

OpenShift Developer

Quarkus, Service Mesh, Serverless

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat

Agenda / etc.

Agenda

- Quarkus
- Service Mesh (istio.io)
- Serverless (knative)

Quarkus Business Value



QUARKUS

Supersonic. Subatomic. Java.

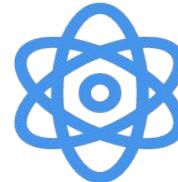
Quarkus - Kubernetes Native Java



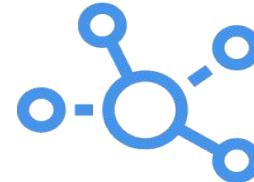
Monolith



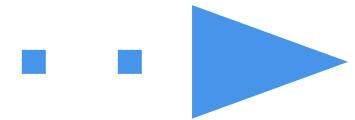
Cloud Native



Microservices



Serverless



Event-Driven
Architecture



kubernetes



Istio



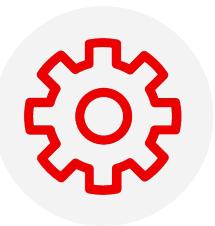
Knative

WHAT IS QUARKUS?

QUARK: elementary particle / **US**: hardest thing in computer science

Quarkus Business Value

"Supersonic, Subatomic Java"



Cost Savings

Cloud efficiency (low memory, fast startup, high density), serverless deployments



Faster Time to Market

Developer productivity, extensions ecosystem, low learning curve, keep competitive edge



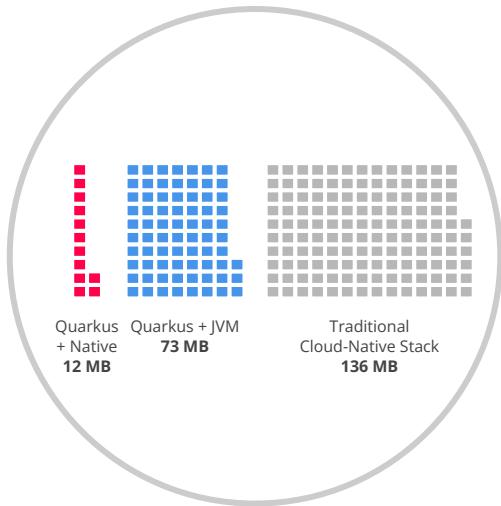
Reliability

Trusted technology, active community, trusted sponsor, proven Java libraries and standards

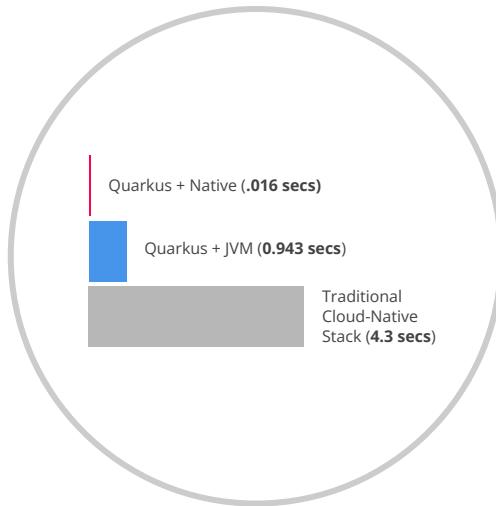
Cost Savings

"both jvm & native compilation modes"

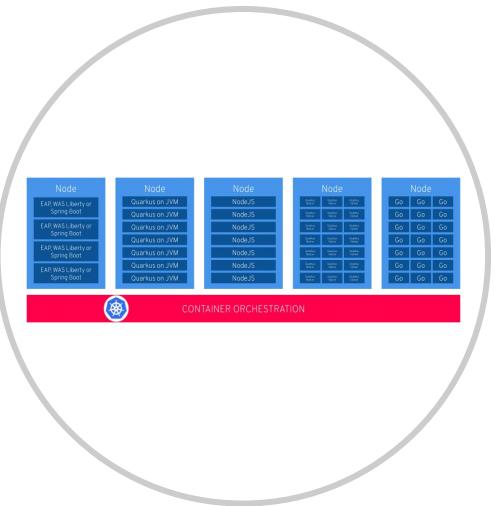
Reduced Memory Footprint



Fast Startup Time



Smaller Disk Footprint



Cost Savings

"from the mouths of production users"

We could run **3x** denser deployments without sacrificing availability and response times of service.

-Lufthansa Technik

We went from **1 min** startup times to **400 ms**.

-Suomen Asiakastieto Oy

We went from **6-7 sec** startup time with plain Java on AWS Lambda to **10 ms** with Quarkus.

-Ennovative Solutions

Faster Time to Market

"Platform that feels familiar but new"

Quarkus allows developers to leverage their existing Java expertise to reduce their time to productivity. It feels familiar and innovative as it unlocks the ability to utilize their Java skills to build modern, cloud-native microservices and serverless applications.



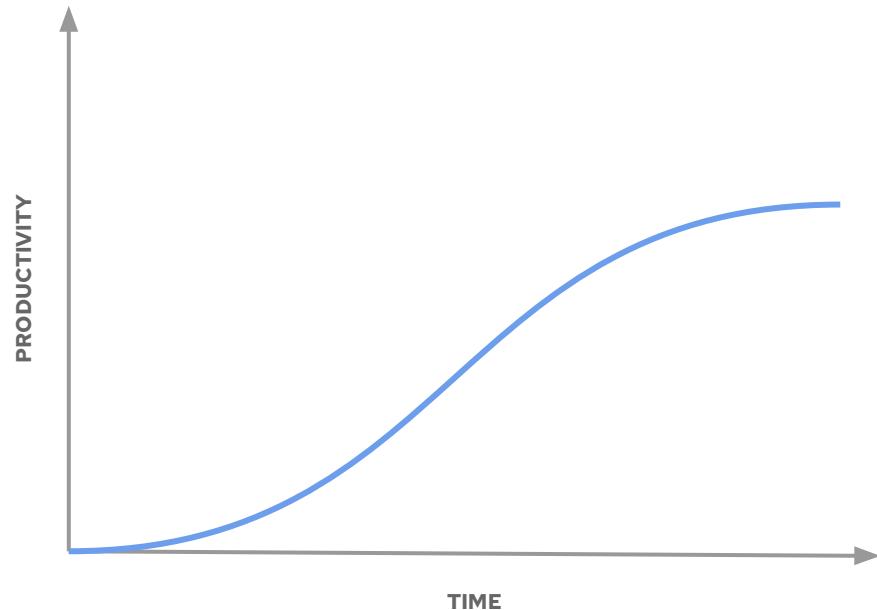
Derived from Java standards like Jakarta EE and Eclipse MicroProfile



Crafted from well known Java libraries such as Hibernate and Spring



Developer tooling and extension library to build applications quicker



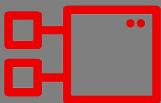
Why Quarkus?

"Quarkus is a great fit if you are looking for..."



IMPROVED APP PERFORMANCE & COST

- Reduce memory consumption
- Increase startup speeds
- Reduce application size
- Looking for Spring or Java alternatives



DIGITAL TRANSFORMATION

- Modernize or optimize existing applications
- Move to a cloud-native or microservices architecture
- Need lightweight connectivity between services
- Next generation business automation



SERVERLESS

- Have seldom used but critical services
- Interested in serverless or FaaS (function as a service)

Use Cases

"Quarkus is an ideal runtime for"



NET NEW

Low memory footprint + lightning fast startup time + small disk footprint = an ideal runtime for Kubernetes-native microservices



MONO 2 MICRO

Quarkus is a great choice to modernize existing monolithic applications by breaking it into smaller, loosely coupled microservices.



SERVERLESS

Scaling up or down (0) is extremely fast with Quarkus making it an ideal runtime for creating serverless applications.



EVENT-DRIVEN/REACTIVE

Quarkus utilizes an asynchronous, reactive event loop that makes it easy to create reactive applications.

Quarkus

Technical Value

“Historical” Enterprise Java Stack

Architecture: **Monoliths**



Deployment: **multi-app, appserver**

Dynamic Application Frameworks

App Lifecycle: **Months**

Application Server

Memory: **1GB+ RAM**

Java Virtual Machine (Hotspot)

Startup Time: **10s of sec**

Operating System + Hardware/VM

“Modern” Enterprise Java Stack

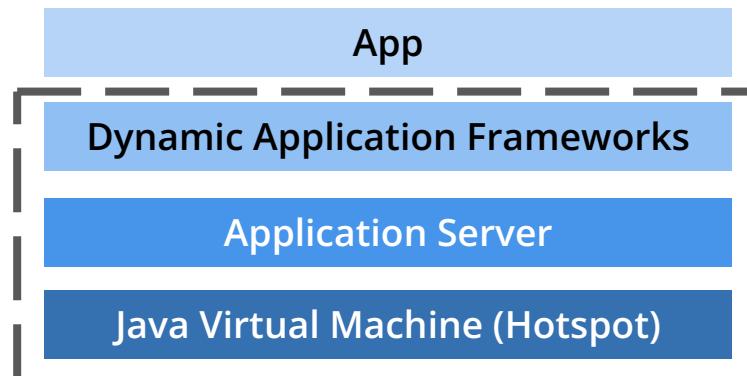
Architecture: **Microservices**

Deployment: **Single App**

App Lifecycle: **Days**

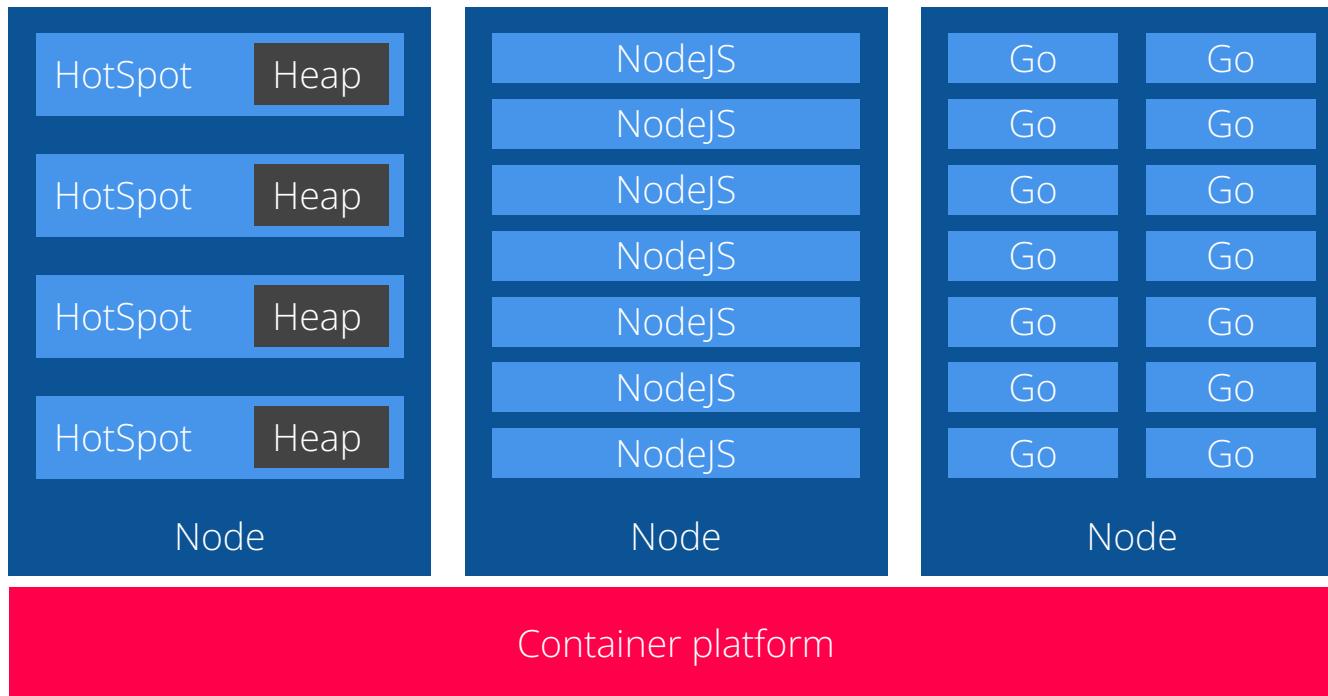
Memory: **100MBs+ RAM**

Startup Time: **Seconds**



**No
Change**

Hidden Truth About Java + Containers



**THERE IS A NEED FOR A
NEW JAVA STACK FOR
CLOUD-NATIVE AND
SERVERLESS**

Experts from cloud-native Java OS projects

VERT.X



Eclipse Vert.x



Hibernate



RESTEasy



Eclipse MicroProfile



WildFly



OpenJDK



QUARKUS

Differentiators



Container First

- Tailors your app for HotSpot & GraalVM
- Fast boot time and low RSS memory
- Serverless fit



Developer Joy

- Live coding
- Unified configuration



Unifies Imperative & Reactive

- Combines blocking and non-blocking
- Built-in event bus



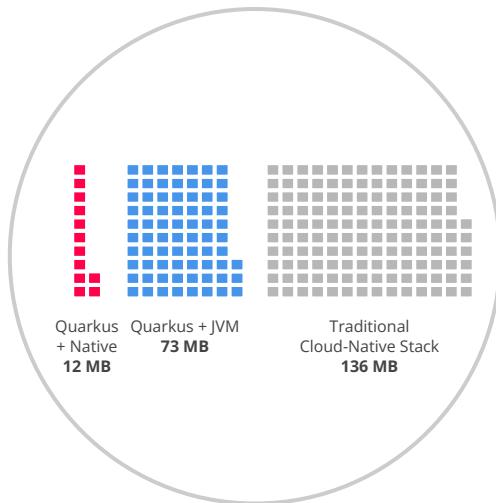
Best of Breed Libraries & Standards

- 90+ extensions
- "Powered by Quarkus" applications

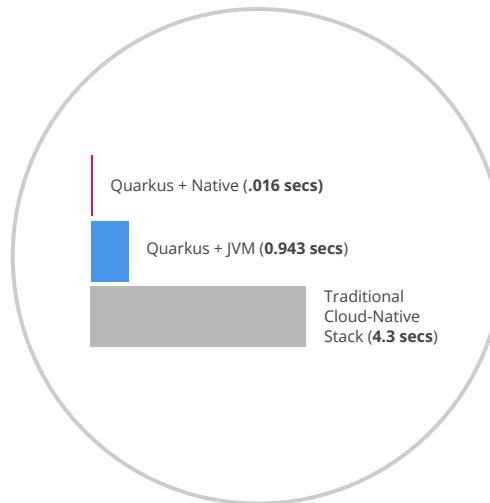
Benefit No. 1: Container First

“We went from 1-min startup times to 400 milliseconds”

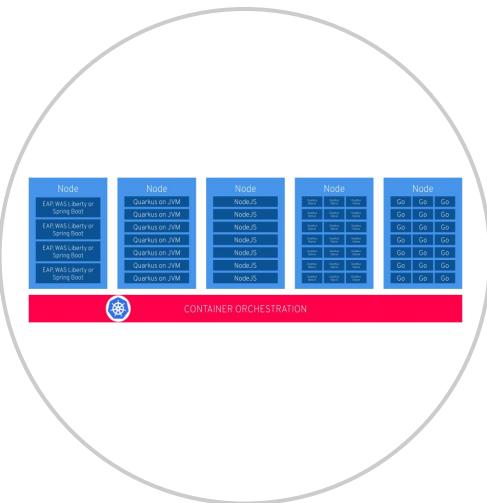
Reduced Memory Footprint



Fast Startup Time



Smaller Disk Footprint

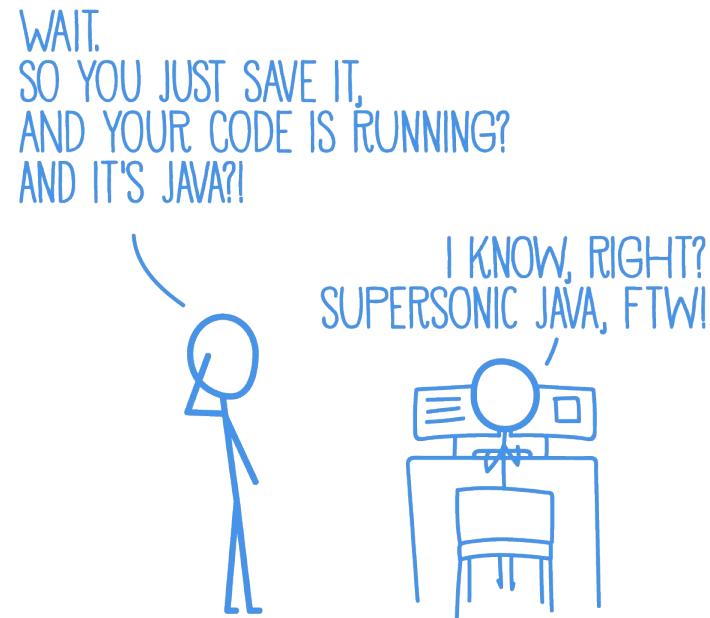


Benefit No. 2: Developer Joy

*“Our developers used to wait **2 to 3 mins** to see their changes. **Live coding** does away with this.”*

A cohesive platform for optimized developer joy:

- Based on standards and more
- Unified configuration
- Live coding
- Streamlined code for the 80% common usages, flexible for the 20%
- No hassle native executable generation



Benefit No. 3: Unifies Imperative and Reactive

```
@Inject
SayService say;

@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return say.hello();
}
```

```
@Inject @Stream("kafka")
Publisher<String> reactiveSay;

@GET
@Produces(MediaType.SERVER_SENT_EVENTS)
public Publisher<String> stream() {
    return reactiveSay;
}
```

- Combine both Reactive and imperative development in the same application
- Inject the EventBus or the Vertx context
- Use the technology that fits your use-case
- Key for reactive systems based on event driven apps

Benefit No. 4: Best of Breed Frameworks & Standards

"When you adopt Quarkus, you will be productive from day one since you don't need to learn new technologies."



Eclipse Vert.x



Hibernate



RESTEasy



Apache Camel



Eclipse MicroProfile



Netty



Kubernetes



OpenShift



Jaeger



Prometheus



Apache Kafka



Infinispan



Flyway



Neo4j



MongoDB



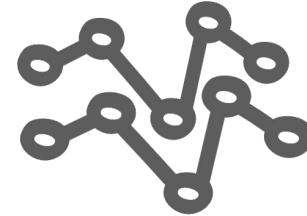
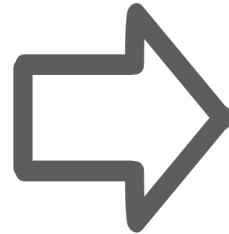
MQTT



Keycloak



Apache Tika



Supersonic, Subatomic

Fast.

Blazing fast to start.

Millisecond fast!

Supersonic, Subatomic Java

REST

Quarkus + Native (via GraalVM) **0.016 Seconds**

Quarkus + JVM (via OpenJDK) **0.943 Seconds**

Traditional Cloud-Native Stack **4.3 Seconds**

REST + CRUD

Quarkus + Native (via GraalVM) **0.042 Seconds**

Quarkus + JVM (via OpenJDK) **2.033 Seconds**

Traditional Cloud-Native Stack **9.5 Seconds**

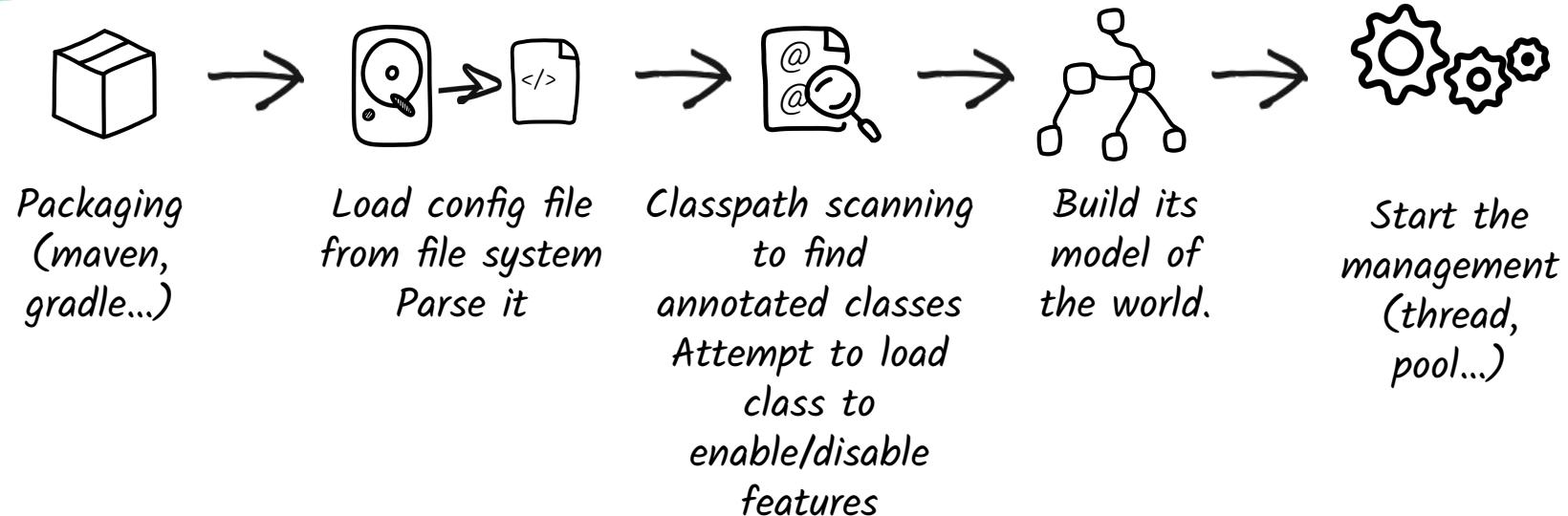
Time to first response

HOW DOES QUARKUS WORK?

How Does a Framework Start?

Build Time

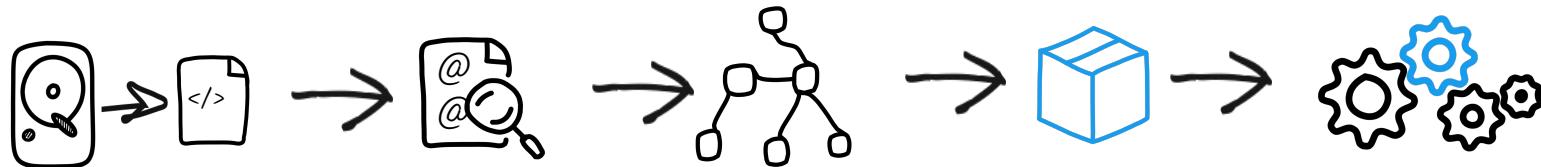
Runtime



The Quarkus Way

Build Time

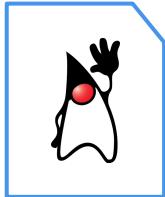
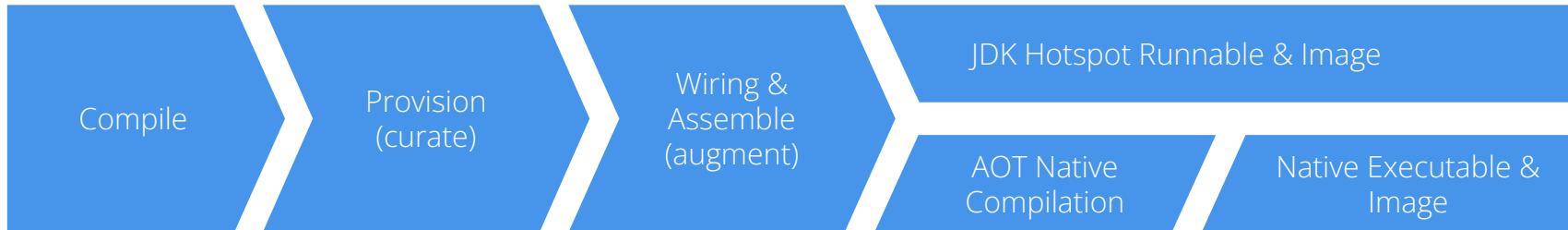
Runtime



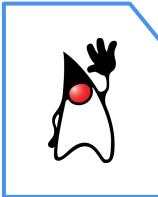
Build Time

Runtime

An ahead-of-time, build-time, runtime



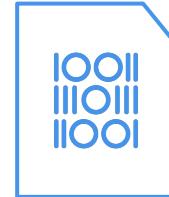
app.jar



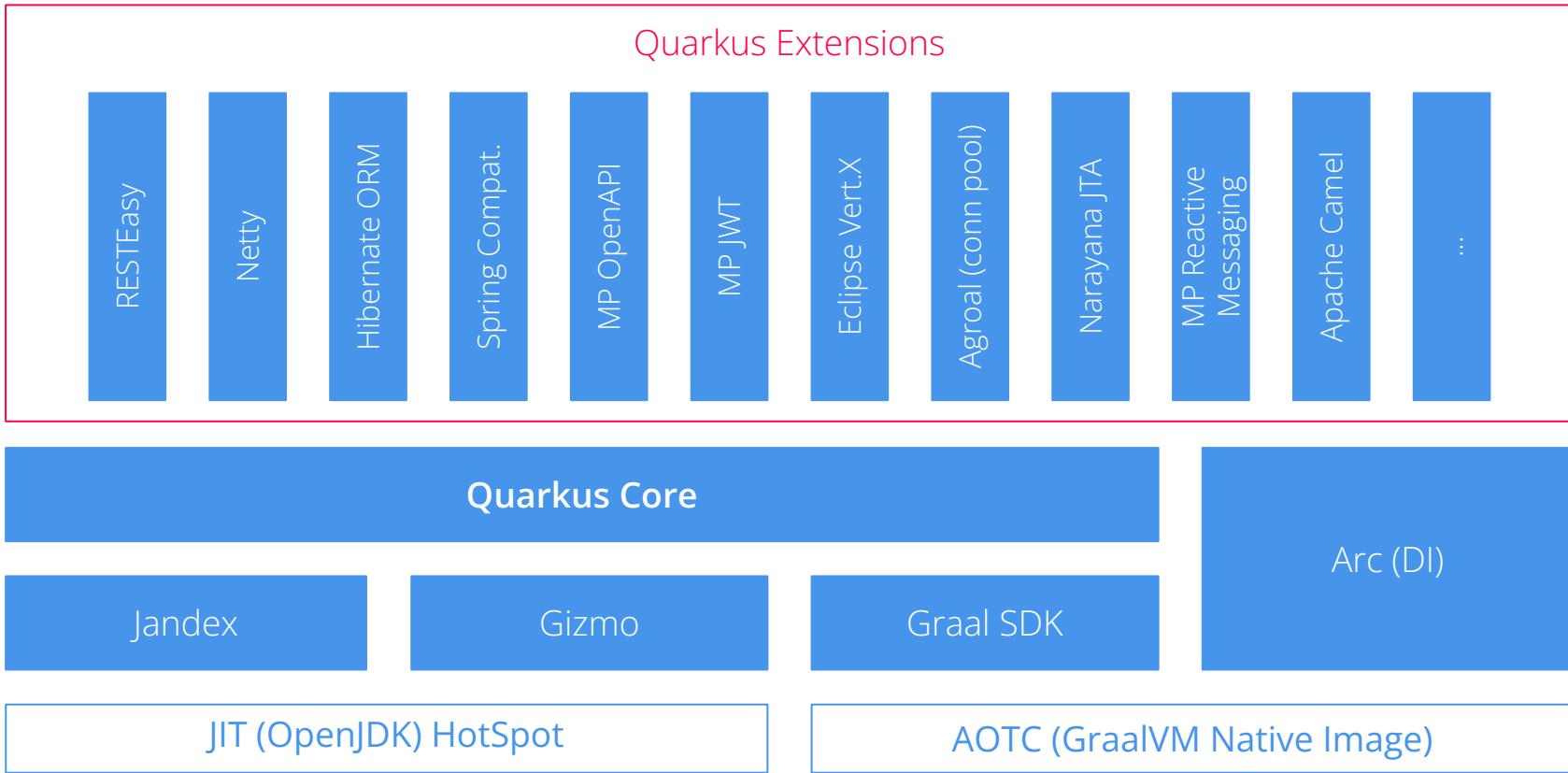
frameworks



Runnable java app



native-app



The Right VM For the Right Deployment

JIT (OpenJDK HotSpot)

- High memory density requirements
- High request/s/MB
- Fast startup time
- Best raw performance (CPU)
- Best garbage collectors
- Higher heap size usage
- Known monitoring tools
- Compile Once, Run anywhere
- Libraries that only work in standard JDK

AOT (GraalVM native image)*

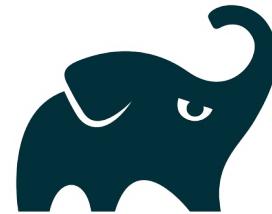
- Highest memory density requirements
- Highest request/s/MB for low heap size usages
- Faster startup time
10s of ms for Serverless

*Currently in Tech Preview



Quarkus Tools - Build

maven



Gradle*

```
mvn io.quarkus:quarkus-maven-plugin:1.3.2.Final-redhat-00001:create \
-DprojectGroupId=org.acme \
-DprojectArtifactId=getting-started \
-DplatformGroupId=com.redhat.quarkus \
-DplatformVersion=1.3.2.Final-redhat-00001 \
-DclassName="org.acme.quickstart.GreetingResource" \
-Dpath="/hello"
cd getting-started
```

*community supported



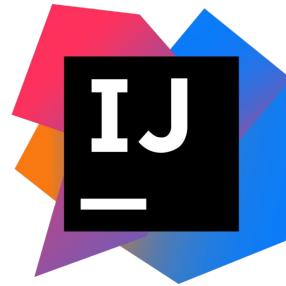
Quarkus Tools - IDE



[VSCode](#)



[Eclipse](#)



[IntelliJ](#)



[che.openshift.io](#)

DEMO

<https://github.com/wpernath/quarkus-worldtour>

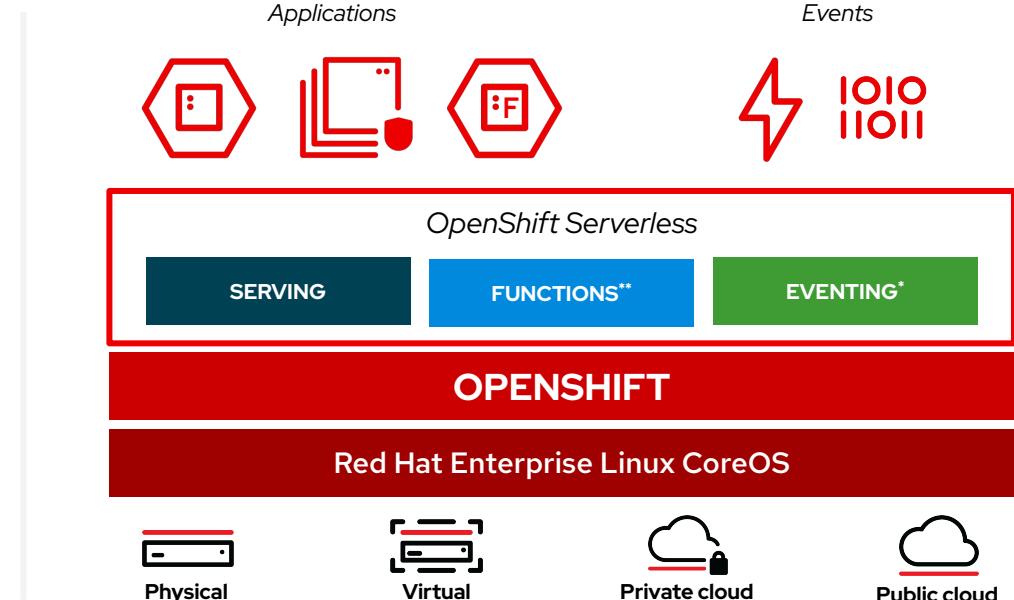
Optional - Serverless / knative



OpenShift Serverless

Event-driven serverless containers and functions

- Deploy and run **serverless containers**
- Use any programming language or runtime
- Modernize existing applications to run serverless
- Powered by a rich ecosystem of event sources
- Manage serverless apps natively in Kubernetes
- Based on open source project **Knative**
- Run anywhere OpenShift runs



* Eventing is currently in Technology Preview

** Functions are currently a work in progress initiative

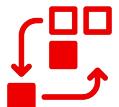
OpenShift Serverless

Key Features



Containers made easy

Simplified developer experience to deploy applications/code on serverless containers abstracting infrastructure & focusing on what matters.



Immutable revisions

Deploy new features: performing canary, A/B or blue-green testing with gradual traffic rollout with no sweat and following best practices.



Automatic scaling

No need to configure number of replicas, or idling. Scale to zero when not in use, auto scale to thousands during peak, with built-in reliability and fault-tolerance.



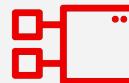
Ready for the Hybrid

Portable serverless running anywhere OpenShift runs, that is on-premises or on any public cloud. Leverage data locality and SaaS when needed.



Any programming language

Use any programming language or runtime of choice. From Java, Python, Go and JavaScript to Quarkus, SpringBoot or Node.js.



Event Driven

Architectures coupled & distributed apps connecting with a variety of built-in or third-party event sources or connectors powered by Operators.

Installation experience

"Easy day 1 and even better for day 2"

- Click Install experience
- Developer & admin experience in Console
- Built-in event sources
- No external dependencies.

 OpenShift Serverless Operator

1.7.0 provided by Red Hat, Inc.

[Install](#)

OPERATOR VERSION
1.7.0

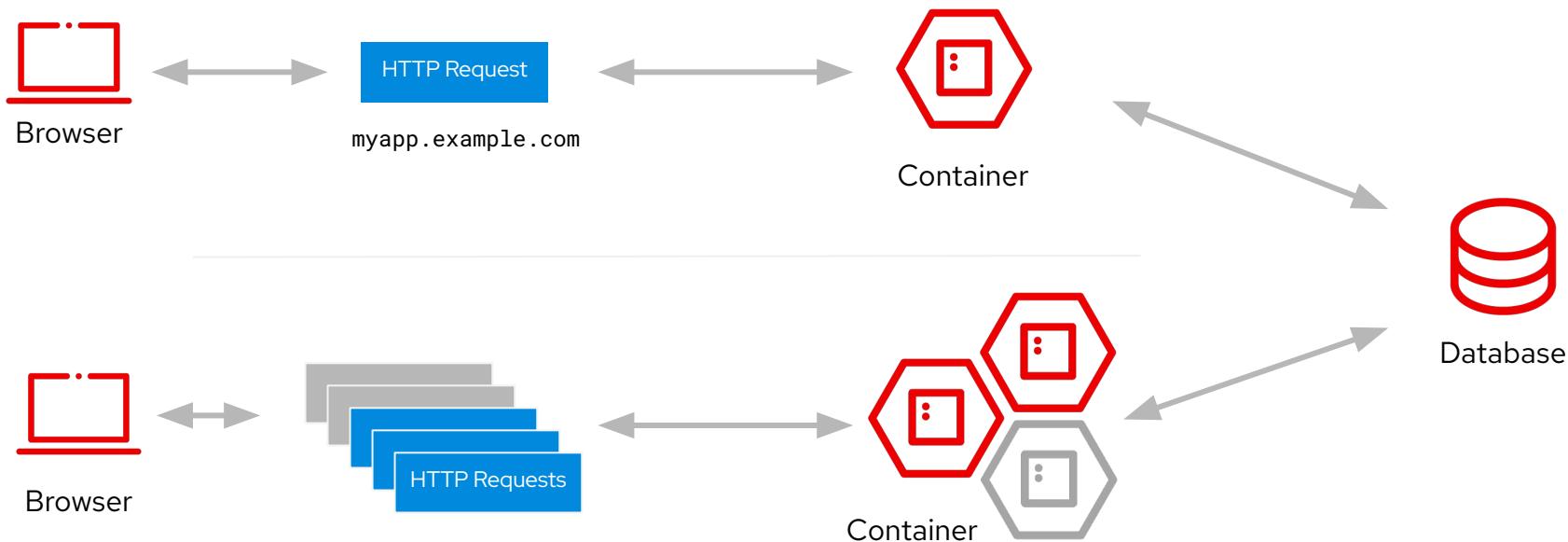
PROVIDER TYPE
Red Hat

PROVIDER
Red Hat, Inc.

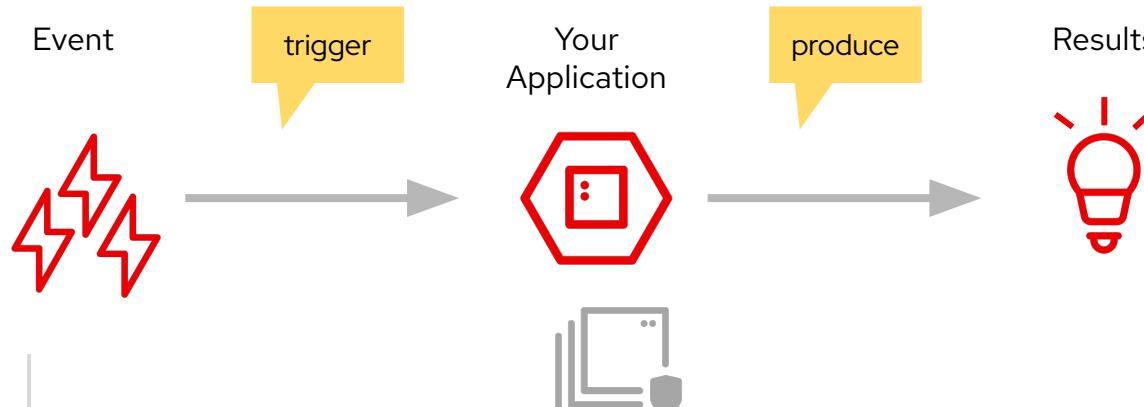
The Red Hat OpenShift Serverless operator provides a collection of APIs that enables containers, microservices and functions to run "serverless". Serverless applications can scale up and down (to zero) on demand and be triggered by a number of event sources. OpenShift Serverless integrates with a number of platform services, such as Metering and Monitoring and it is based on the open source project Knative.

The "Serverless Pattern"

A serverless web application



The "Serverless Pattern"



HTTP Requests

Kafka Messages

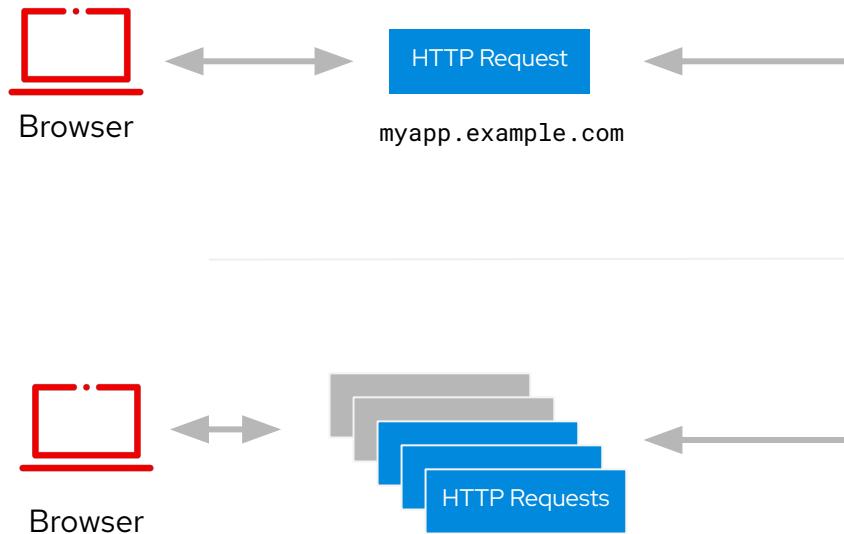
Image Uploaded

New Order

Login from user

The "Serverless Pattern"

A serverless web application

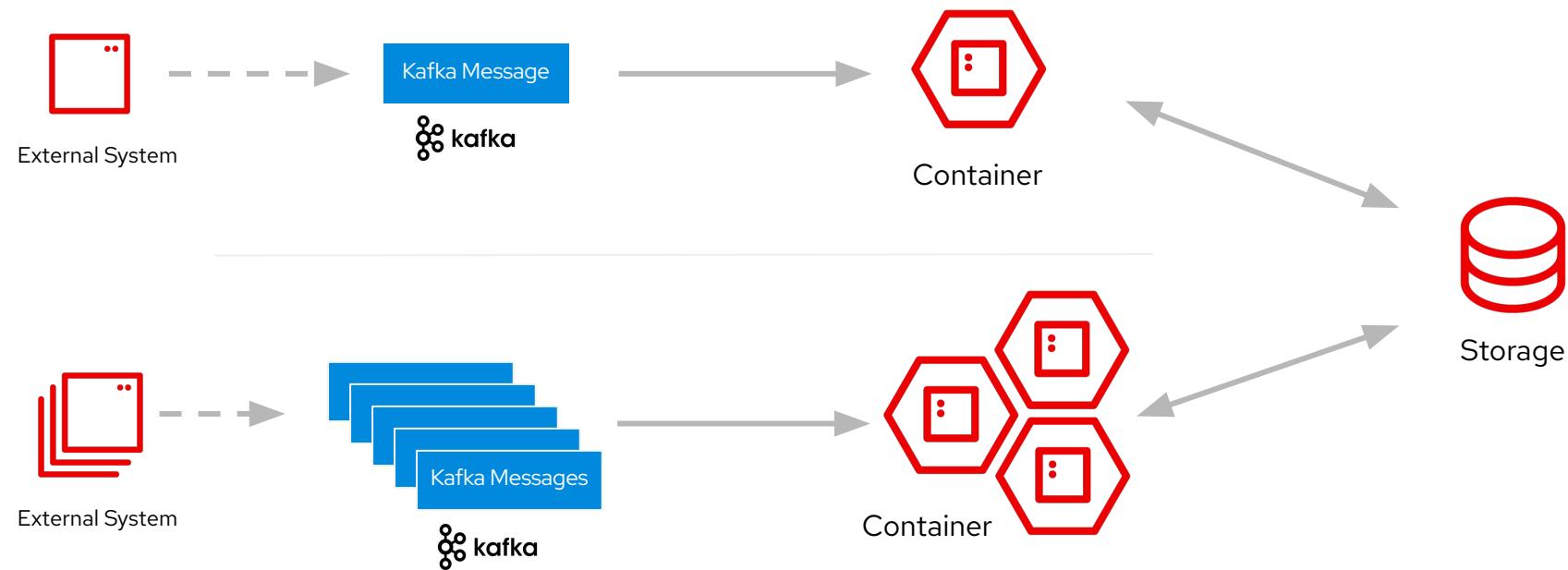


Benefits of this model:

- No need to setup auto-scaling and load balancers
 - Scale down and save resources when needed.
 - Scale up to meet the demand.
- No tickets to configure SSL for applications
- Enable Event Driven Architectures (EDA) patterns
- Enable teams to associate cost with IT
- Modernize existing applications to run as serverless containers

The "Serverless Pattern"

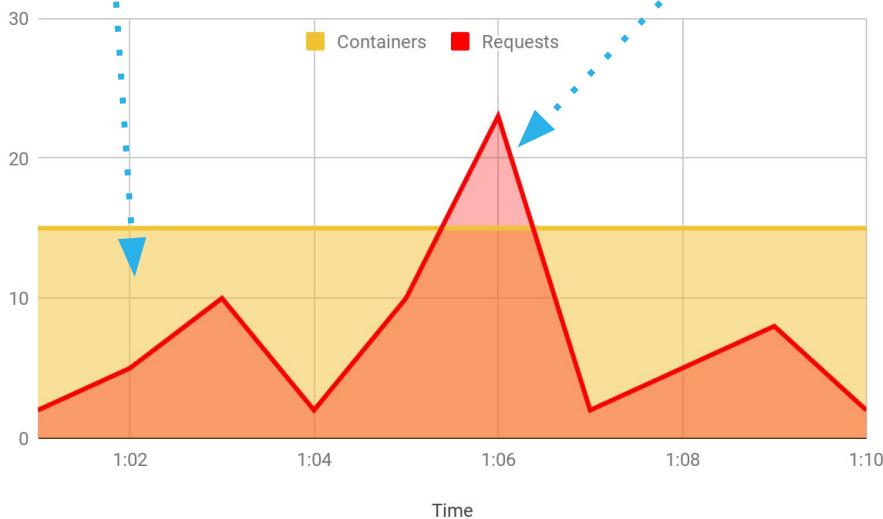
Processing a Kafka message



Serverless Operational Benefits

Over provisioning

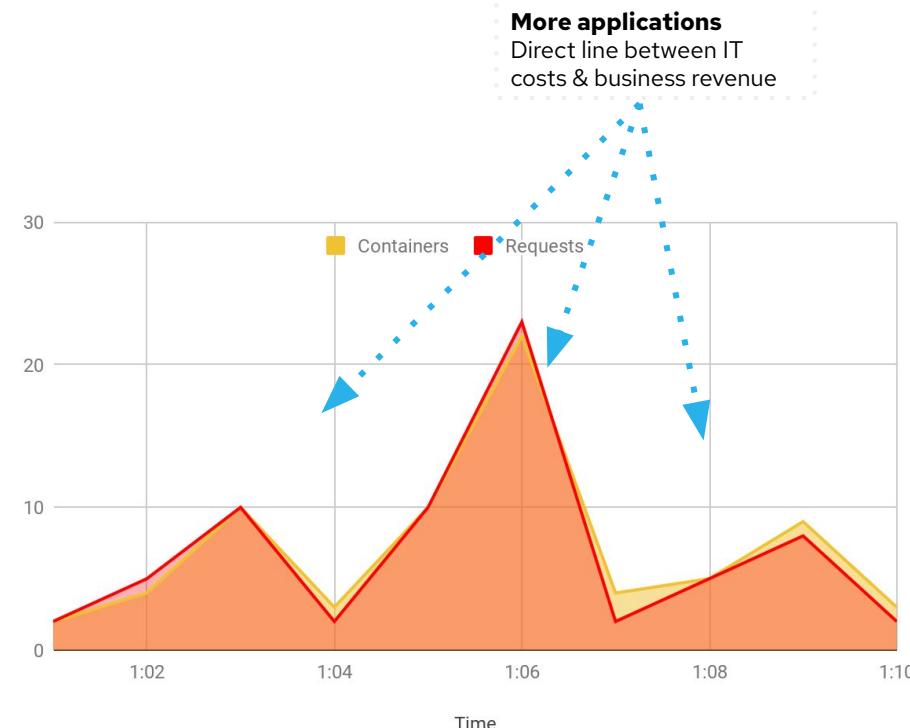
Time in capacity planning
IT cost of idle resources



NOT Serverless

Under provisioning

Lost business revenue
Poor quality of service



with Serverless

Choosing the Right Tool

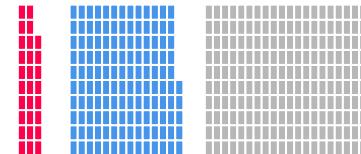
Your Team



The Ecosystem



Performance





Developer experience

</> Java
</> Node
</> Python
</> Go
</> Ruby
</> Ruby on Rails
</> PHP
</> Perl
</> .NET Core

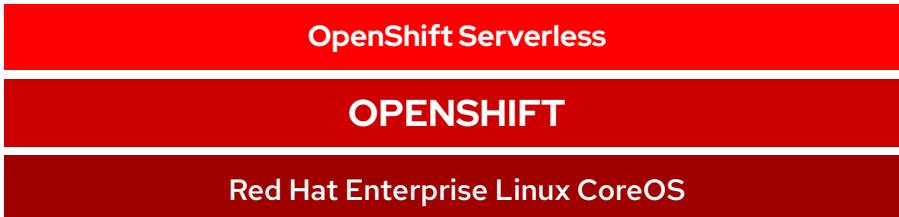
```
$ kn service create --image=
```



CLI



UI



Physical



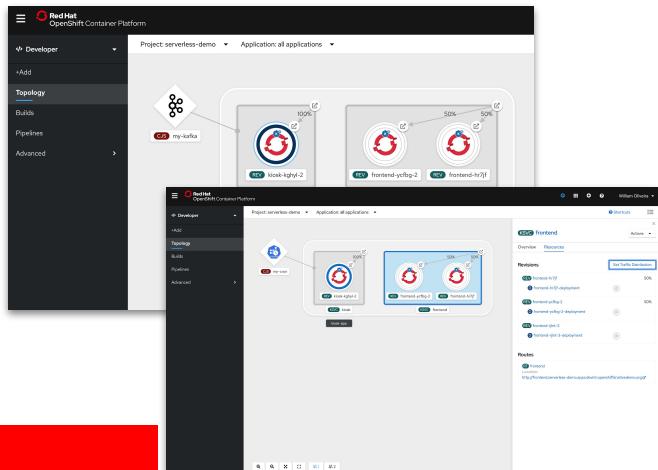
Virtual



Private cloud



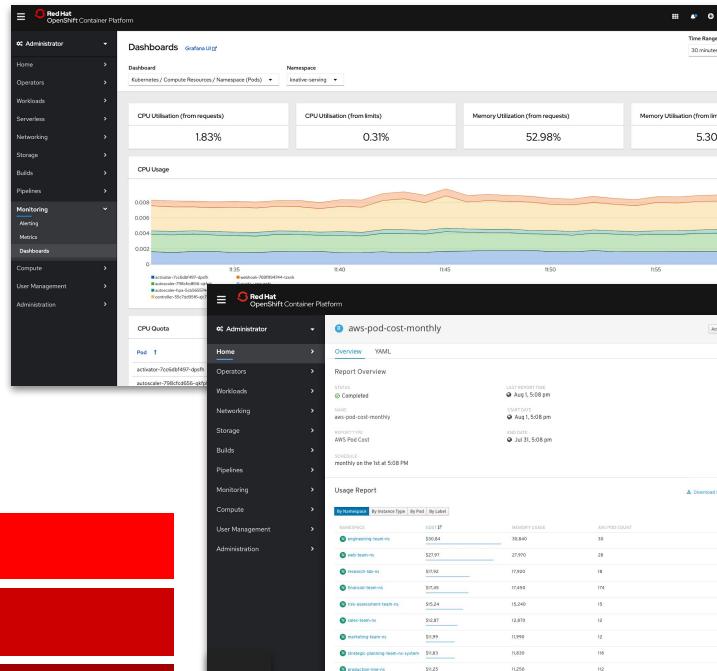
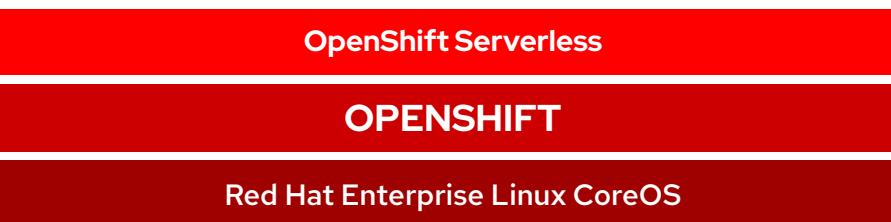
Public cloud





Admin experience

- Monitoring, Metering and Logging
- Disconnected install support (air-gapped)
- Egress proxy with TLS support
- Over the air updates and patches



Physical



OPENSSHIFT

Virtual



Private cloud



Public cloud



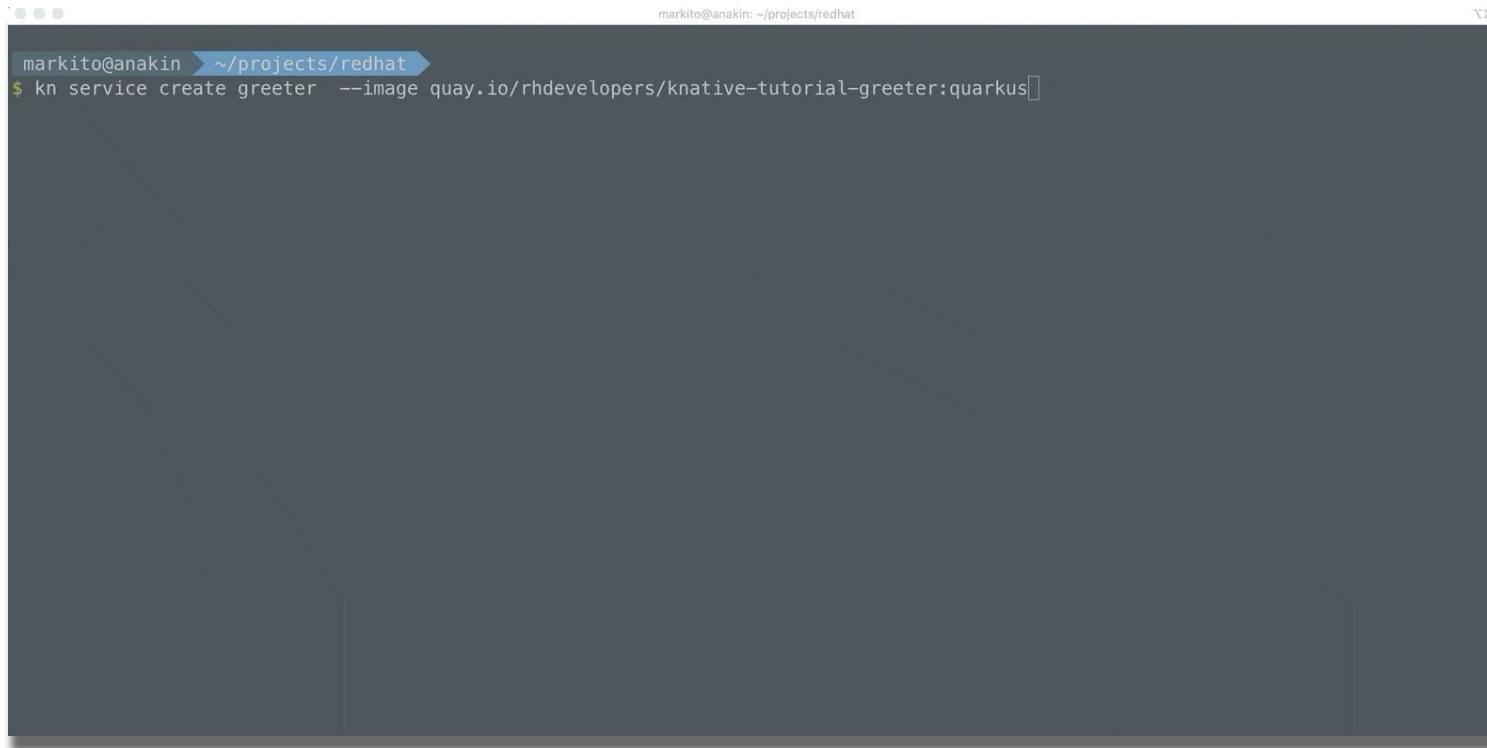
Command Line Experience

```
$ kn service create myService --image=[registry/mycontainer:v1] --min-scale=1 --max-scale=100
```

```
$ kn service update myService --traffic myService-rev1=50,myService-rev2=50
```

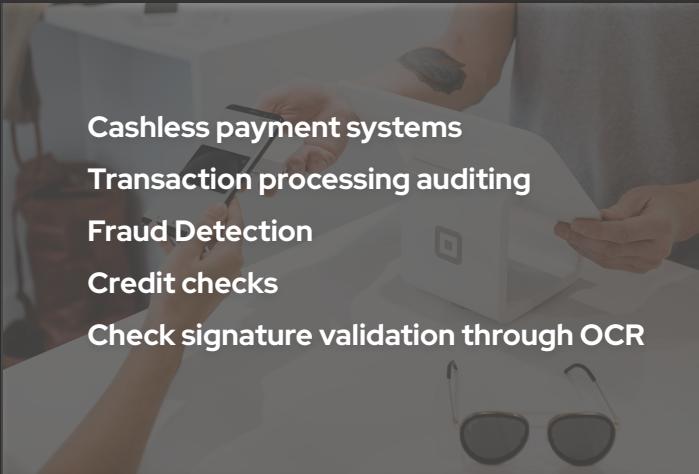
```
$ kn source cronjob create my-cron --schedule "* * * * */1" --data "ping" --sink svc:myService
```

Hello World with Quarkus!



markito@anakin ~/projects/redhat

```
$ kn service create greeter --image quay.io/rhdevelopers/knative-tutorial-greeter:quarkus
```



Cashless payment systems
Transaction processing auditing
Fraud Detection
Credit checks
Check signature validation through OCR

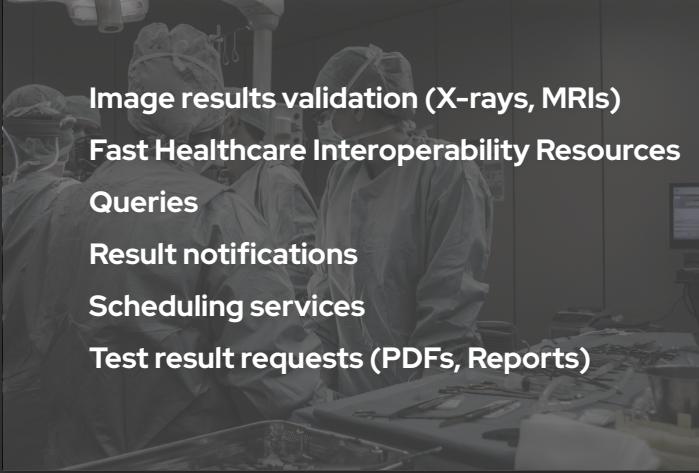
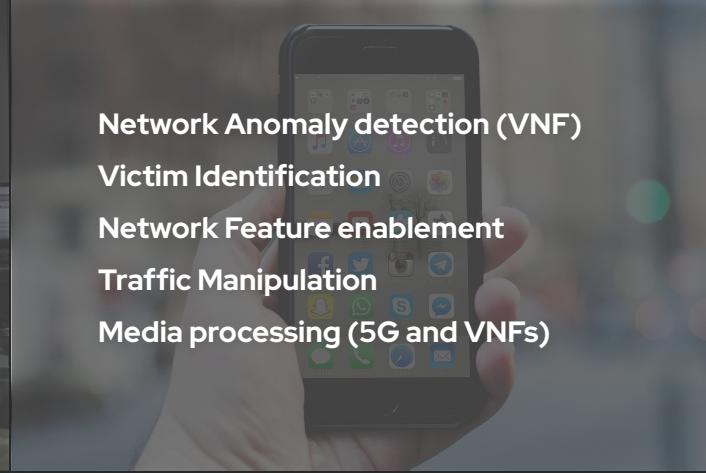


Image results validation (X-rays, MRIs)
Fast Healthcare Interoperability Resources
Queries
Result notifications
Scheduling services
Test result requests (PDFs, Reports)



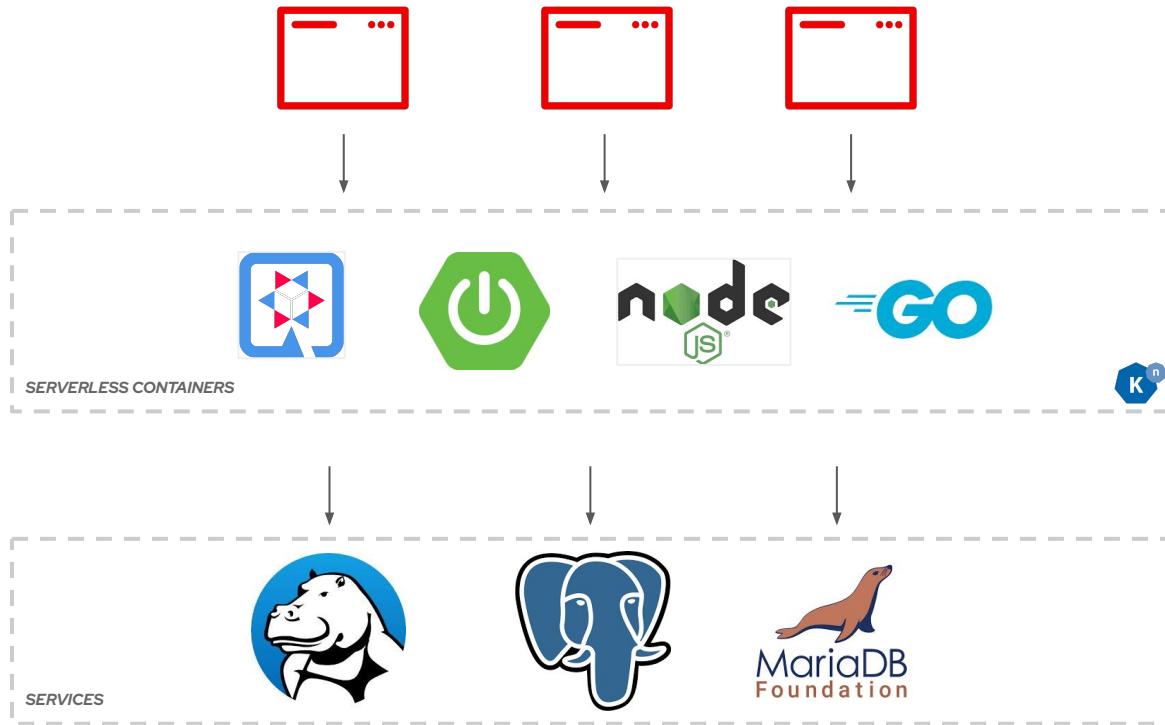
Product thumbnail generation
Chatbots and CRM functions
Marketing Campaign notifications
Sales Audit
Content Push



Network Anomaly detection (VNF)
Victim Identification
Network Feature enablement
Traffic Manipulation
Media processing (5G and VNFs)



Web Applications and APIs



Language or runtime of your choice:

OpenJDK



python™



django



VERT.X



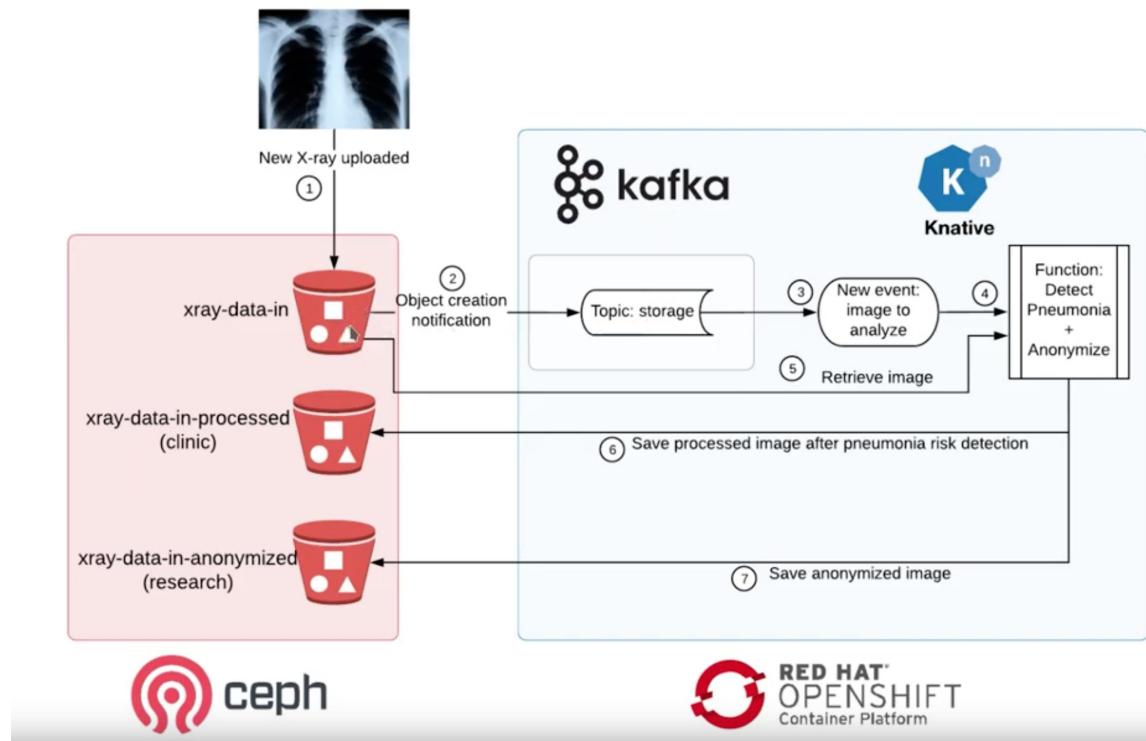
RAILS



Data Transformation

Common Use cases

- Image analysis* (medical, AI/ML, media)
- Video format transcoding
- File type conversion (financial, medical)
- Data extraction
- Lightweight Data Transformation
- Invoice Generation



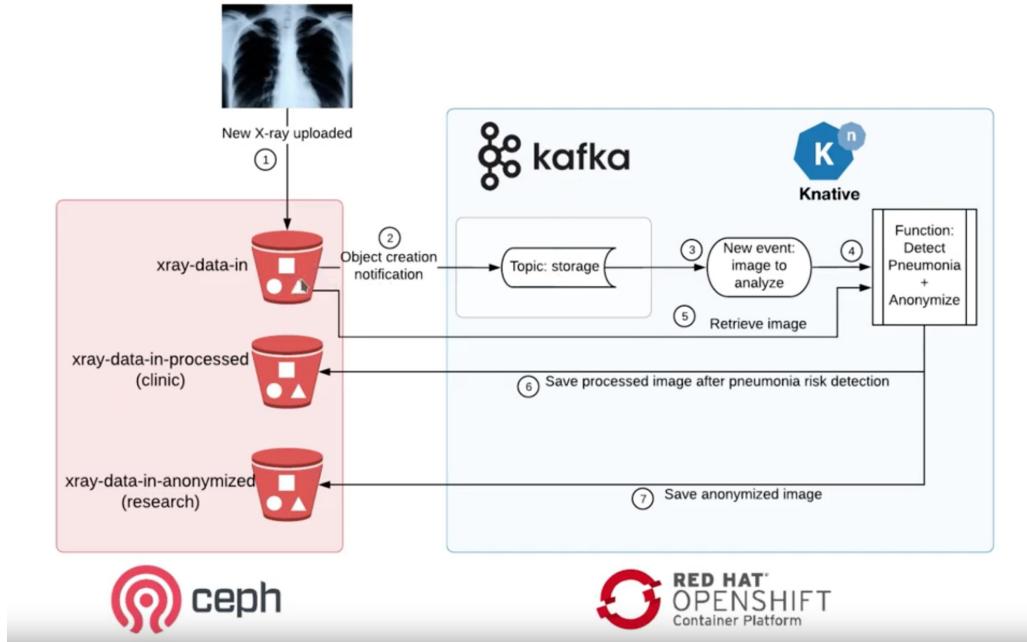


Ceph + Kafka + Serverless

"Automated AI/ML Data Pipelines"

Common Use cases

- Image analysis*
(medical, AI/ML, media)
- Video format transcoding
- File type conversion
(financial, medical)
- Data extraction
- Data Transformation
- Invoice Generation





Building on OpenShift Serverless with Red Hat Services

Connected Services

How Knative services interact with the outside world.



Service Orchestrator

Composing multiple services together into an application.



Event Streaming

All modern architectures need some Kafka.



API Gateway

Next gen APIs still require management.



Implementing Services

Functions, languages, and the vagaries of cold starts.



The Dirty Word in Serverless

Yep, you still need state to handle long-lived orchestration.

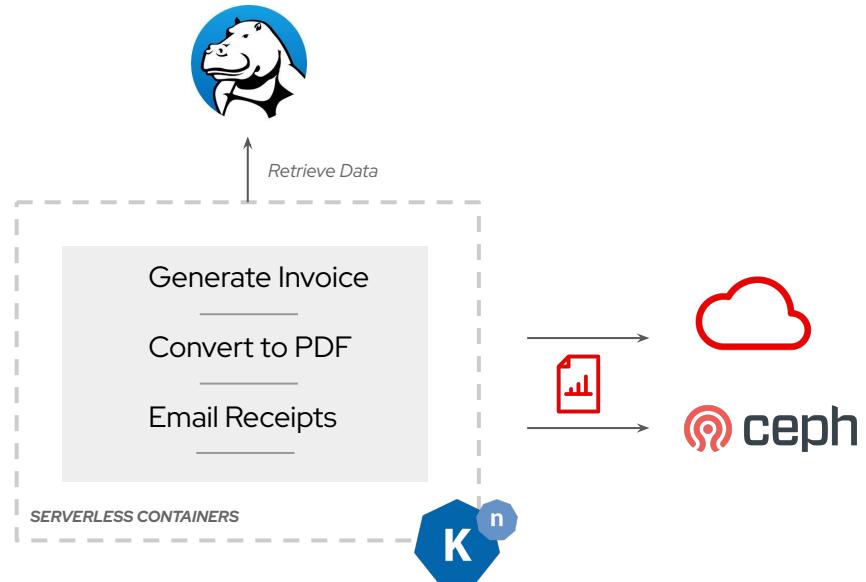




Scheduled Apps Cron Jobs



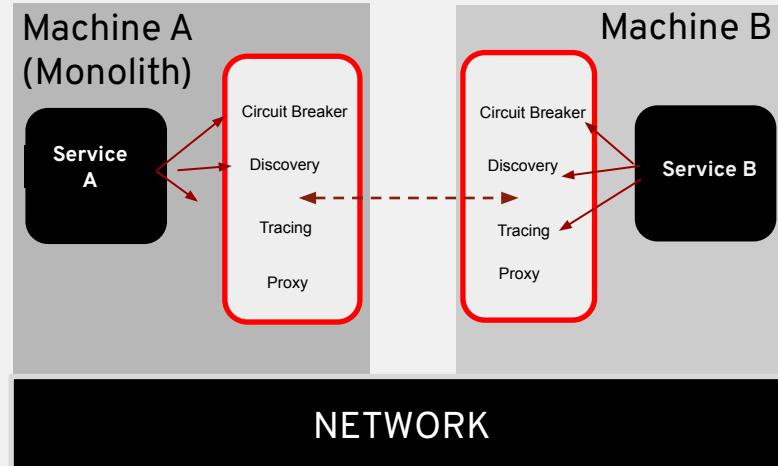
- Every Hour →
- Every Day →
- Every Week →
- Every Month →



Optional - ServiceMesh / Istio

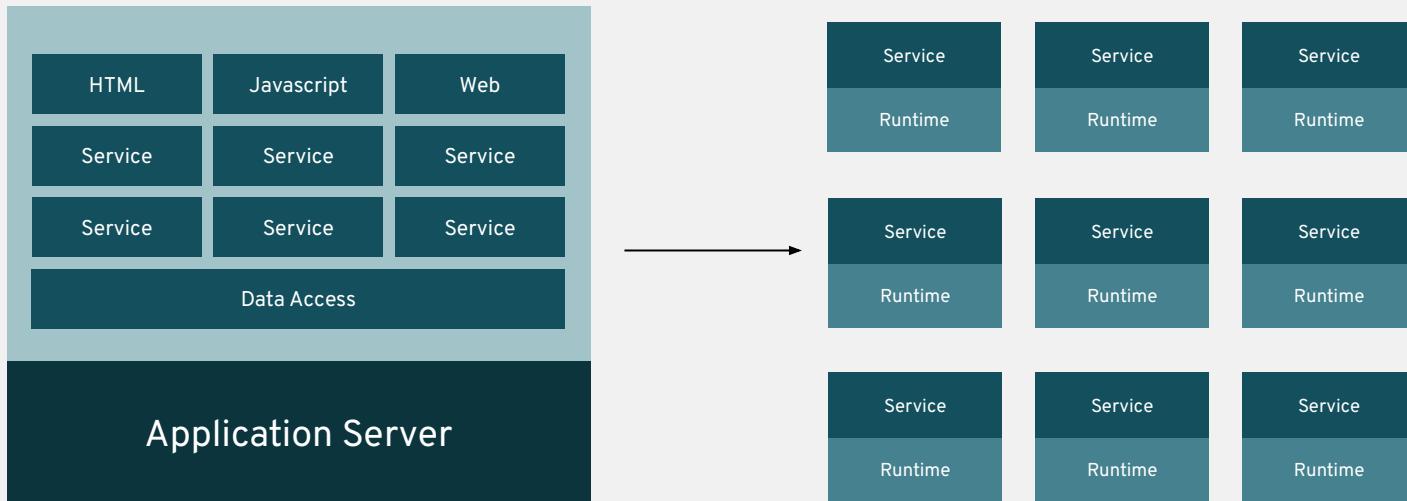
OVERVIEW

WHAT IS A SERVICEMESH ?



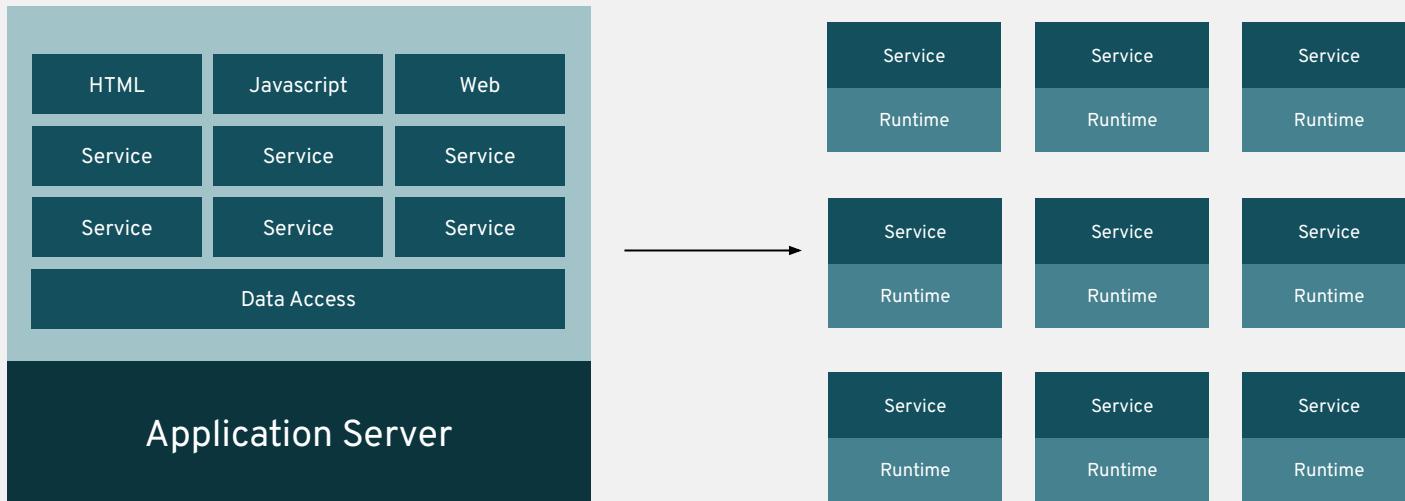
UNDER THE HOOD

MICROSERVICES ARCHITECTURE

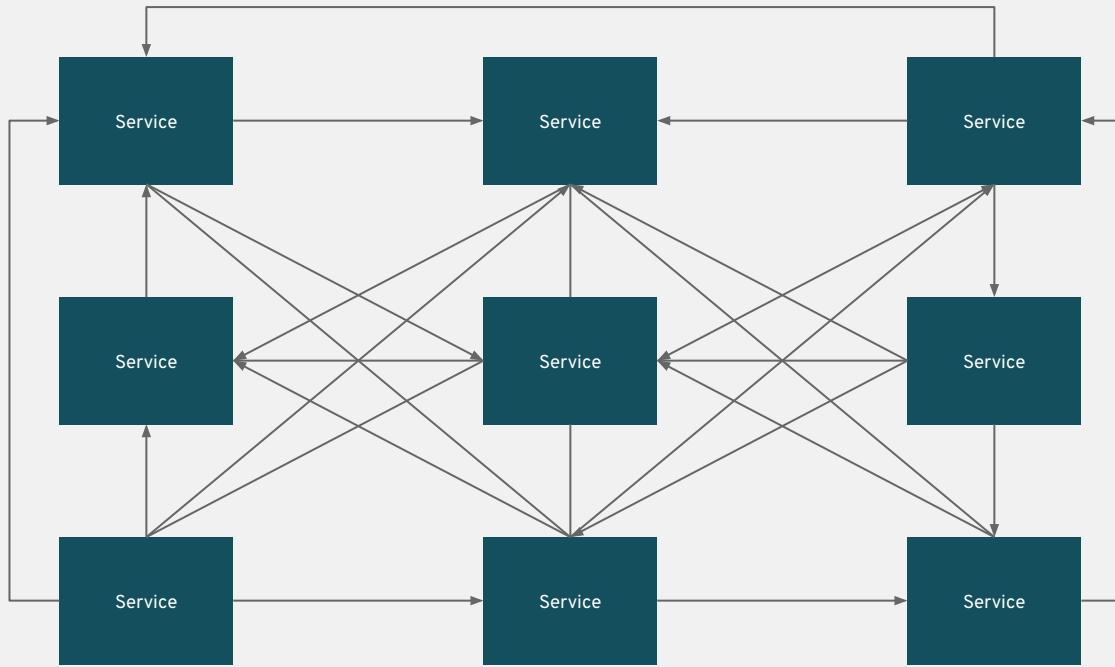


MICROSERVICES ARCHITECTURE

DISTRIBUTED



DISTRIBUTED ARCHITECTURE





HOW TO DEAL WITH THE COMPLEXITY?

DEPLOYMENT

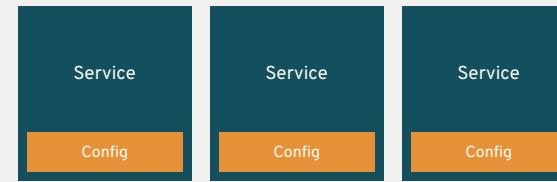
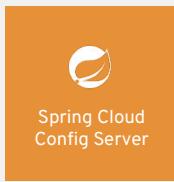
Service
Container

Service
Container

Service
Container

INFRASTRUCTURE

CONFIGURATION

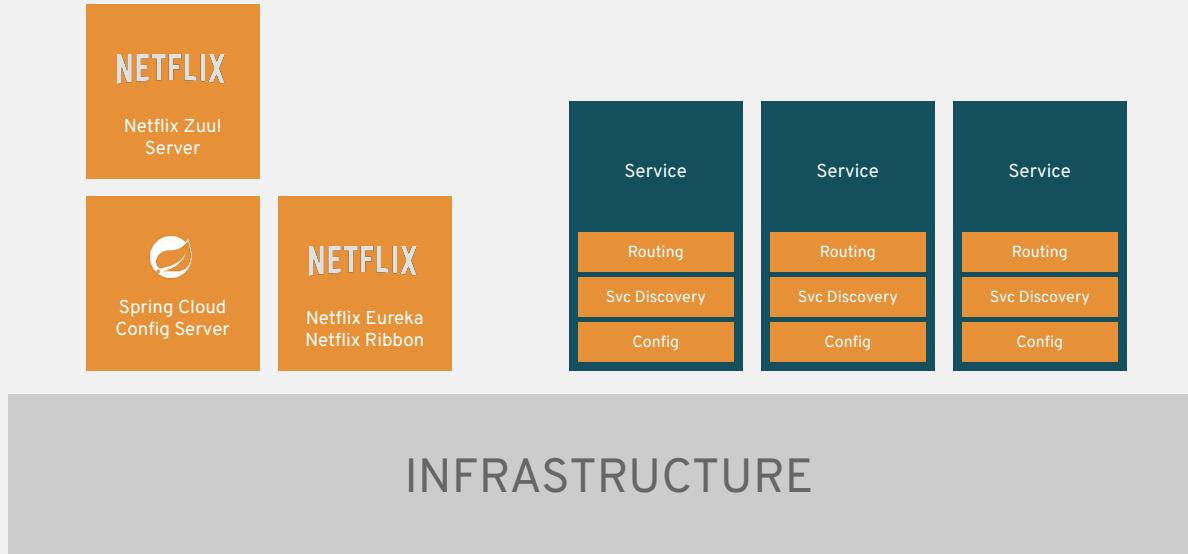


INFRASTRUCTURE

SERVICE DISCOVERY



DYNAMIC ROUTING



FAULT TOLERANCE



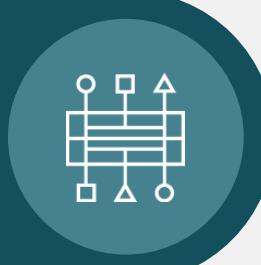
TRACING AND VISIBILITY



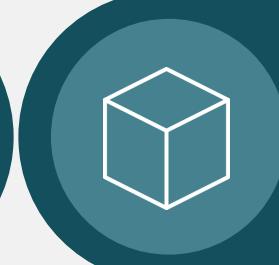
INFRASTRUCTURE

WHAT ABOUT...?

POLYGLOT
APPS



EXISTING
APPS





**THERE SHOULD BE A
BETTER WAY**



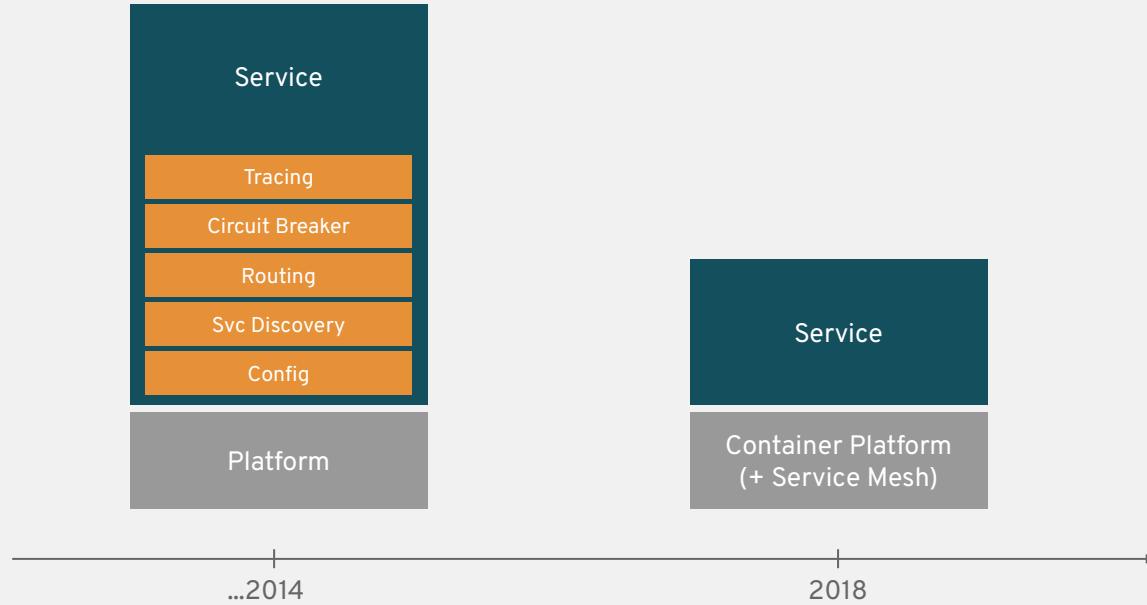


ADDRESS THE COMPLEXITY IN THE INFRASTRUCTURE

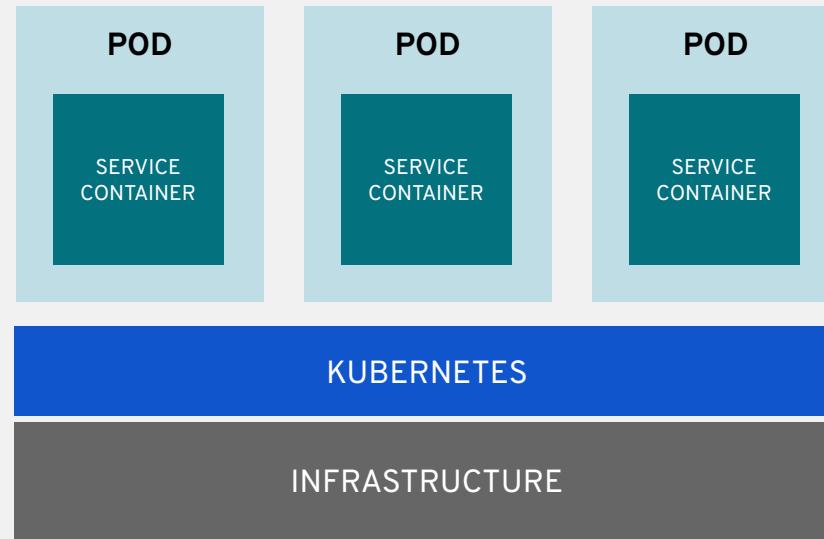
SERVICE MESH

A dedicated infrastructure layer for
service-to-service communications

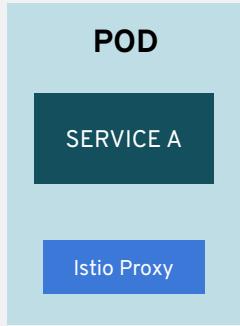
MICROSERVICES EVOLUTION



AUTOMATING CONTAINER DEPLOYMENT



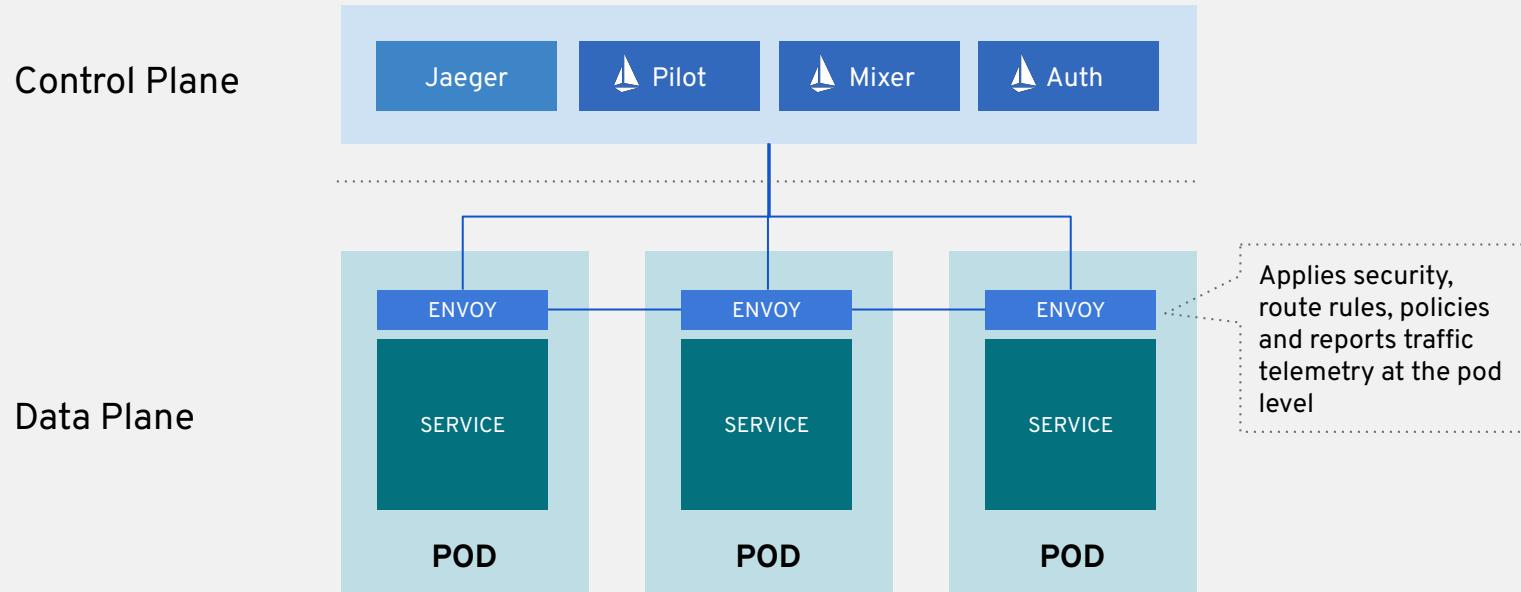
SIDECARS



- Two or more containers deployed to same pod
- Share
 - Same
 - Namespace
 - Pod IP
 - Shared lifecycle
- Used to enhance the co-located containers
- Istio Proxy (L7 Proxy)
 - Proxy all network traffic in and out of the app container

Source: <http://blog.kubernetes.io/2015/06/the-distributed-system-toolkit-patterns.html>

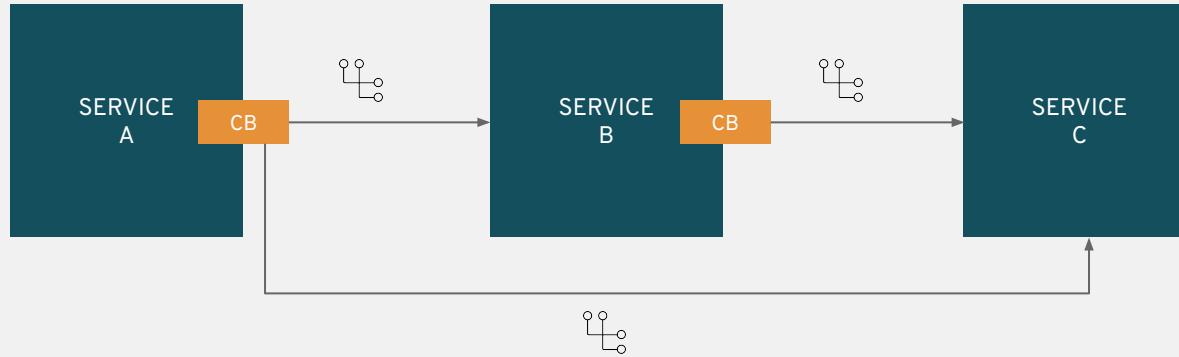
SERVICE MESH ARCHITECTURE



MAJOR FUNCTIONALITY

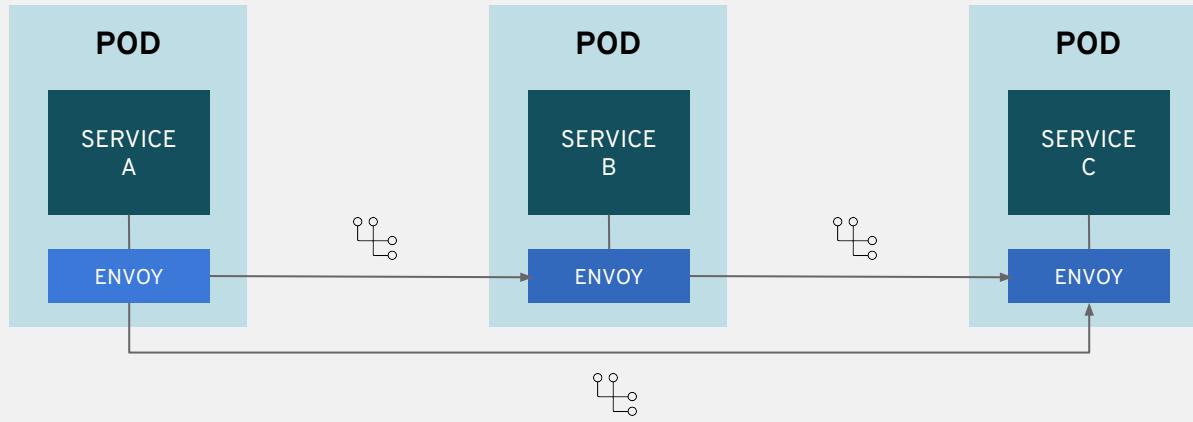
FAULT TOLERANCE

CIRCUIT BREAKERS WITHOUT ISTIO



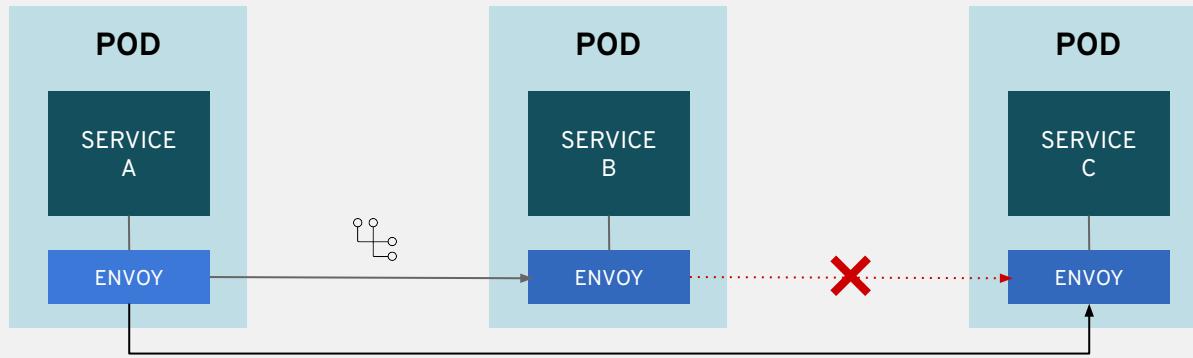
coupled to the service code

CIRCUIT BREAKERS WITH ISTIO



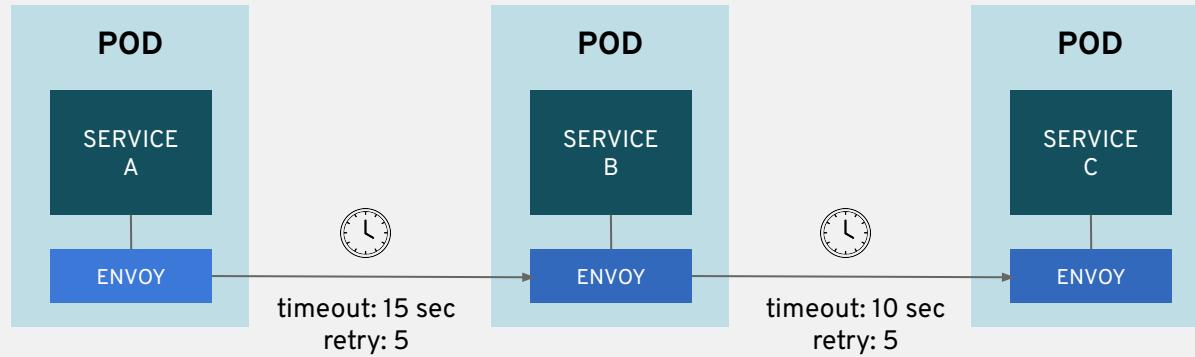
transparent to the services

CIRCUIT BREAKERS WITH ISTIO



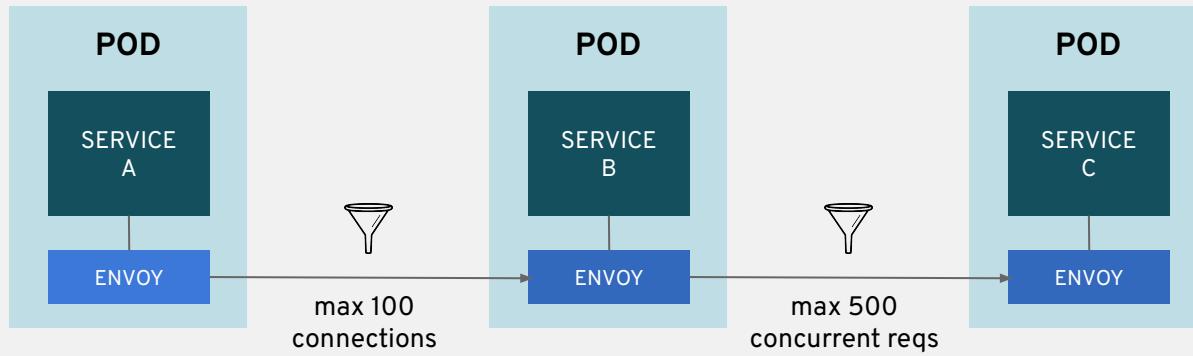
improved response time with global circuit status

TIMEOUTS AND RETRIES WITH ISTIO



configure timeouts and retries, transparent to the services

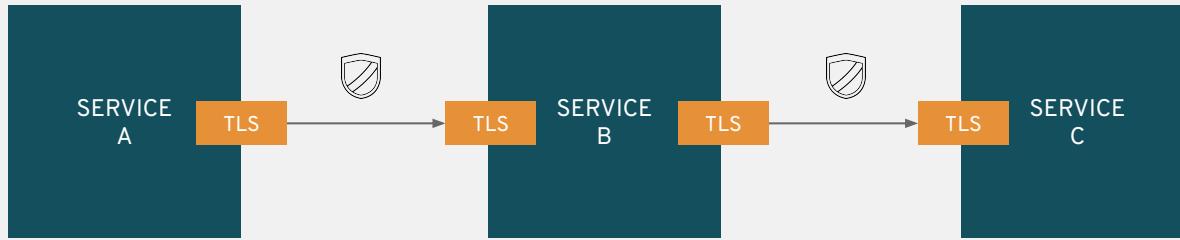
RATE LIMITING WITH ISTIO



limit invocation rates, transparent to the services

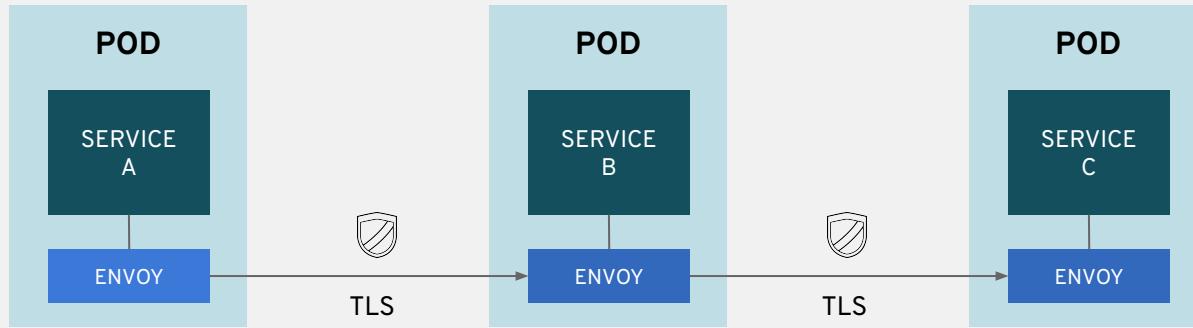
SERVICE SECURITY

SECURE COMMUNICATION WITHOUT ISTIO



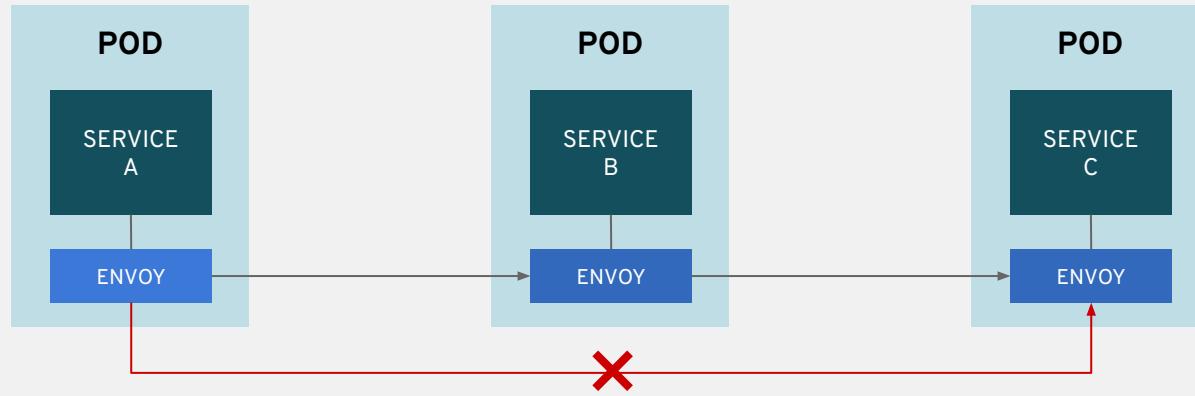
coupled to the service code

SECURE COMMUNICATION WITH ISTIO



mutual TLS authentication, transparent to the services

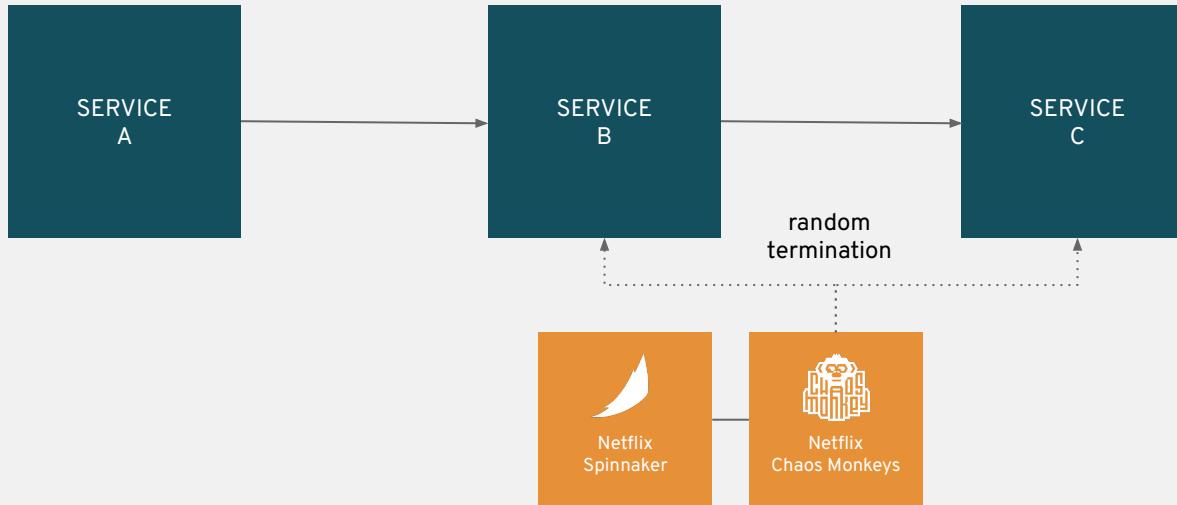
CONTROL SERVICE ACCESS WITH ISTIO



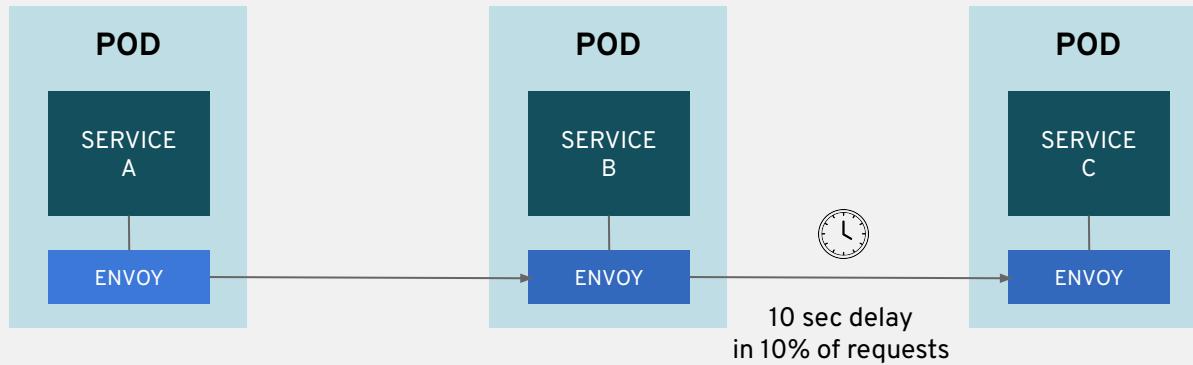
control the service access flow, transparent to the services

CHAOS ENGINEERING

CHAOS ENGINEERING WITHOUT ISTIO

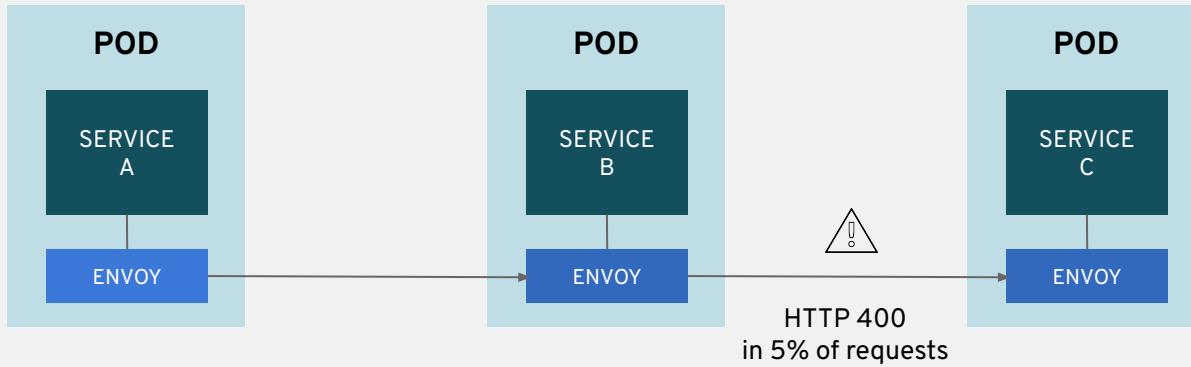


CHAOS ENGINEERING WITH ISTIO



inject delays, transparent to the services

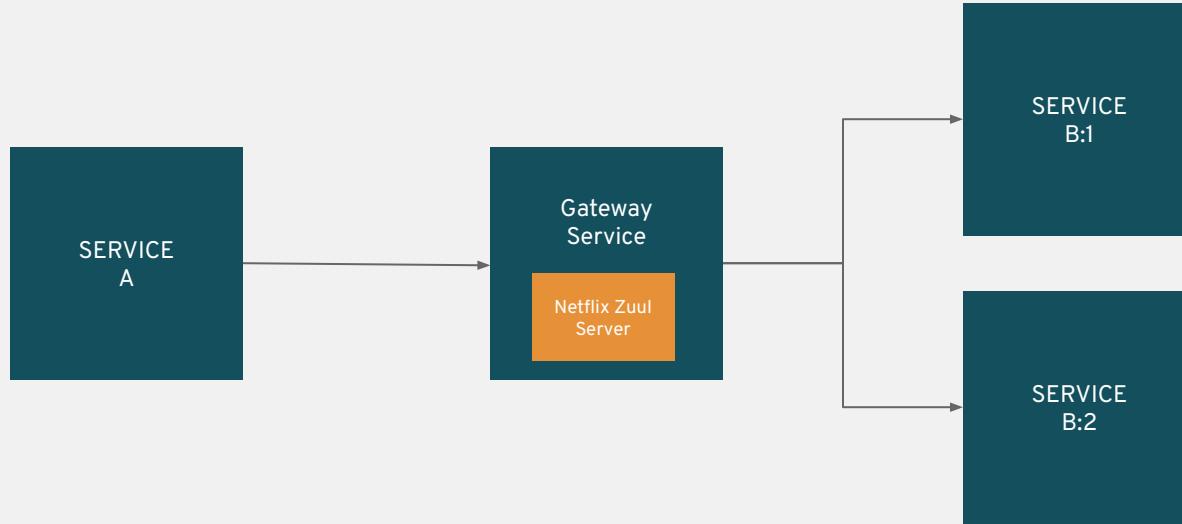
CHAOS ENGINEERING WITH ISTIO



inject protocol-specific errors, transparent to the services

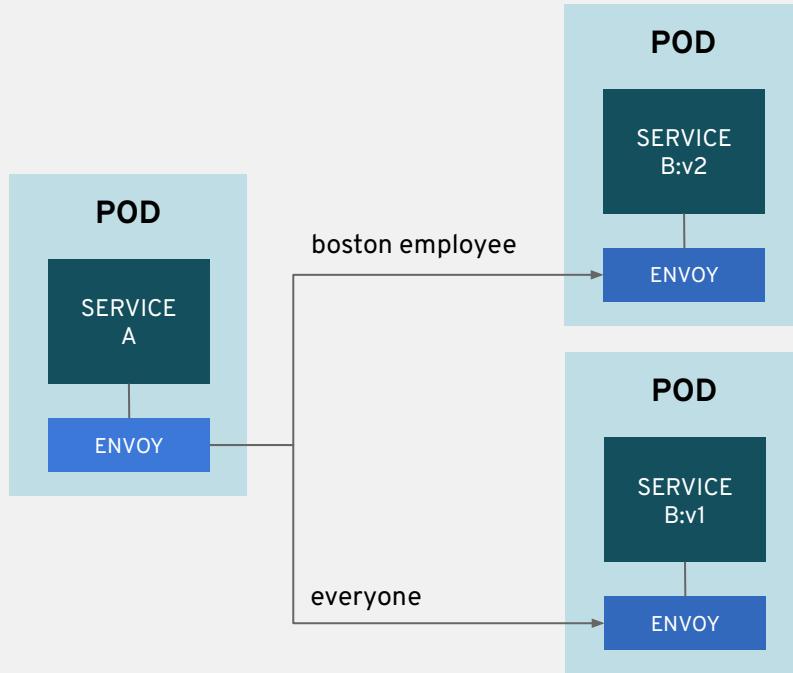
DYNAMIC ROUTING

DYNAMIC ROUTING WITHOUT ISTIO

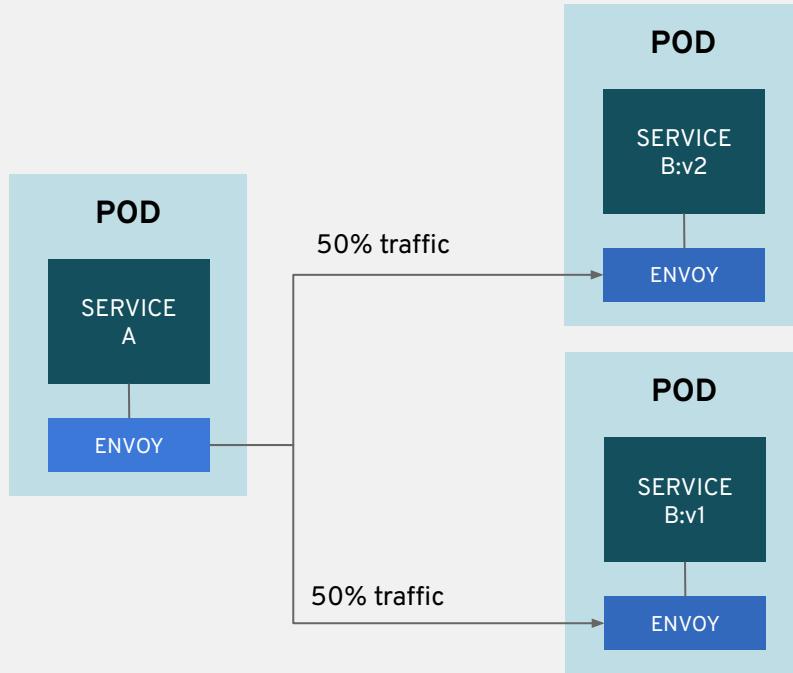


custom code to enable dynamic routing

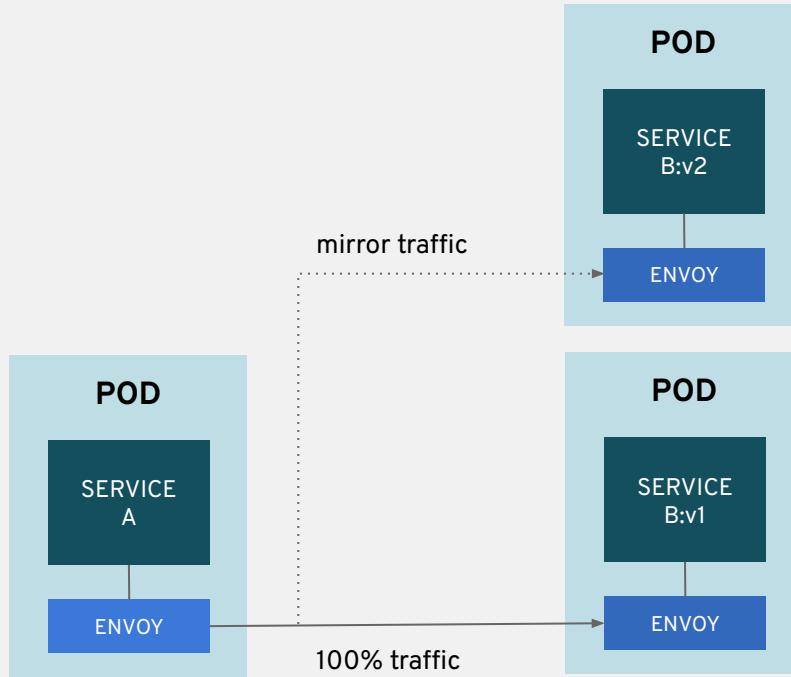
CANARY DEPLOYMENT WITH ISTIO



A/B DEPLOYMENT WITH ISTIO



DARK LAUNCHES WITH ISTIO



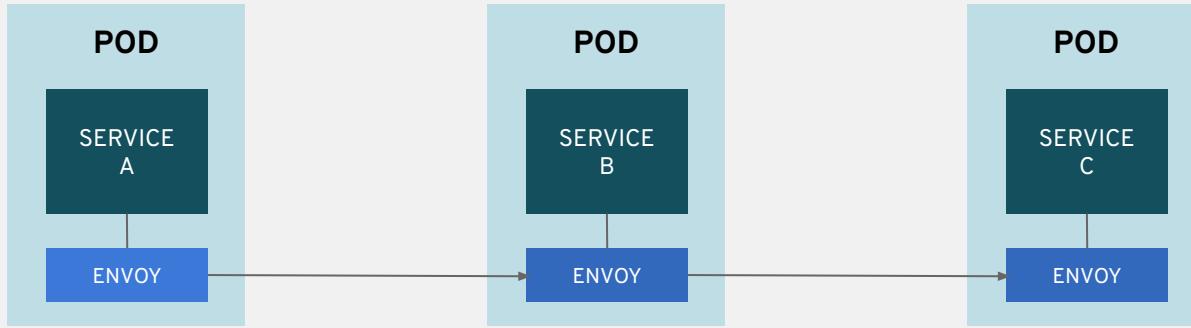
DISTRIBUTED TRACING (JAEGER)

DISTRIBUTED TRACING WITHOUT ISTIO



code to enable dynamic tracing

DISTRIBUTED TRACING WITH ISTIO & JAEGER



discovers service relationships and process times,
transparent to the services



SERVICE MESH OBSERVABILITY (KIALI)

kiali

Overview

Graph

Applications

Workloads

Services

Istio Config

Distributed Trac...

Namespace: **bookinfo**

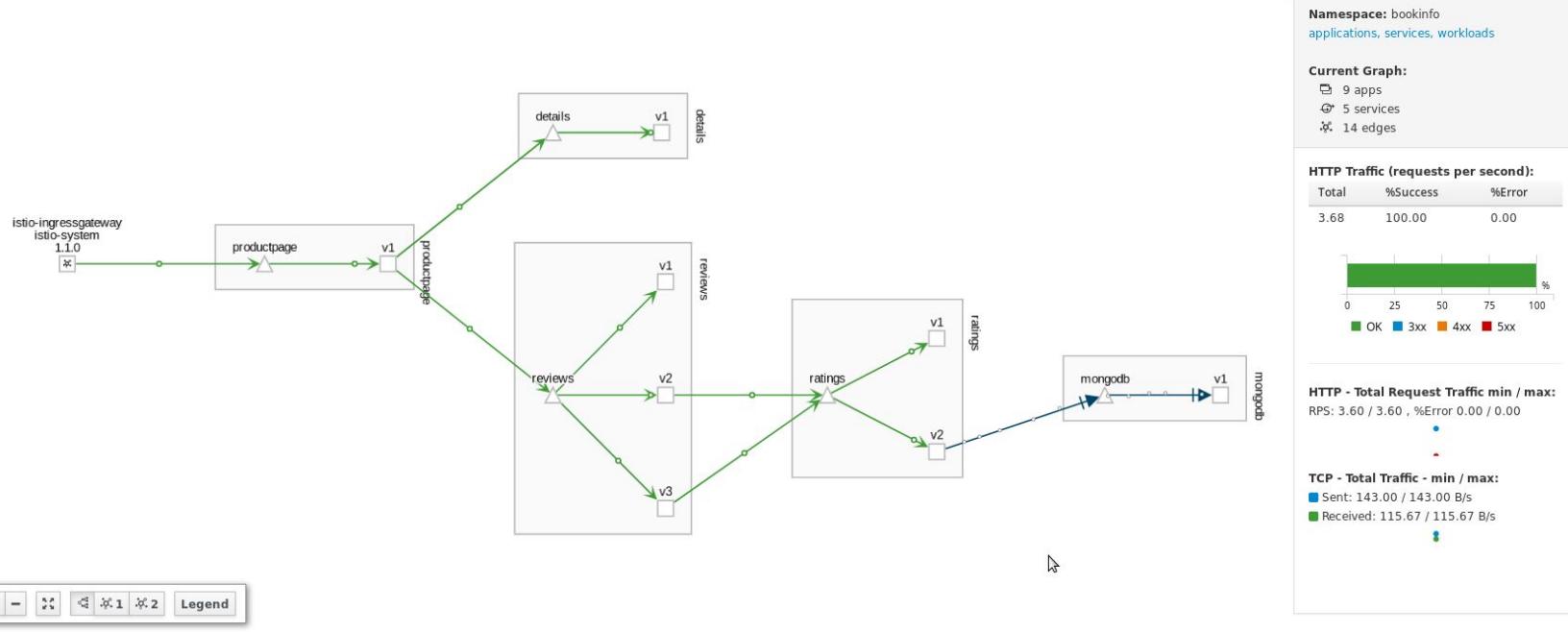
Graph

Display Edge Labels Graph Type Versioned app

Find... Hide...

Feb 18, 16:07:37 ... Feb 18, 16:08:37

Fetching Last min Every 15 sec



Istio - Caveats

- Caveats
 - Be careful, istio is NOT a ServiceBus
 - Be careful, istio is NOT an API Manager
 - Be careful, istio is NOT meant to be a router
 - Be careful, istio is NOT a BPM tool
- Istio is meant to help solving some challenges
 - Centrally encryption / management of internal service communication
 - Distributed networking / communication of services
 - Increase Fault tolerance
 - Fault simulation



Thank you



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



twitter.com/RedHat