST service deployed in EAP used $\frac{1}{5}$ of the memory used by Spring Boot under load and was **2x** faster!

Runtime (framework)	Boot time server only	Boot time including app deployment	Memory usage without load	Memory usage under load	Measured throughput
JBoss EAP (Java EE)	2 - 3 sec	3 sec	40 MB	200 - 400 MB	23K req/sec
JBoss EAP (Spring)	2 - 3 sec	7 sec	40 MB	500 - 700 MB	9K req/sec
JBoss WS/Tomcat (Spring)	0 - 1 sec	8 sec	40 MB	0.5 - 1.5 GB	8K req/sec
Fat JAR (Spring Boot)	N/A	3 sec	30 MB	0.5 - 2.0 GB	11K req/sec
Fat JAR (WF Swarm)	1-2 sec	5 sec	30 MB	250 - 350 MB	27K req/sec

Don't believe it? Try it out yourself <http://bit.ly/modern-java-runtimes>

Ok, so it's (also) about being lighter?

Theoretically, yes. But, beware:

- A simple ReST service deployed in EAP used $\frac{1}{5}$ of the memory used by Spring Boot under load and was **2x** faster!

Runtime (framework)	Boot time server only	Boot time including app deployment	Memory usage without load	Memory usage under load	Measured throughput
JBoss EAP (Java EE)	2 - 3 sec	3 sec	40 MB	200 - 400 MB	23K req/sec
JBoss EAP (Spring)	2 - 3 sec	7 sec	40 MB	500 - 700 MB	9K req/sec
JBoss WS/Tomcat (Spring)	0 - 1 sec	8 sec	40 MB	0.5 - 1.5 GB	8K req/sec
Fat JAR (Spring Boot)	N/A	3 sec	30 MB	0.5 - 2.0 GB	11K req/sec
Fat JAR (WF Swarm)	1-2 sec	5 sec	30 MB	250 - 350 MB	27K req/sec

Don't believe it? Try it out yourself <http://bit.ly/modern-java-runtimes>

Key Differentiators

Polyglot

- Language agnostic platform
- Initial focus on Java & JavaScript

Best in class OSS

- Container, Kubernetes, Java, JavaScript, Spring

Poly-architecture

- Fast monoliths (existing Java EE, Spring MVC)
- Mini and microservices
- Serverless (in the future)

Key Differentiators

Deployment & Packaging

Physical Servers



Virtual Servers



Containers



- Resource efficiency
- Automation for microservices, but also support traditional applications
- Enable faster and more consistent deployments from Development to Production
- Enable application portability across 4 infrastructure footprints: Physical, Virtual, Private & Public Cloud

Key Differentiators



Multiple Runtimes supported in single SKU

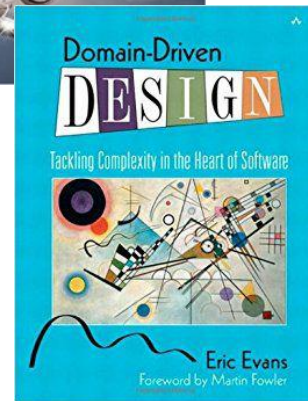
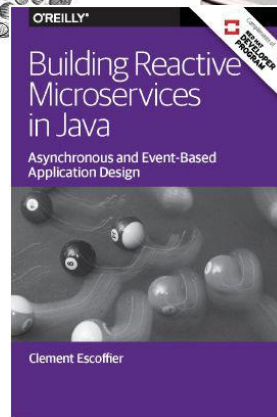
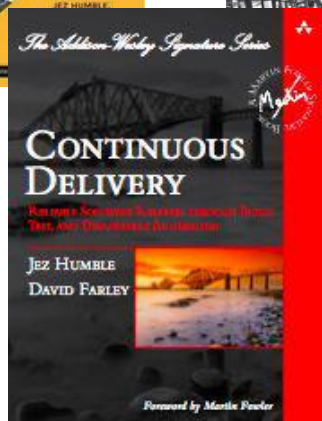
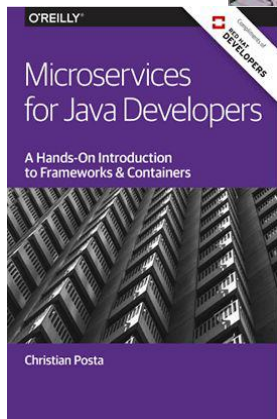
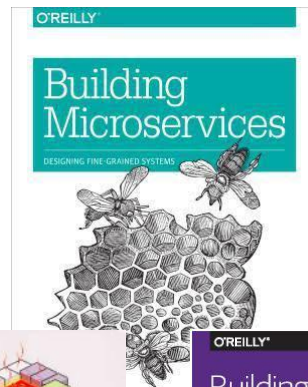
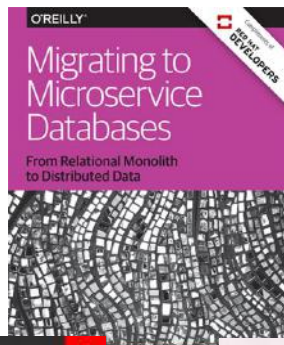
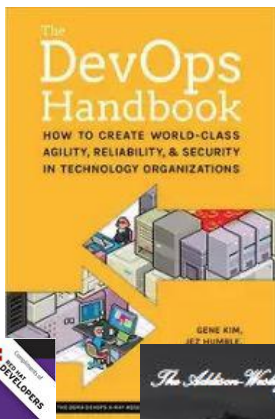
Support 12-factor / cloud-native design-patterns :

- Healthcheck / load-balancing / proxying
- Registry / config.
- Rolling upgrades / retries / failover
- Separation of concerns

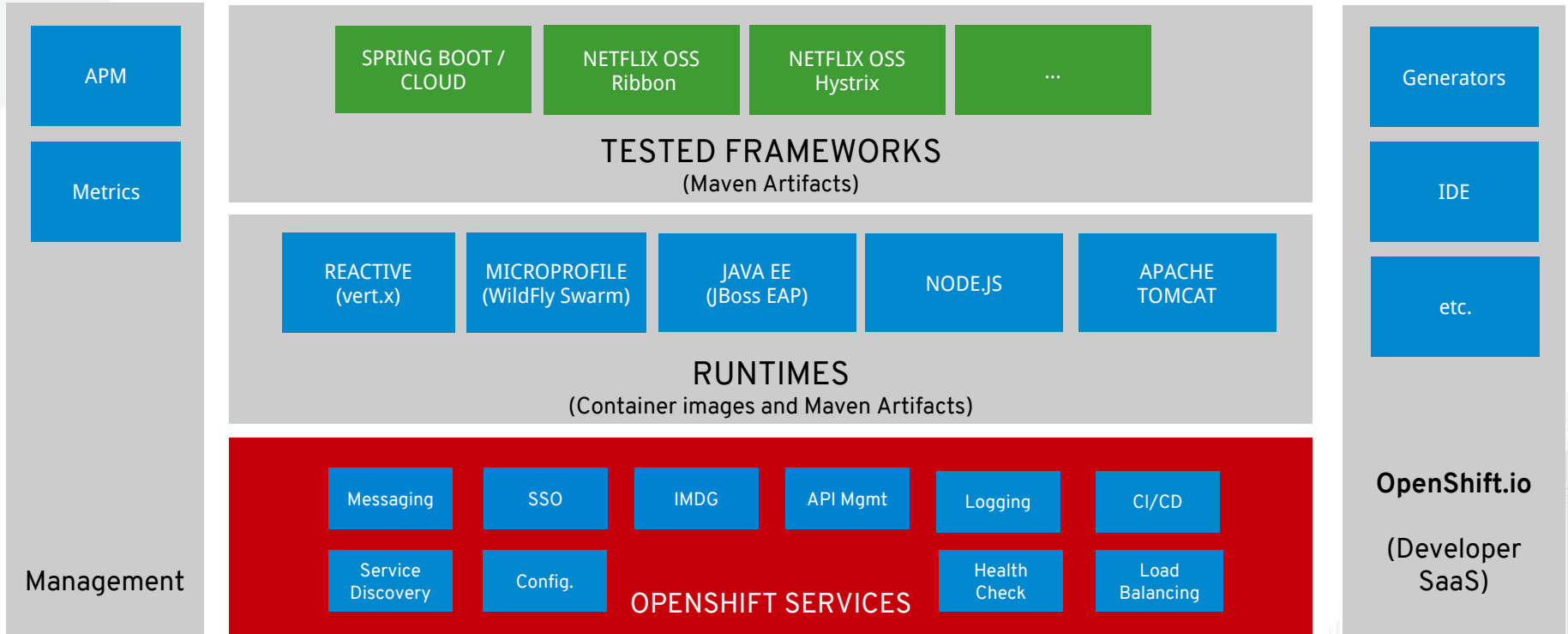
Cloud-scale design

- Networking, storage, auto-scaling, logs, alerting

The books you'll need to read



RHOAR: OpenShift Application Runtimes



QUESTIONS?





RED HAT

OPEN SOURCE DAY

Europe, Middle East & Africa