redhat.

MONOLITHS TO MICROSERVICES: APP TRANSFORMATION

Hands-on Technical Workshop Overview

Matt Davis
AppDev SA / Farmer
Email: mattd@redhat.com
@matt_s_davis

WHAT YOU WILL LEARN

- Industry trends around enterprise application development
- Red Hat's approach to application modernization
- How to discuss migration and modernization with your customers or managers
- Migrating an existing legacy Java™ EE app to Red Hat JBoss Enterprise Application Platform on OpenShift.
- Using modern frameworks like Spring Boot, Wildfly Swarm, Eclipse Vert.x, and Node.js to implement cloud native microservices and replace monolithic functionality.
- Developing and deploying using Red Hat OpenShift Container Platform, Red Hat OpenShift Application Runtimes, and DevOps processes.
- The benefits and challenges with microservices, including use cases for reactive microservices.
- Preventing and detecting issues in a distributed system.
- Pros and cons with different packaging techniques for microservices

AGENDA

9:00AM–9:15AM
WELCOME

9:15AM–10:15AM
MOVING EXISTING APPS TO THE CLOUD

10:15AM–11:00AM
DEVELOPER INTRO TO OPENSHIFT

11:00AM–11:15AM
BREAK

11:15AM–12:30PM
MONOLITHS TO MICROSERVICES PART 1

12:30PM–1:00PM
LUNCH

1:00PM–2:15PM
MONOLITHS TO MICROSERVICES PART 2

2:15PM–3:00PM
REACTIVE MICROSERVICES WITH ECLIPSE VERT.X

3:00PM–3:15PM
BREAK

3:15PM–4:45PM
PREVENT AND DETECT DISTRIBUTED APP ISSUES

4:45PM–5:15PM
PACKAGING MICROSERVICES | API-LED MODERNIZATION

5:15PM–5:30PM
WRAP-UP AND CLOSING REMARKS

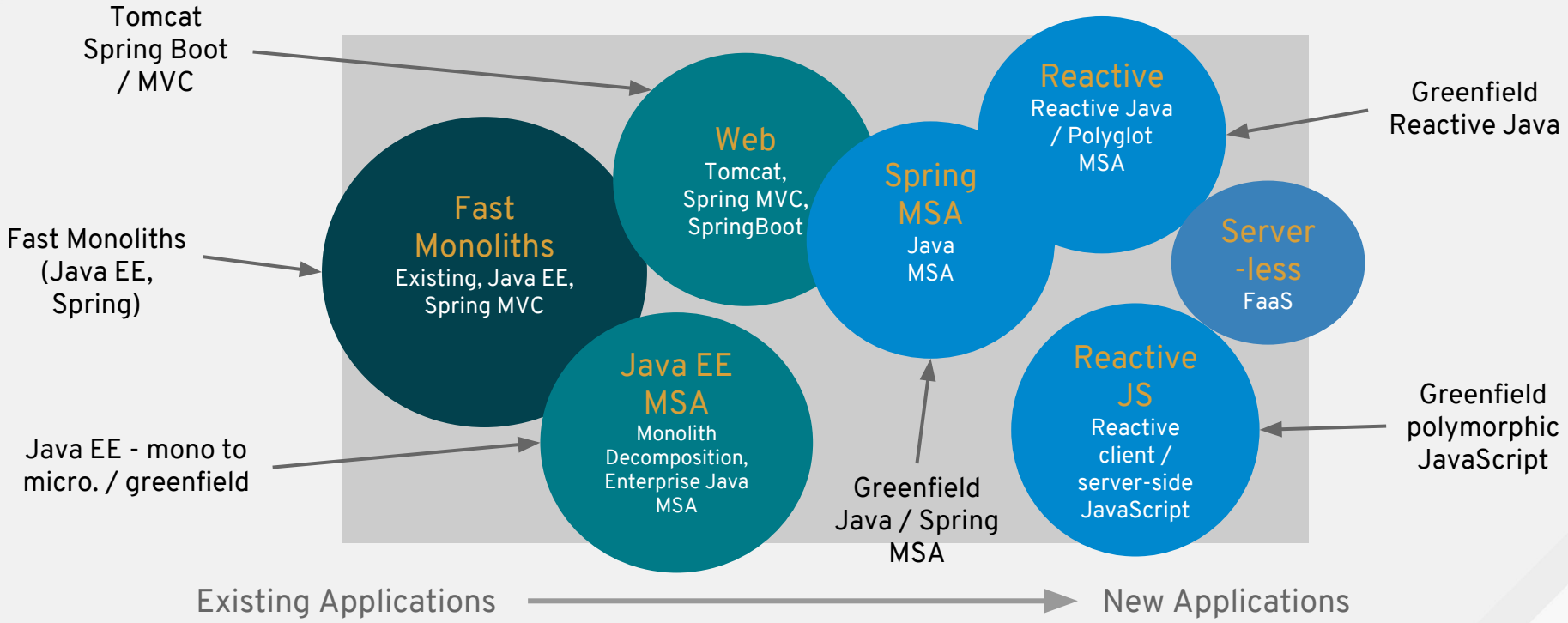
PREREQUISITES

- Laptop with recent browser:
  - Chrome, Firefox, Internet Explorer/Edge 10+, or Safari 9+ installed
- Are a Java developer, architect, or developer team lead interested in learning more about the latest technologies for modern application development
- Working knowledge of Java programming
- Familiarity with Linux container technology and concepts

Hands-on Labs: openshift-modernize-apps.katacoda.com

MOVING EXISTING APPS TO THE CLOUD

THE SPECTRUM OF ENTERPRISE APPS



MIGRATION AND MODERNIZATION APPROACHES

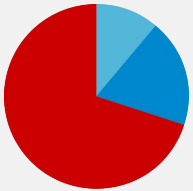
MODERNIZING EXISTING APPS

- Reuse existing functionality and data as much as possible
- Move existing workloads to a modern deployment platform
- Apply new processes, products, and technology to existing apps

DEVELOPING NEW APPLICATIONS

- API-centric polyglot microservices architecture
- Autonomous development teams
- Agile development, continuous deployment, DevOps culture
- Containerized & orchestrated cloud deployments

APPLICATION MODERNIZATION



Existing Apps

How much work required to rewrite?

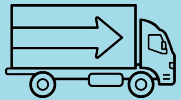
Review
Analyze
Prioritize

- Lift & Shift — Smaller or frozen apps are candidates here
- Connect & Extend — Highly scaled and high rate of change apps are candidates
- Rip & Re-write — Highly scaled and high rate of change apps are candidates
- Repurchase — Not a target
- Retire — Not a target
- Retain as is — Not a target

PATTERNS IN MODERNIZING WORKLOADS

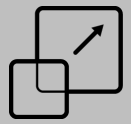
LIFT & SHIFT

- **Containerize** existing workloads
- Deploy them on a **PaaS**
- Keep external integrations and data on legacy
- Legacy applications have to be well written and suited



CONNECT & EXTEND

- **Legacy remains intact**
- New layer - new capabilities
- Deploy on **PaaS**
- **New integration points** between legacy and new layers **(Need for Agile Integration)**



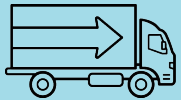
RIP & RE-WRITE

- **Legacy is totally replaced**
- New interfaces and data
- Use **PaaS** to run
- Some data and features can be re-wrapped, but mostly are retired.

PATTERNS IN MODERNIZING WORKLOADS

LIFT & SHIFT

- **Containerize** existing workloads
- Deploy them on a **PaaS**
- Keep external integrations and data on legacy
- Legacy applications have to be well written and suited



FOCUS FOR THIS SECTION

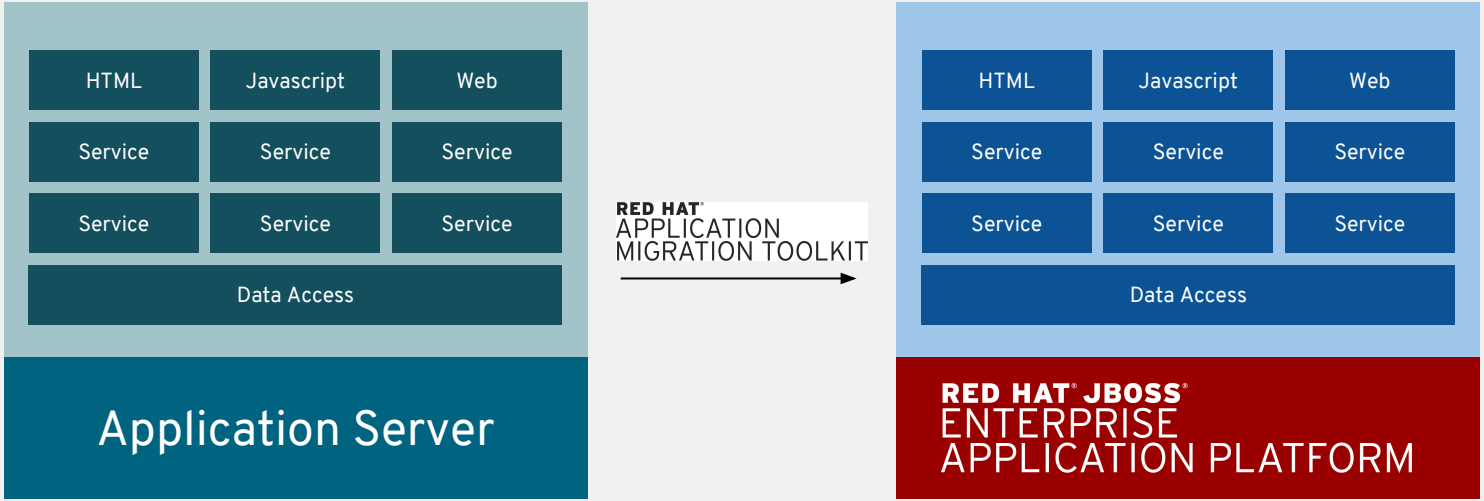
CONNECT & EXTEND

- **Legacy remains intact**
- New layer - new capabilities
- Deploy on **PaaS**
- **New integration points** between legacy and new layers **(Need for Agile Integration)**

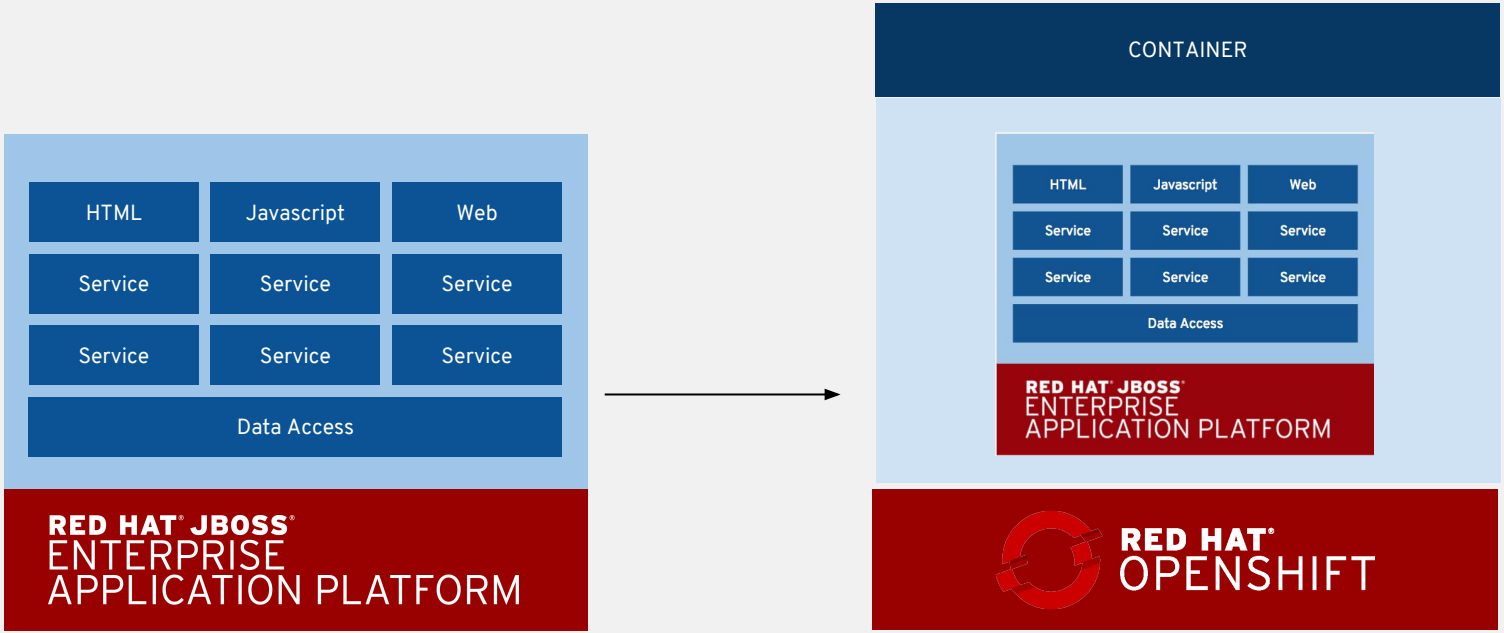
RIP & RE-WRITE

- **Legacy is totally replaced**
- New interfaces and data
- Use **PaaS** to run
- Some data and features can be re-wrapped, but mostly are retired.

LIFT-AND-SHIFT MONOLITH TO CLOUD



LIFT-AND-SHIFT MONOLITH TO CLOUD



Majestic Monolith

https://m.signalvnoise.com/the-majestic-monolith-29166d022228

MAJESTIC (FAST-MOVING) MONOLITH

- Large organizations have a tremendous amount of resources **invested in existing** monolith applications
- Looking for a **sane way** to capture the benefits of containers and orchestration **without having to complete rewrite**
- **OpenShift** provides the platform for their existing investment with the benefit of a **path forward** for microservice based apps in the future

Why migrate to JBoss EAP?

Runtime [1][2] (framework)	Boot time server only	Boot time including app deployment	Memory usage without load	Memory usage under load	Measured[3] 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

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

[1] The microservice is a simple REST application.
[2] All runtimes are using their default settings
[3] The performance test was conducted with ApacheBench using 500K request with 50 users and keep-alive enabled.

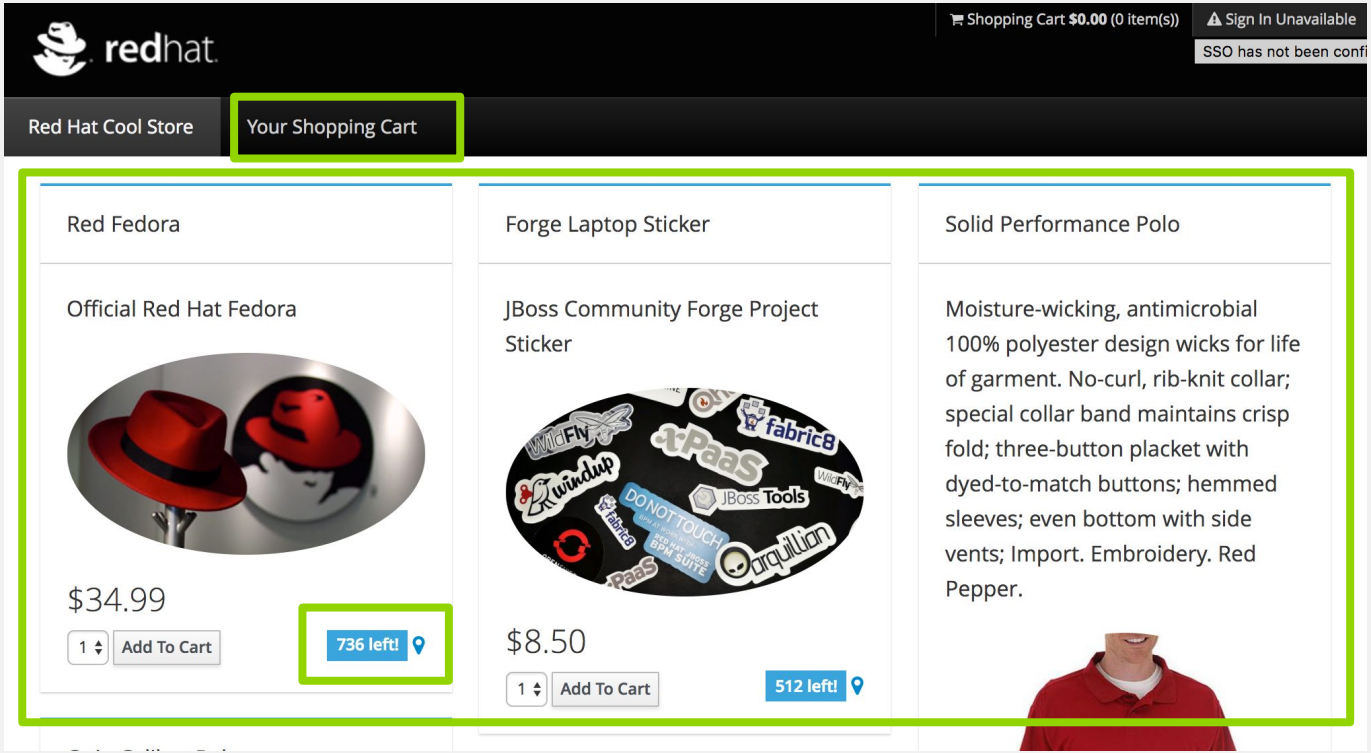
LAB: MOVING EXISTING APPS TO THE CLOUD

GOAL FOR LAB

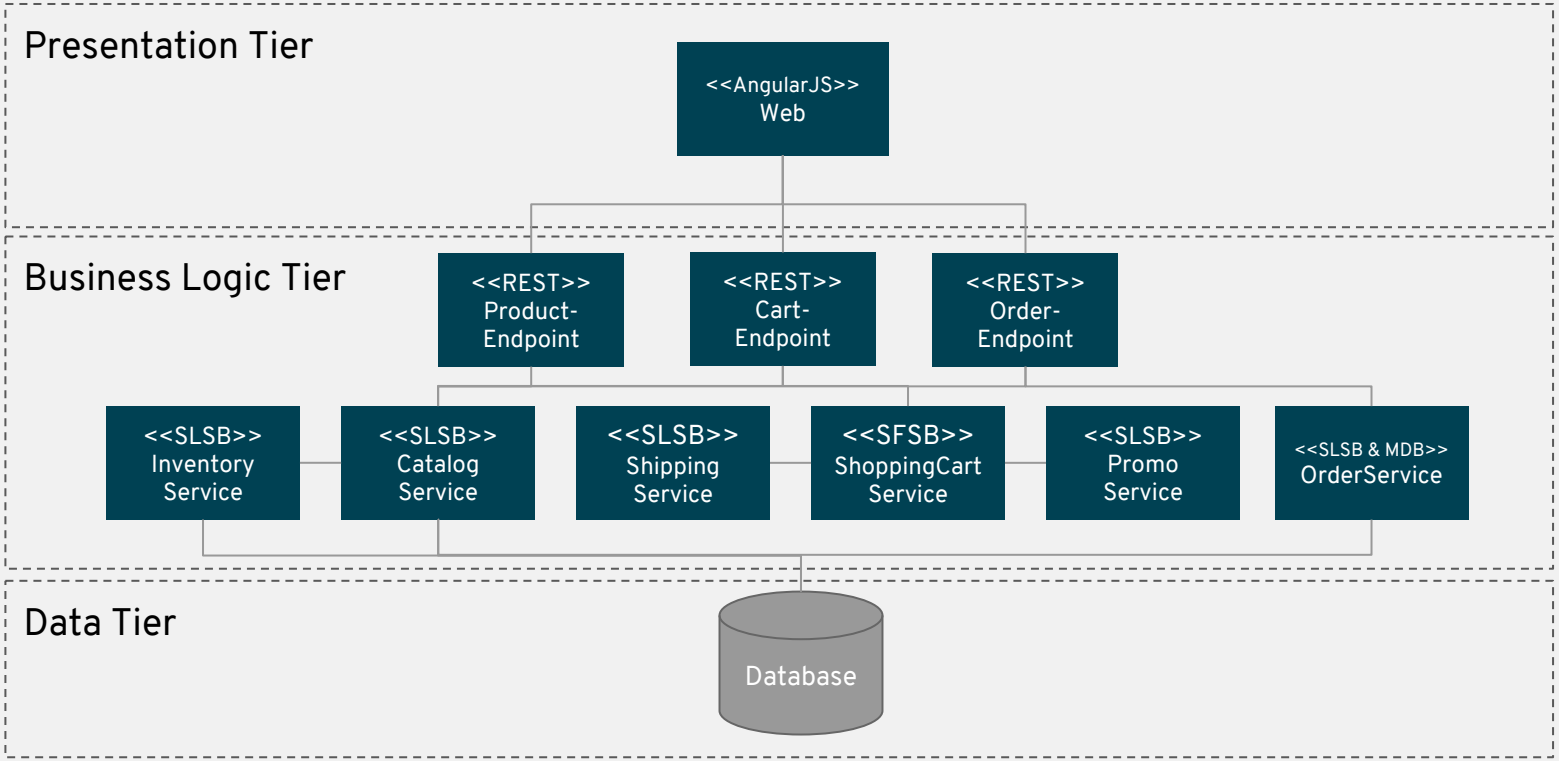
In this lab you will learn:

- How to use lab environment for today
- How to migrate an existing legacy Java EE application (CoolStore) from Weblogic to **JBoss EAP** using **Red Hat Application Migration Toolkit**
- How to deploy the result to **OpenShift container platform** to create a *Fast Moving Monolith*
- Different alternatives to building and deploying an application

COOLSTORE APPLICATION



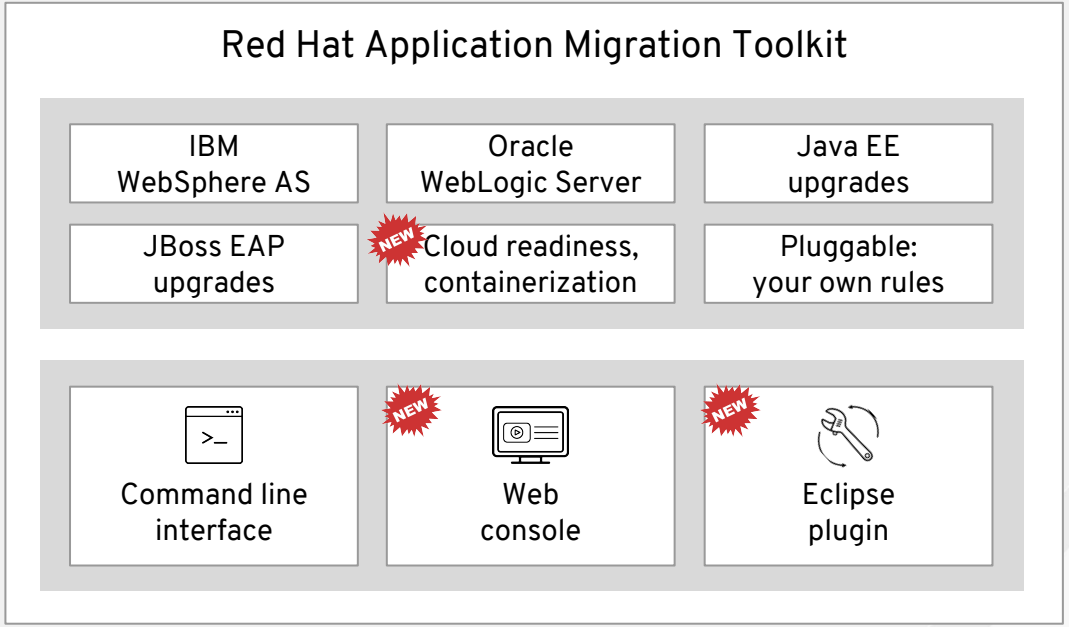
COOLSTORE APPLICATION



RED HAT® APPLICATION MIGRATION TOOLKIT

Catalyze large scale application modernizations and migrations

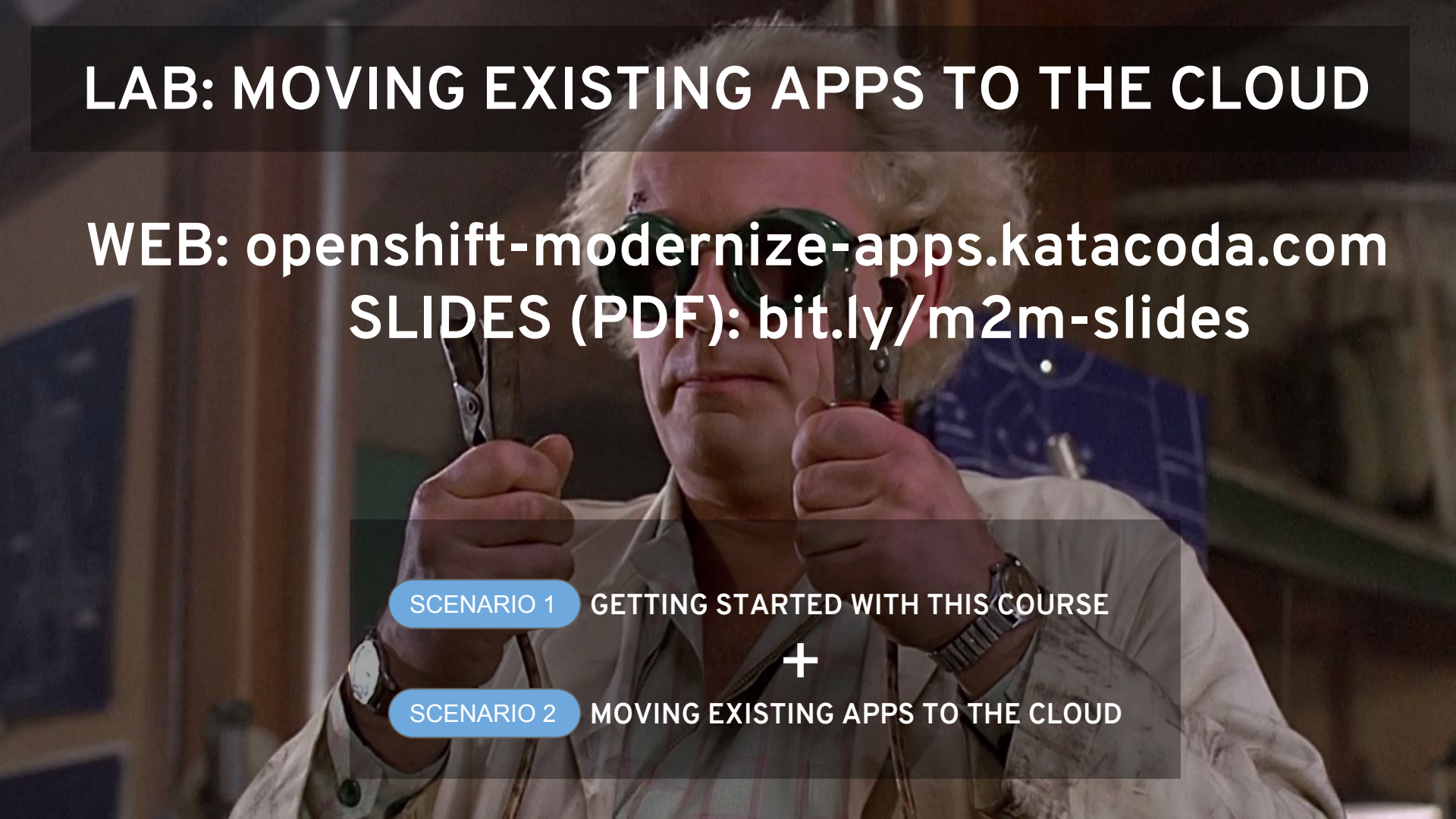
- Automate analysis
- Support effort estimation
- Accelerate code migration
- Free & Open Source



LAB: MOVING EXISTING APPS TO THE CLOUD

WEB: openshift-modernize-apps.katacoda.com

SLIDES (PDF): bit.ly/m2m-slides



SCENARIO 1 GETTING STARTED WITH THIS COURSE

+

SCENARIO 2 MOVING EXISTING APPS TO THE CLOUD

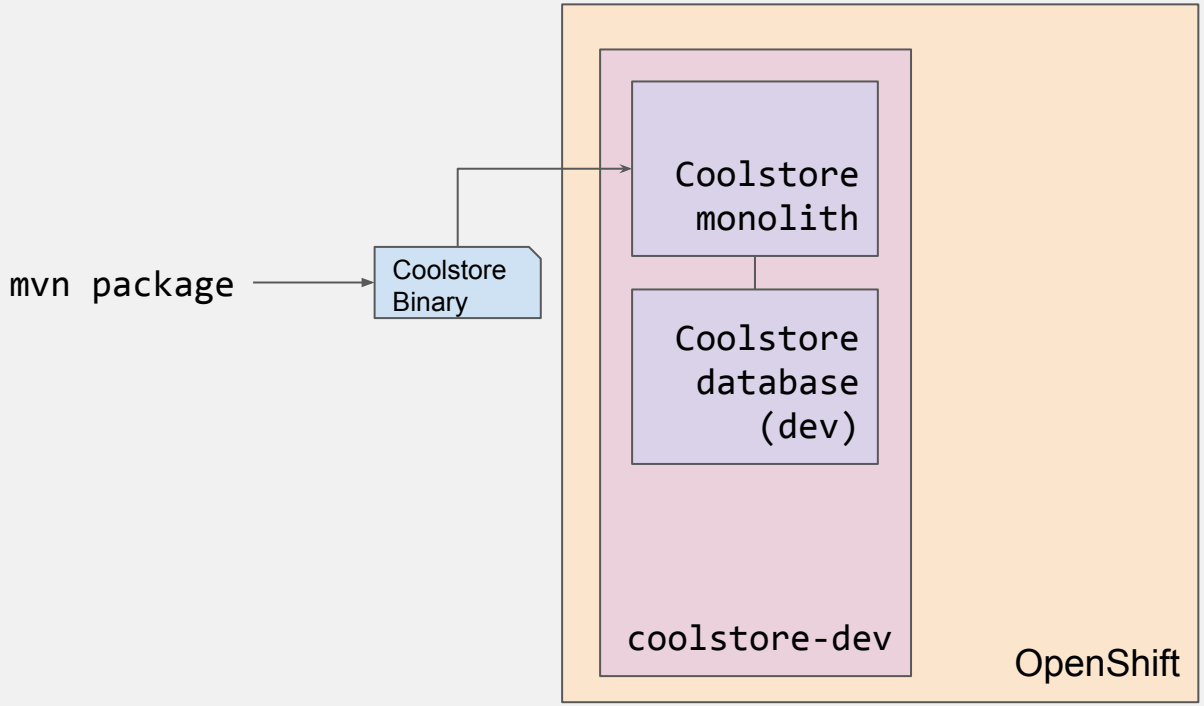
WRAP-UP AND DISCUSSION

RESULT OF LAB

In this lab you:

- Familiarized yourself with the Lab environment
- Migrated the CoolStore monolith from Weblogic to **JBoss EAP** using **Red Hat Application Migration Toolkit**
- Created a new development project on **OpenShift**
- Deployed the migrated app to OpenShift using a Template and a Binary Build
- In the next lab you will explore OpenShift deeper as a developer

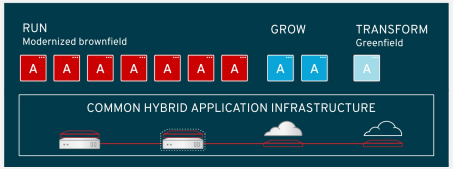
RESULT OF LAB



RED HAT APPLICATION MIGRATION & MODERNIZATION PROGRAM

Red Hat provides the most comprehensive technologies, tools and services to support you

TODAY and TOMORROW



COMBINE TRANSFORMATION

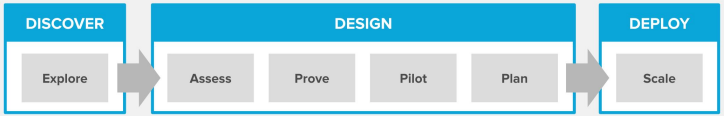
CORE MIGRATION EXISTING & NEW WORKLOADS	APPLICATION SERVERS	ESB & INTEGRATION PLATFORMS	BPM & DECISION MANAGEMENT	APPLICATION INFRASTRUCTURE
MODERNIZATION INITIATIVES ENABLING BUSINESS VELOCITY	BETTER SOFTWARE ARCHITECTURE	AGILE INTEGRATION	STREAMLINE APPLICATION LIFECYCLE	CONTINUOUS INNOVATION

Migration → Modernization

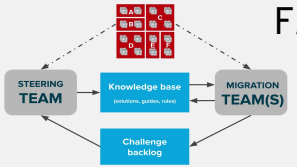
Making old apps new again ← Modern app development

BENEFITS

APPROACH



FACTORY



redhat.

JUMPSTART YOUR MODERNIZATION WITH RED HAT OPEN INNOVATIONS LABS

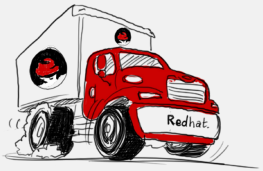
MODERNIZE TRADITIONAL APPS
- Extend applications
- Optimize applications
- Scale applications
- Expose to orchestration

INNOVATION ACCELERATED

DEVELOP CONTEMPORARY APPS
- Develop on PaaS environment
- Transform how you design and develop apps
- Adopt lean and agile principles
- Master DevOps practices

COLLABORATION

Space to work,
innovate, and discuss



RESIDENCY

An eight-week accelerated
teaming engagement

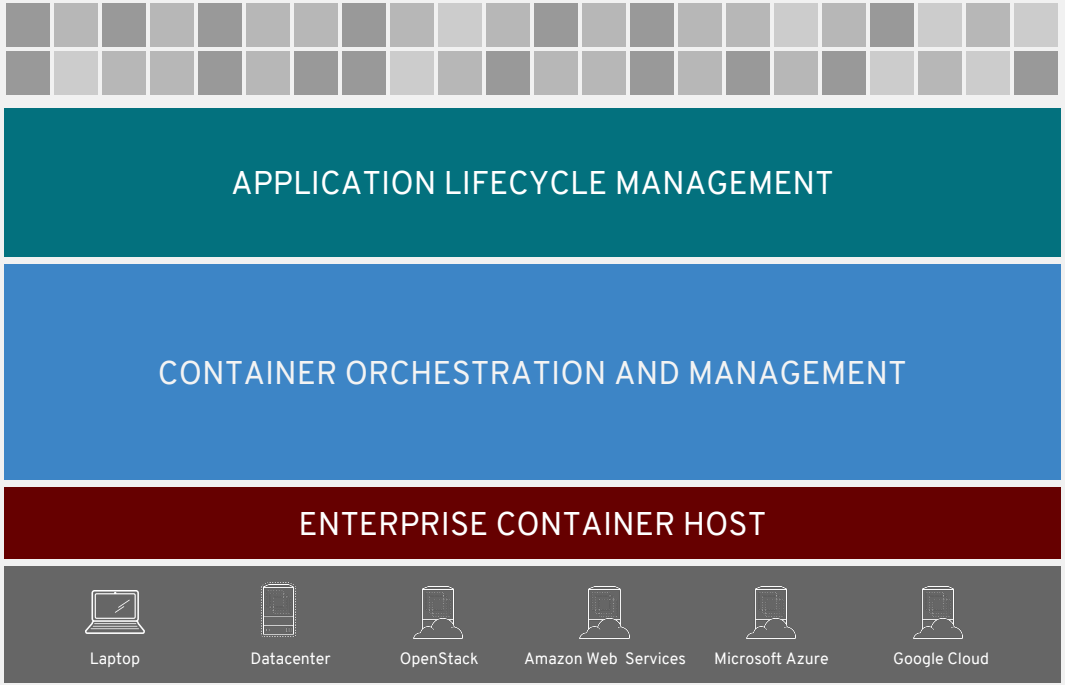
COMMUNITY INCUBATION

Communities
supporting innovation

A DEVELOPER INTRODUCTION TO OPENSHIFT

A secure and enterprise-grade container application platform based on Kubernetes for traditional and cloud-native applications

CLOUD-NATIVE CAPABILITIES WITH RED HAT OPENSHIFT

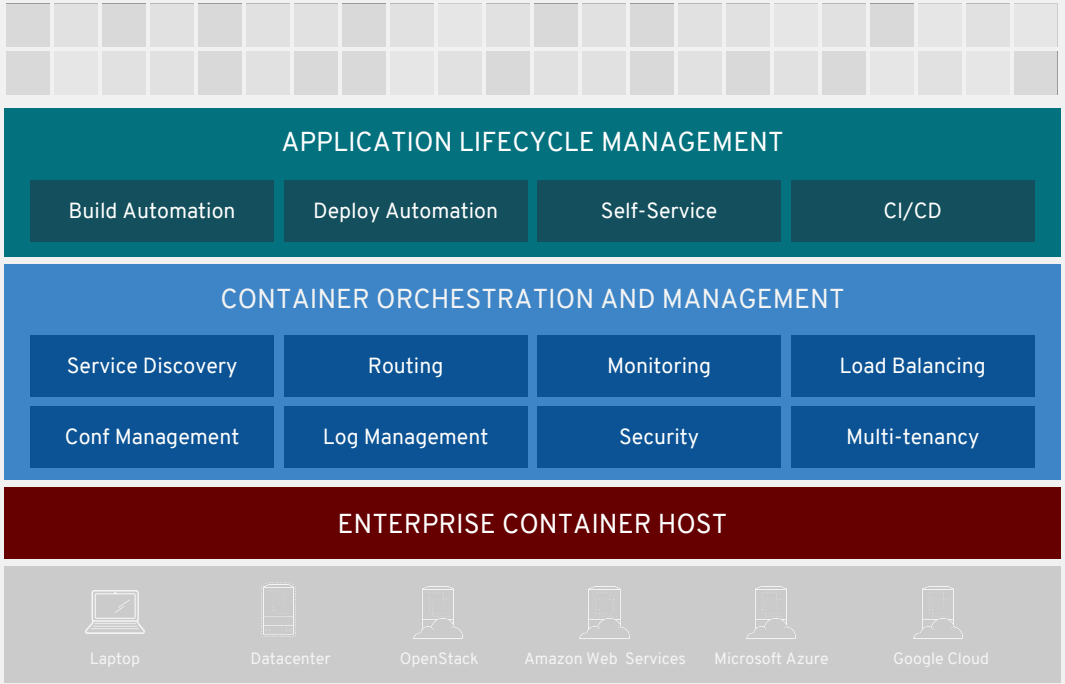


ANY CONTAINER

RED HAT® OPENSHIFT

ANY INFRASTRUCTURE

CLOUD-NATIVE CAPABILITIES WITH RED HAT OPENSHIFT



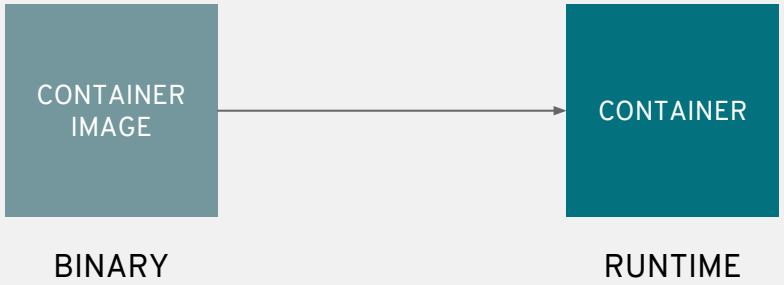
ANY CONTAINER

RED HAT® OPENSHIFT

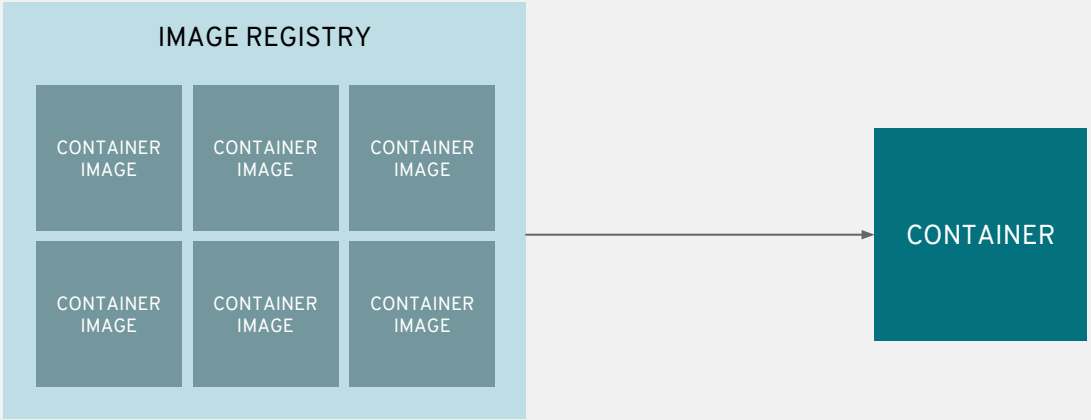
ANY INFRASTRUCTURE

A container is the smallest compute unit

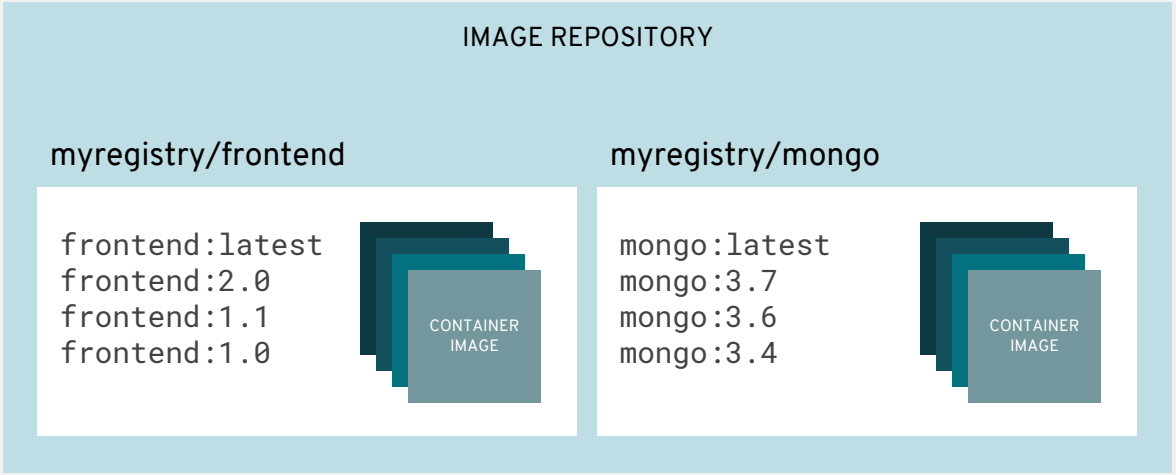
containers are created from container images during a build



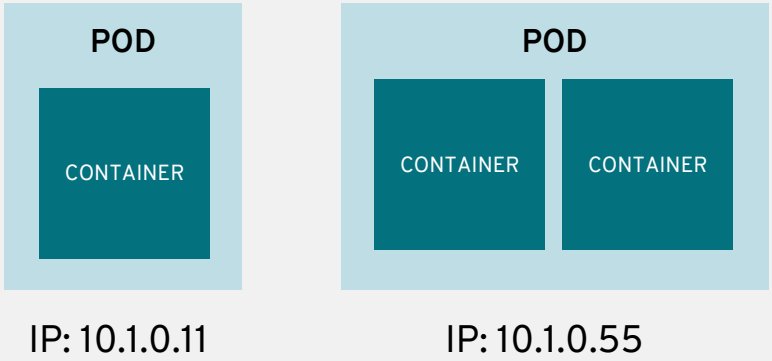
container images are stored in an image registry



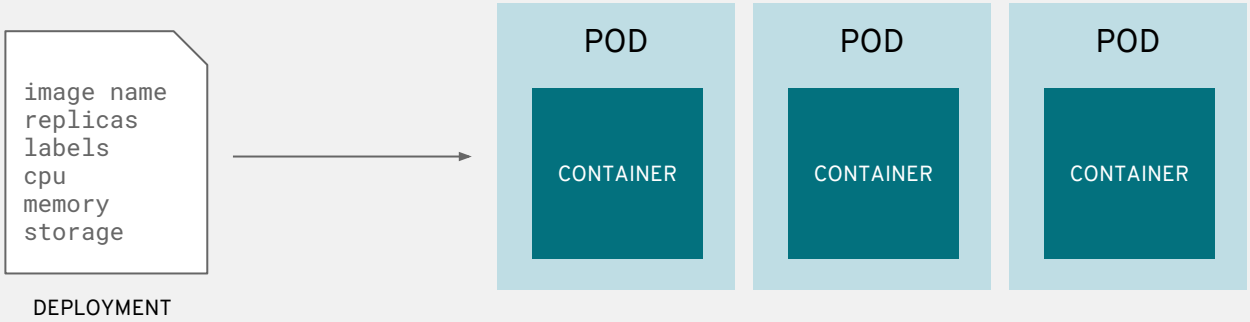
an image repository contains all versions of an image in the image registry



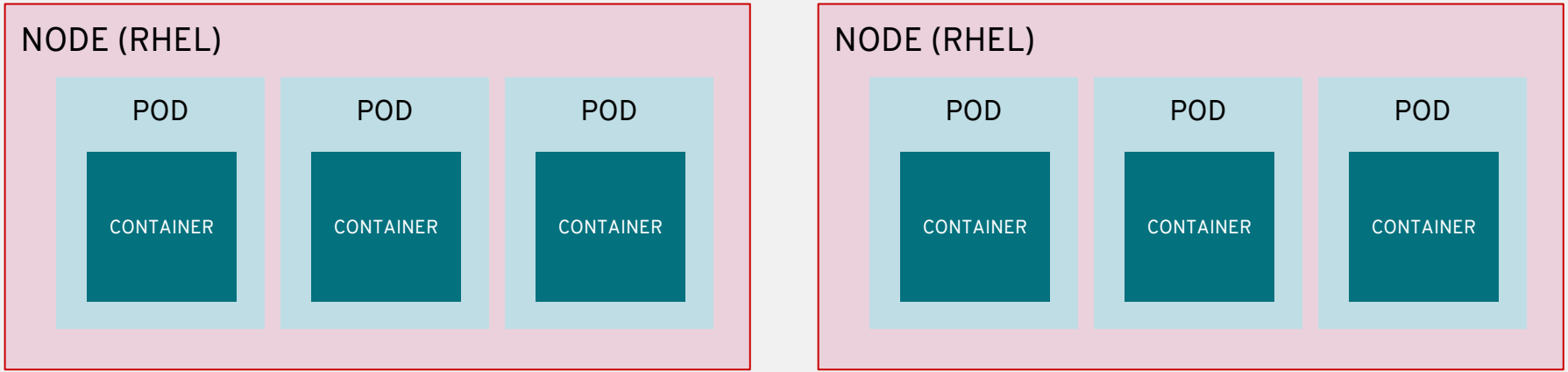
containers are wrapped in pods which are units of deployment and management, and share a common network address



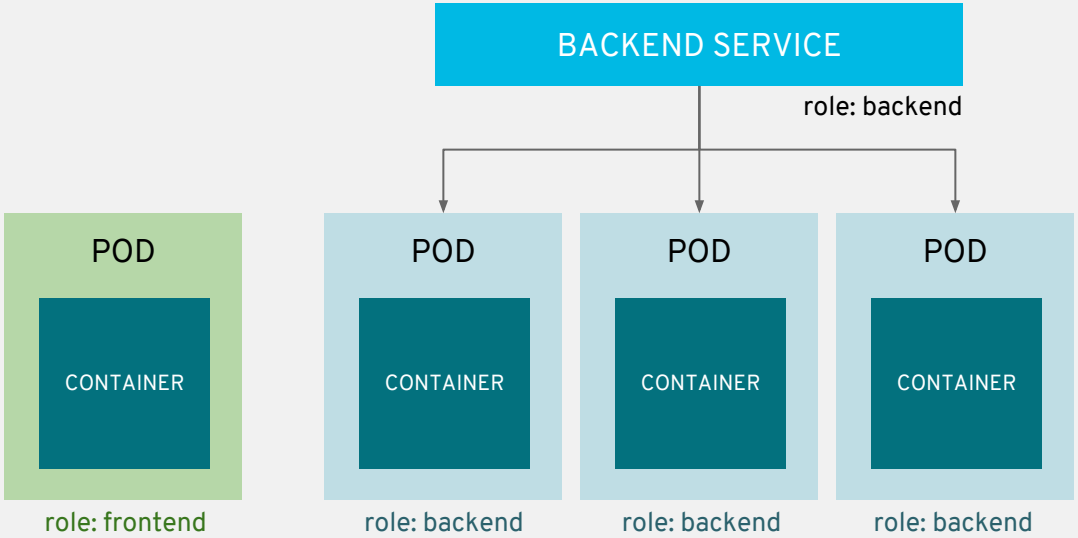
pods configuration is defined in a `deployment`



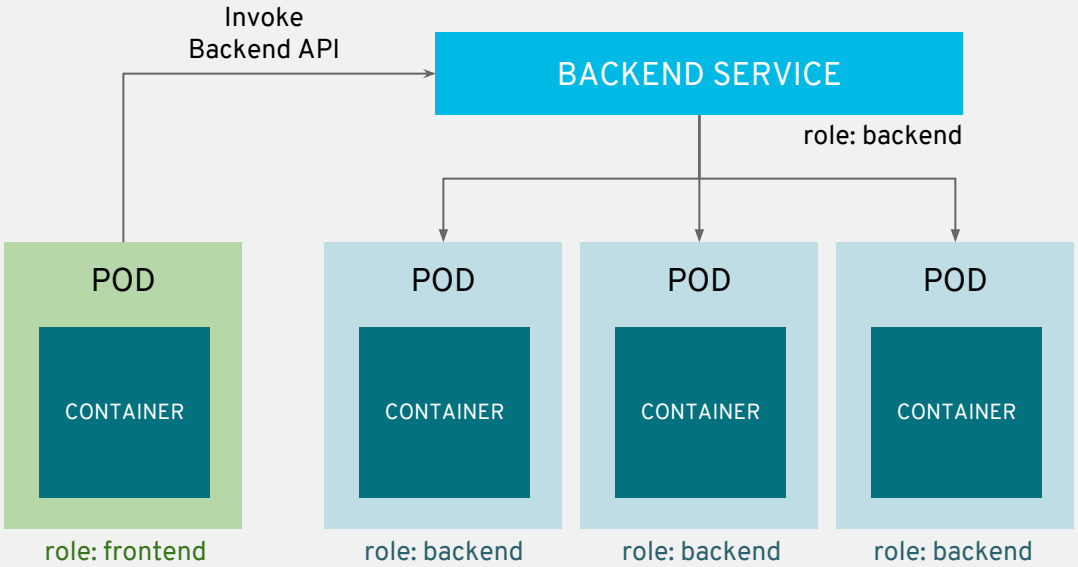
pods are deployed to and run on nodes



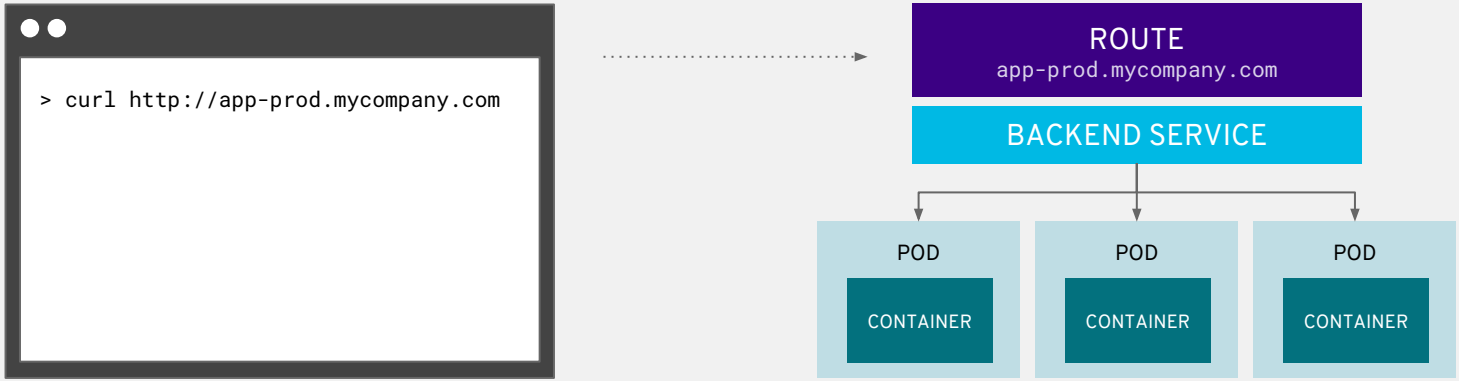
services provide internal load-balancing and service discovery across pods



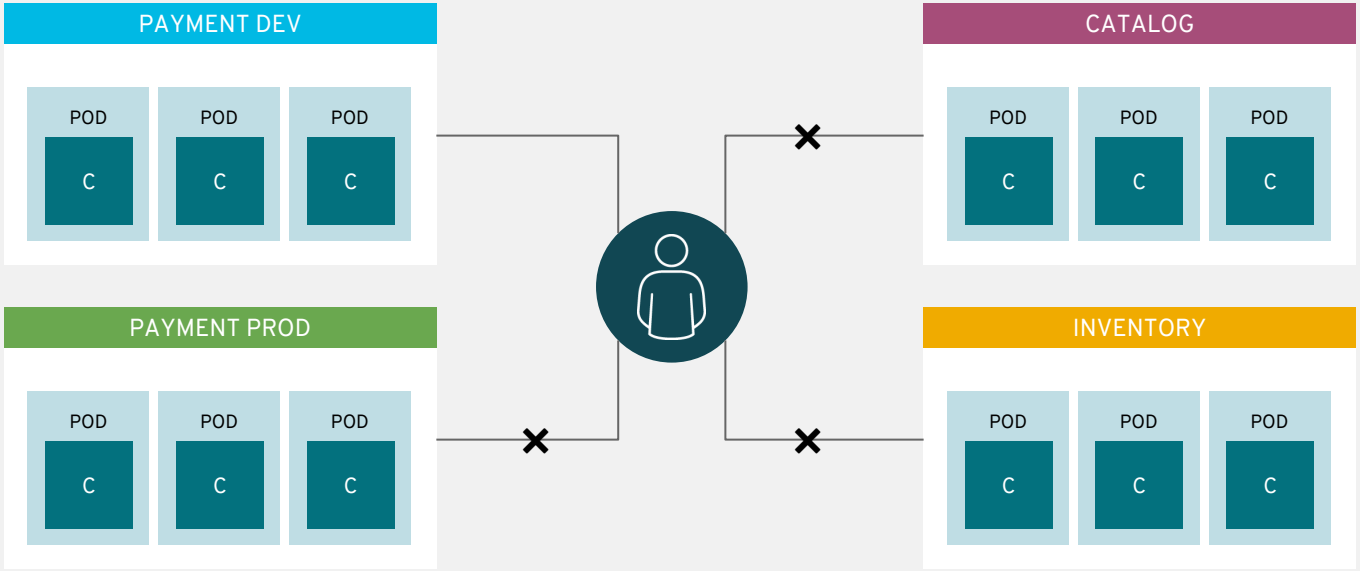
apps can talk to each other via services



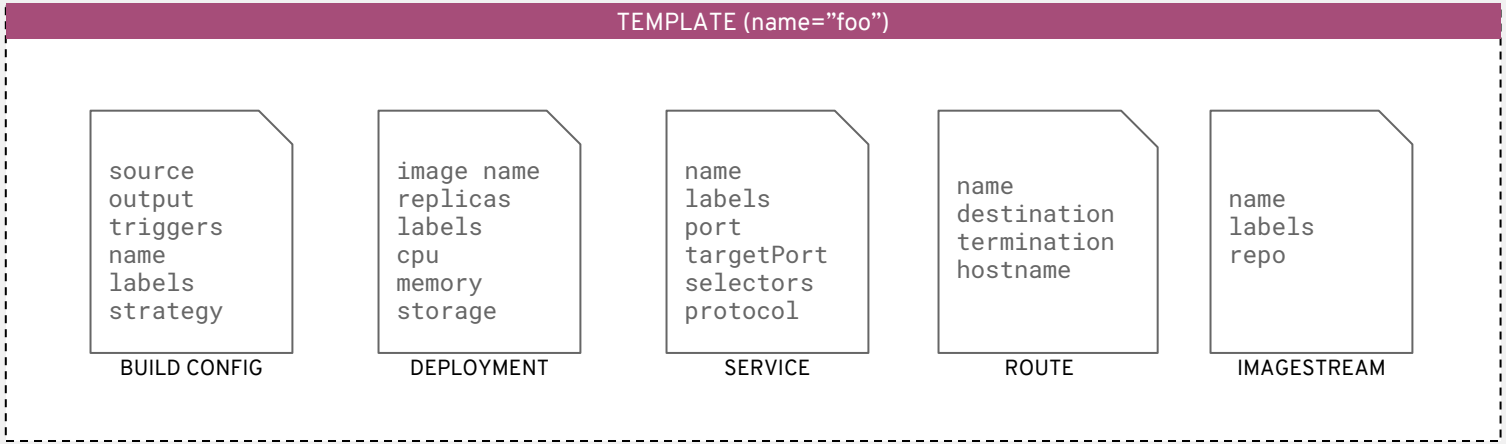
routes add services to the external load-balancer and provide readable urls for the app



projects isolate apps across environments, teams, groups and departments



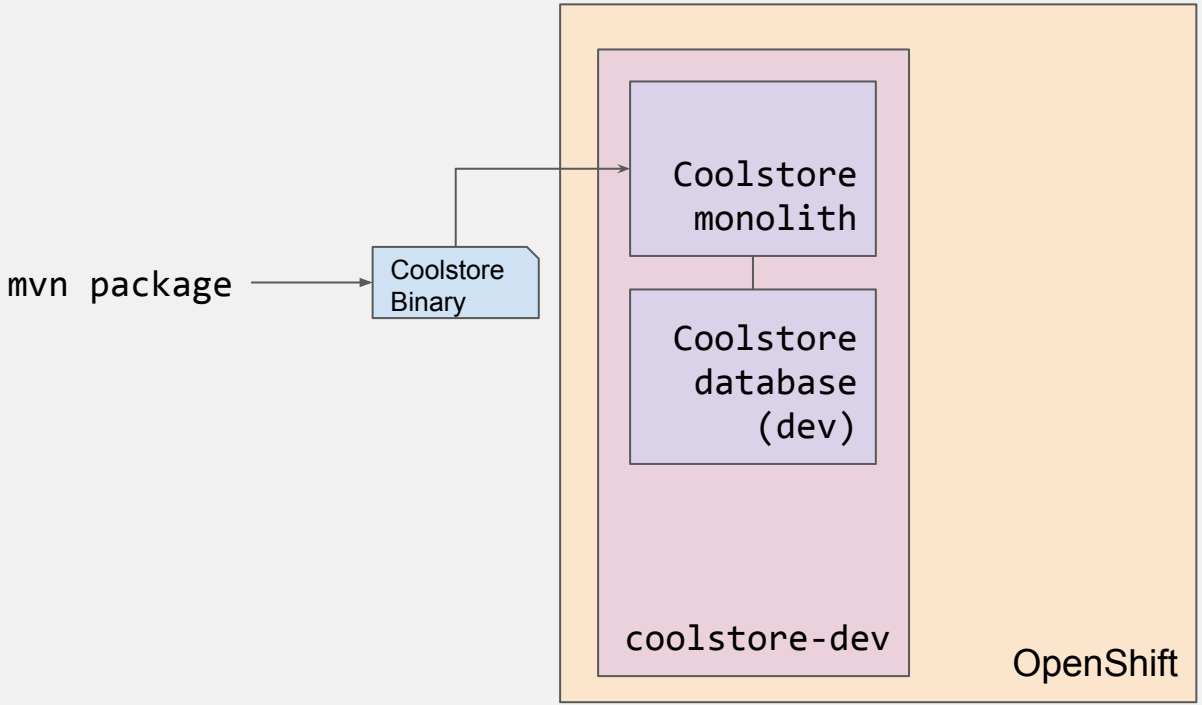
templates define a blueprint for an application that can be instantiated within a project




```
$ oc new-app foo
```

LAB: DEVELOPER INTRODUCTION TO OPENSHIFT

CURRENT STATE



GOAL FOR LAB

In this lab you will learn:

- Important OpenShift concepts for developers
- How OpenShift makes developers and architects happier
- How to do efficient round-trip development:
  - Separate **dev** from **prod** environments
  - Quick deployments using **rsync** / port-forwarding
  - Promoting apps using **CI/CD Pipelines**

LAB: DEVELOPER INTRO TO OPENSHIFT

WEB: openshift-modernize-apps.katacoda.com

SLIDES (PDF): bit.ly/m2m-slides

SCENARIO 3 A DEVELOPER INTRODUCTION TO OPENSHIFT

WRAP-UP AND DISCUSSION

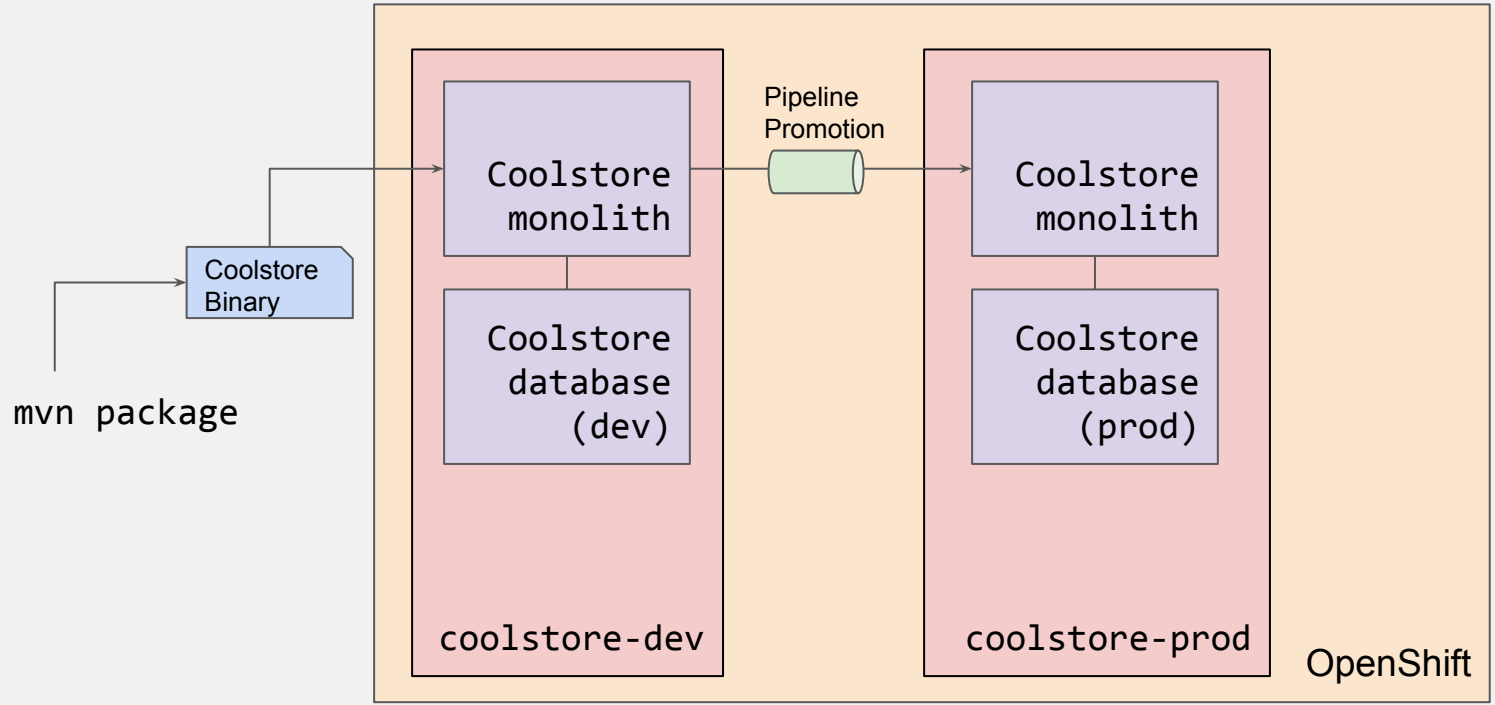
RESULT OF LAB

In this lab you learned how to:

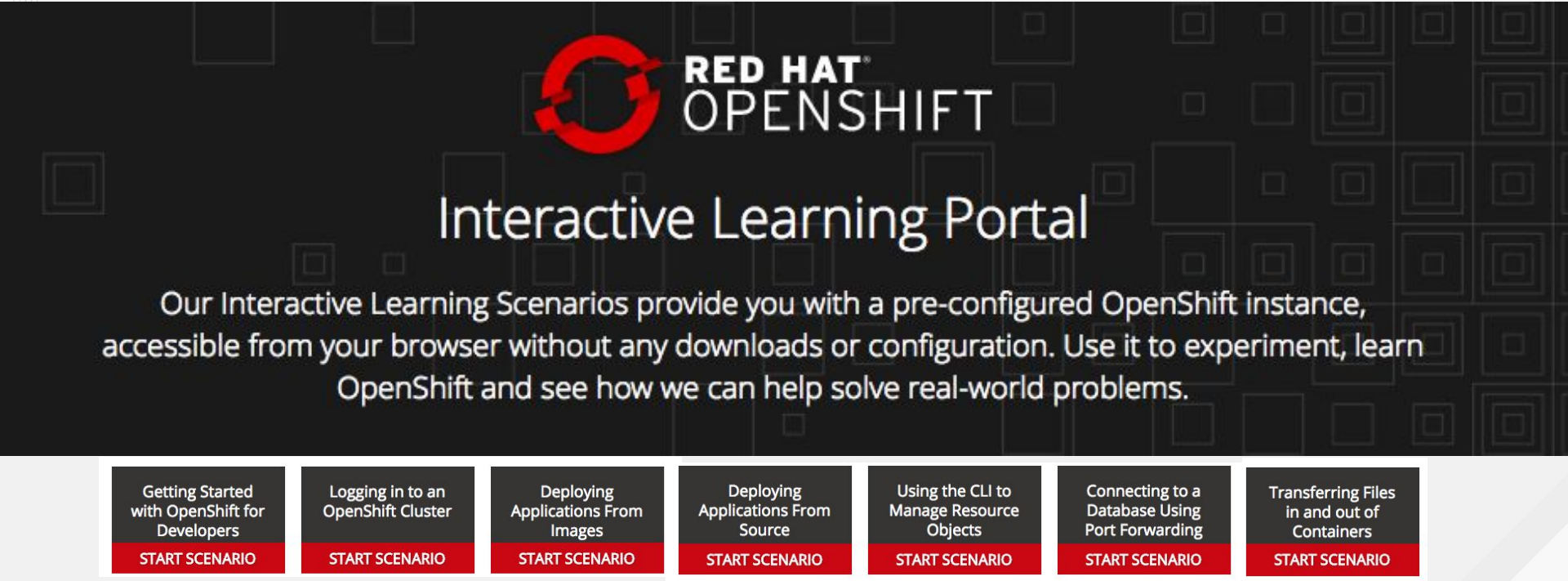
- Do quick deployments with **`oc rsync`**
- Create a production environment separate from dev
- Promote tested/verified builds between environments using OpenShift pipeline builds

You should now have two projects (dev and prod) running the same CoolStore app! In the next lab we will begin the process of breaking the monolith up into microservices.

DESIRED RESULT OF SCENARIO 3



LEARN MORE: learn.openshift.com



PART 3: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT

WHY MONOLITH TO MICROSERVICES

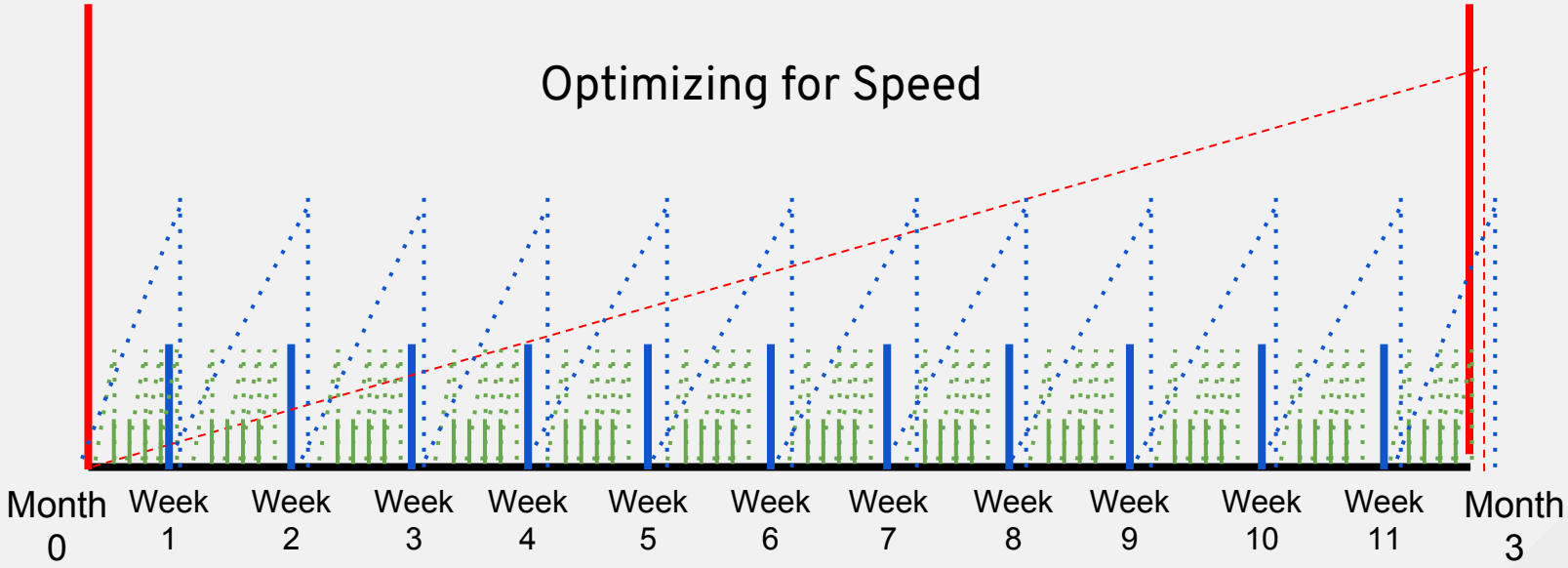
Break things down (organizations, teams, IT systems, etc) down into smaller pieces for greater parallelization and autonomy and focus on reducing time to value.

REDUCING TIME TO VALUE

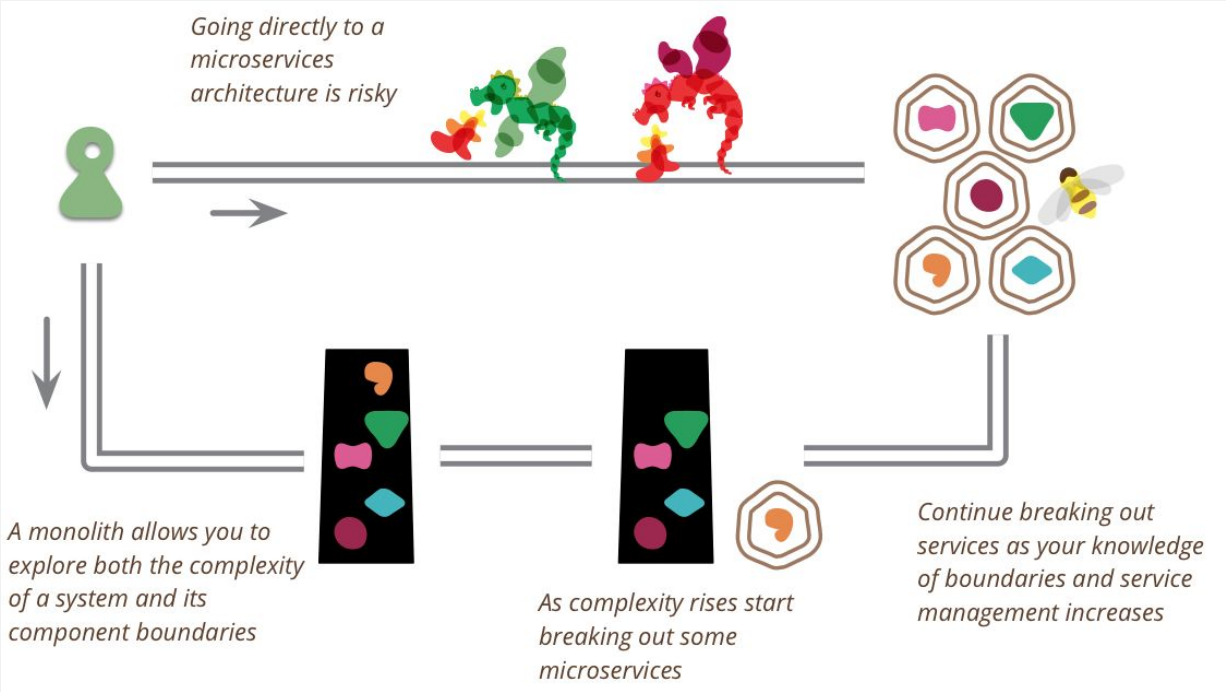
Monolith Lifecycle

Fast Moving Monolith

Microservices



Monolith First?



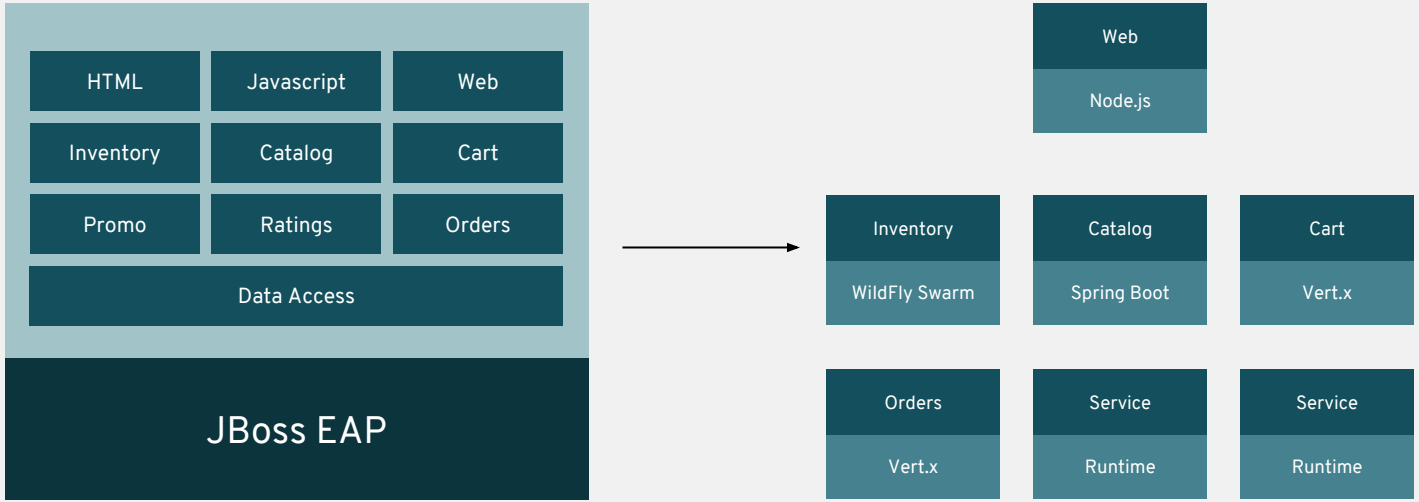
http://martinfowler.com/bliki/MonolithFirst.html

THE BIGGER PICTURE: THE PATH TO CLOUD-NATIVE APPS

A DIGITAL DARWINISM

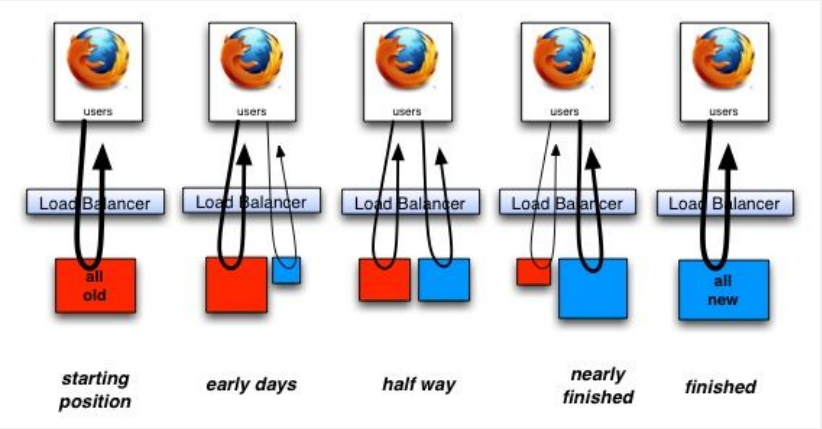
STRANGLING THE MONOLITH

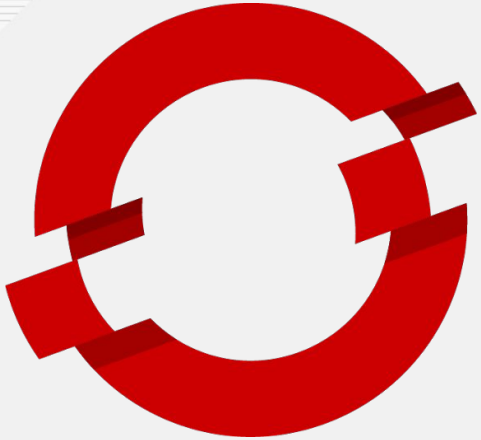
- In this lab, you will begin to ‘strangle’ the coolstore monolith by implementing its services as external microservices, split along business boundaries
- Once implemented, traffic destined to the original monolith’s services will be redirected (via OpenShift software-defined routing) to the new services



STRANGLING THE MONOLITH

- Strangling - **incrementally** replacing functionality in app with something better (cheaper, faster, easier to maintain).
- As functionality is replaced, “dead” parts of monolith can be removed/retired.
- You can also wait for all functionality to be replaced before retiring anything!
- Business stakeholders find this attractive as new functionality is added more rapidly during strangulation.

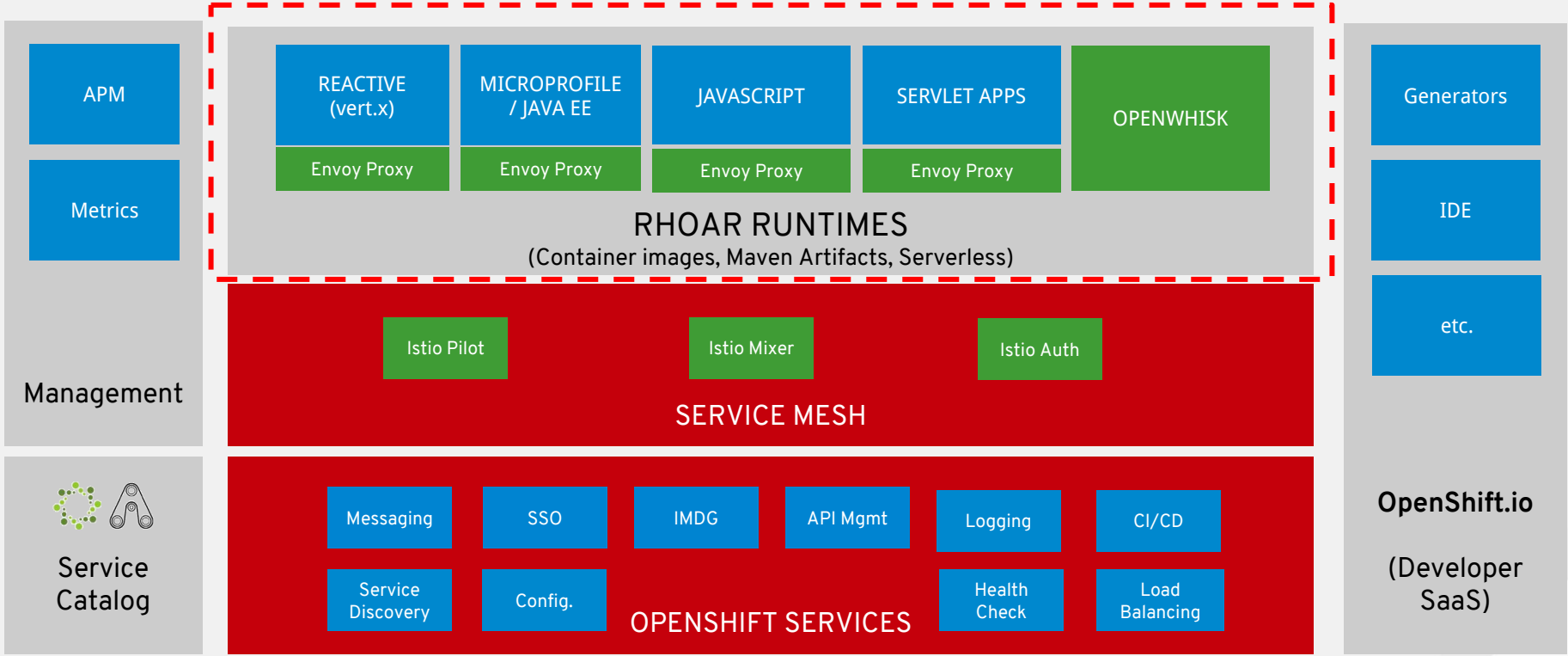




RED HAT® OPENSHIFT

Application Runtimes

RHOAR PRODUCT ARCHITECTURE



== Future

ENTERPRISE JAVA

RED HAT JBOSS
ENTERPRISE
APPLICATION PLATFORM

SERVLET APPS

JAVA MICROSERVICES

JAVASCRIPT FLEXIBILITY

REACTIVE SYSTEMS

TOMCAT SIMPLICITY

RED HAT JBOSS
WEB SERVER

SPRING

redhat.

- Microservices for Developers using Spring Framework
- An opinionated approach to building Spring applications
- Historical alternative to Java EE
- Getting started experience
- Spring MVC / DI / Boot most popular

Spring in RHOAR

- **It's the same Spring you know and love**
- Tested and Verified by Red Hat QE
  - Spring Boot, Spring Cloud Kubernetes, Ribbon, Hystrix
- Red Hat components fully supported
  - Tomcat, Hibernate, CXF, SSO (Keycloak), Messaging (AMQ), …
- Native Kubernetes/OpenShift integration (Spring Cloud)
    - Service Discovery via k8s (DNS), Ribbon
    - Spring Config via ConfigMap
- Developer Tooling (launch.openshift.io, starters)
- Additional planned support for
  - Transactions (Naryana)

Cloud Native Support in Spring

- Health Checks (actuator)
- Externalized Config (spring-cloud-kubernetes)
- Client-side discovery / load balancing (Eureka/Kubernetes)
- Circuit Breaking / Bulkheading (Hystrix)
- Logging / Monitoring / Tracing / Metrics
- Secure deployments with Keycloak
- API Documentation (Swagger)

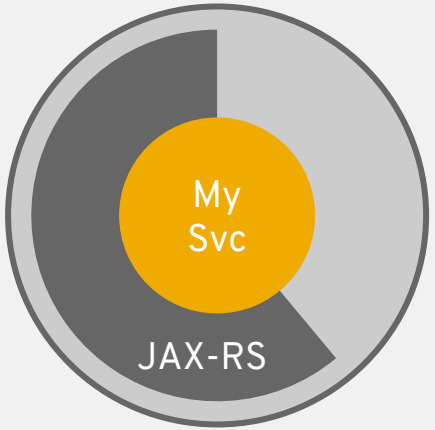
WILDFLY SWARM

redhat.

WildFly SWARM

Java EE microservices

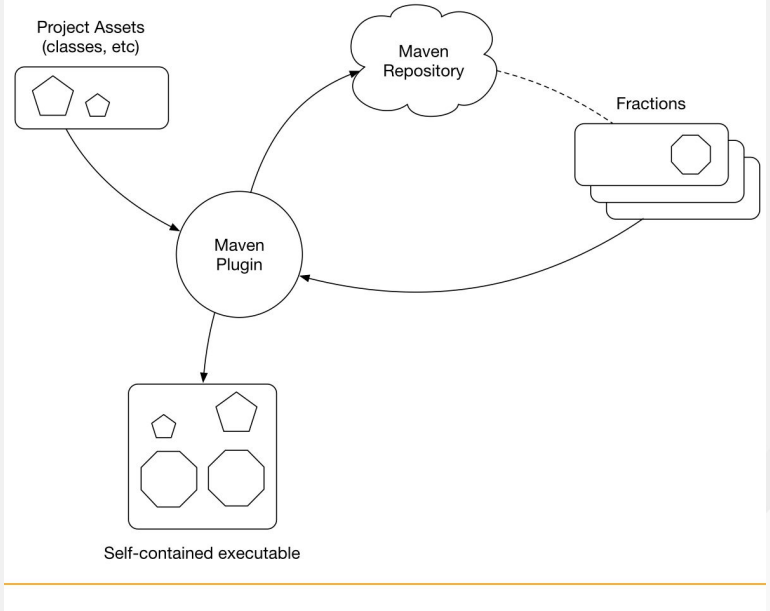
- Leverage Java EE expertise
- Open standard
- Microservices focus
- Optimized for OpenShift
- Super lightweight
- Tooling for Developers
- MicroProfile Implementation




```
$ java -jar my_microservice.jar
```

WildFly Swarm “pieces” - Fractions

- A tangible unit providing a specific piece of functionality
- Embodied in a maven artifact
- To support the compositional aspect in WF Swarm
- Provides the “runtime” capabilities
- Means to add API dependencies (e.g. JAX-RS)
- Means to configure the system
  - With reasonable defaults
- Means to discover other components (topology)
- Means to alter deployments (e.g. keycloak)
- Can be auto-detected or explicitly declared



Cloud Native Support in WildFly Swarm

- Health Checks
- Externalized Config
- Client-side discovery / load balancing
- Circuit Breaking / Bulkheading
- Logging / Monitoring / Tracing / Metrics
- Secure deployments with Keycloak
- MicroProfile
- API Documentation

WildFly Swarm and RHOAR

WildFly SWARM

Build microsevices

- Embeddable (Fat Jar)
- Lightweight
- Modular & extensible
- Built from WildFly (Trusted and Reliable)

MICROPROFILE™

RED HAT® OPENSHIFT Application Runtimes

Upstream (Unsupported)

Flyway	JMS	Jolokia
Logstash	Vert.x Integration	Infinispan
Fluentd	Consul	jGroups
Swagger	Spring	

Tested and Verified

Hystrix | Ribbon | MySQL | Oracle DB

Additional Supported Fractions

Metrics	Health
Configuration	Monitor
Keycloak	Topology

Supported Specifications

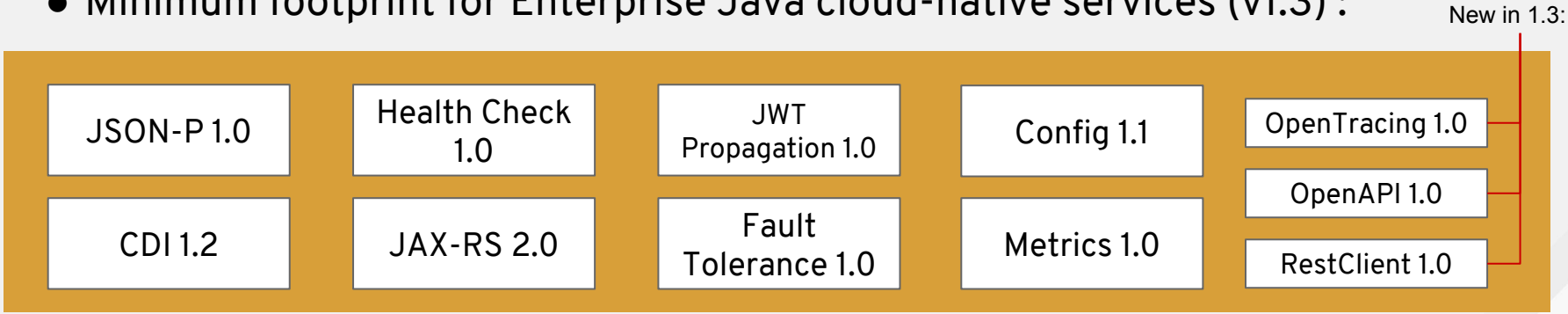
Java EE 7 Web Profile | MicroProfile 1.0

MICROPROFILE™

OPTIMIZING ENTERPRISE JAVA

eclipse

- Defines **open source** Java **microservices** specifications
- Industry Collaboration - Red Hat, IBM, Payara, Tomitribe, London Java Community, SouJava, Oracle, Hazelcast, Fujitsu, SmartBear...
- **WildFly Swarm** is **Red Hat's** implementation
- Minimum footprint for Enterprise Java cloud-native services (v1.3) :



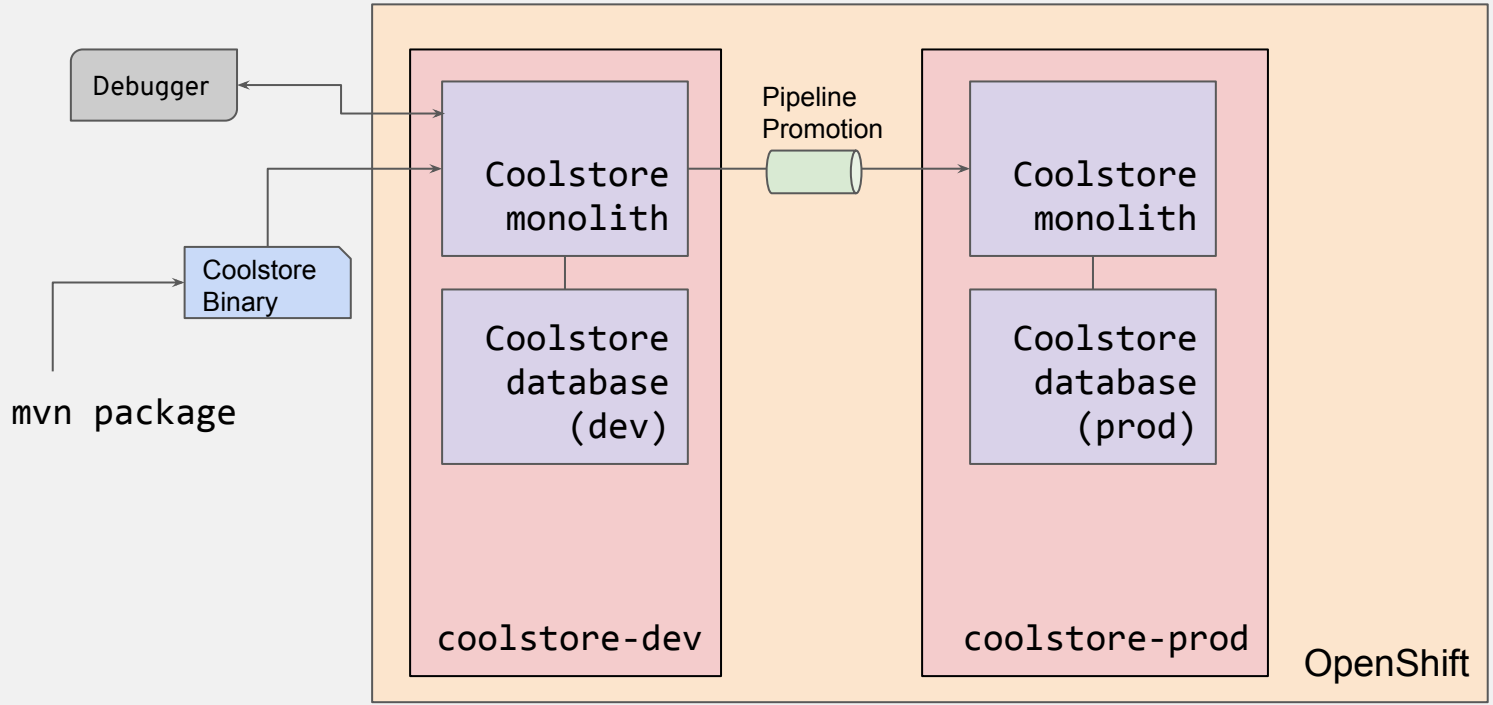
LAB: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT

GOAL FOR LAB

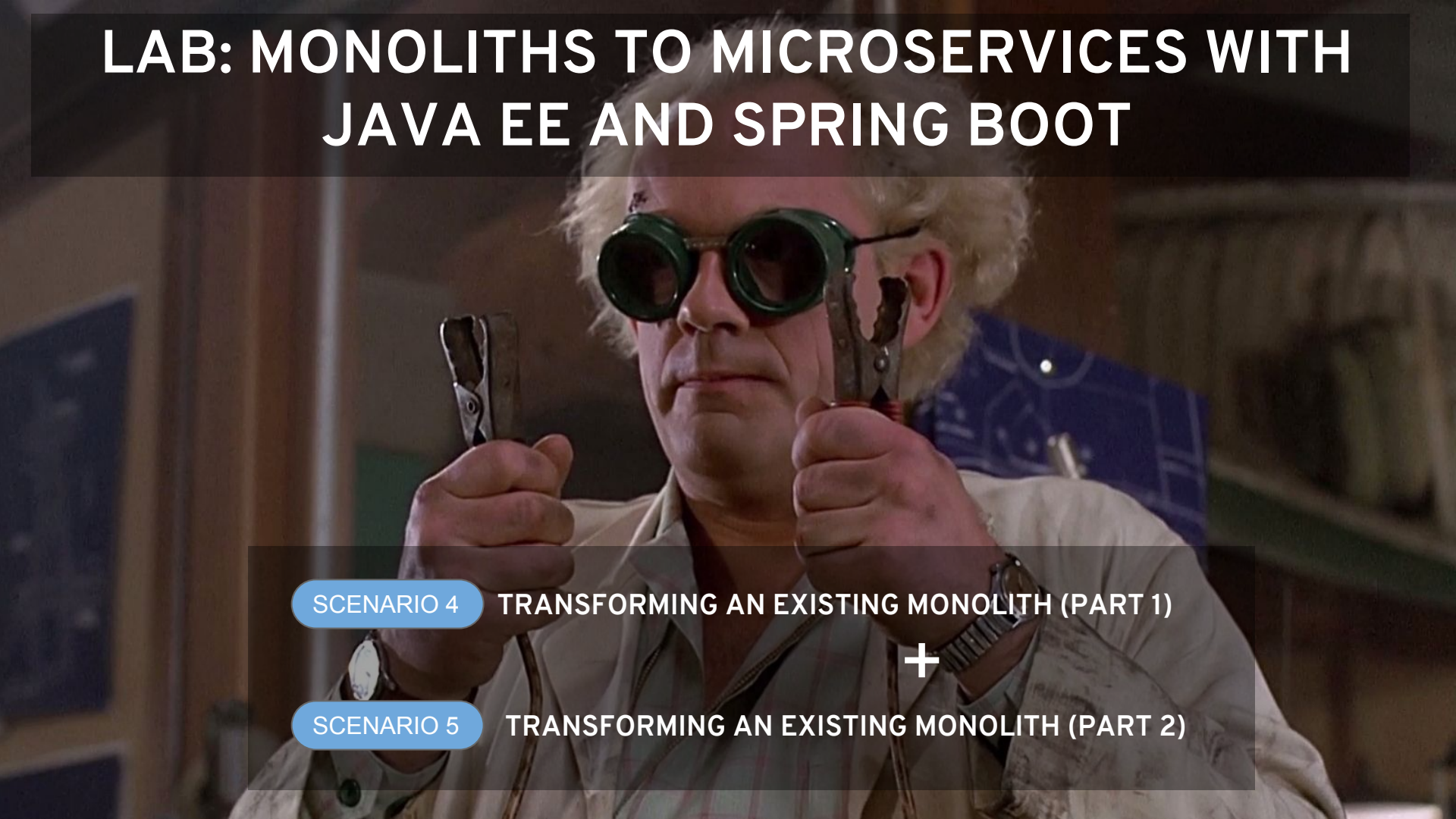
In this lab you will learn:

- How Red Hat OpenShift and Red Hat OpenShift Application Runtimes (RHOAR) help jumpstart app modernization
- Benefits and challenges of microservices
- How to transform existing monolithic applications to microservices using strangler pattern and 12-factor app patterns.
- Use modern app dev frameworks like WildFly Swarm and Spring Boot to implement microservice applications on OpenShift

CURRENT STATE - THE MONOLITH



LAB: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT



SCENARIO 4 TRANSFORMING AN EXISTING MONOLITH (PART 1)

+

SCENARIO 5 TRANSFORMING AN EXISTING MONOLITH (PART 2)

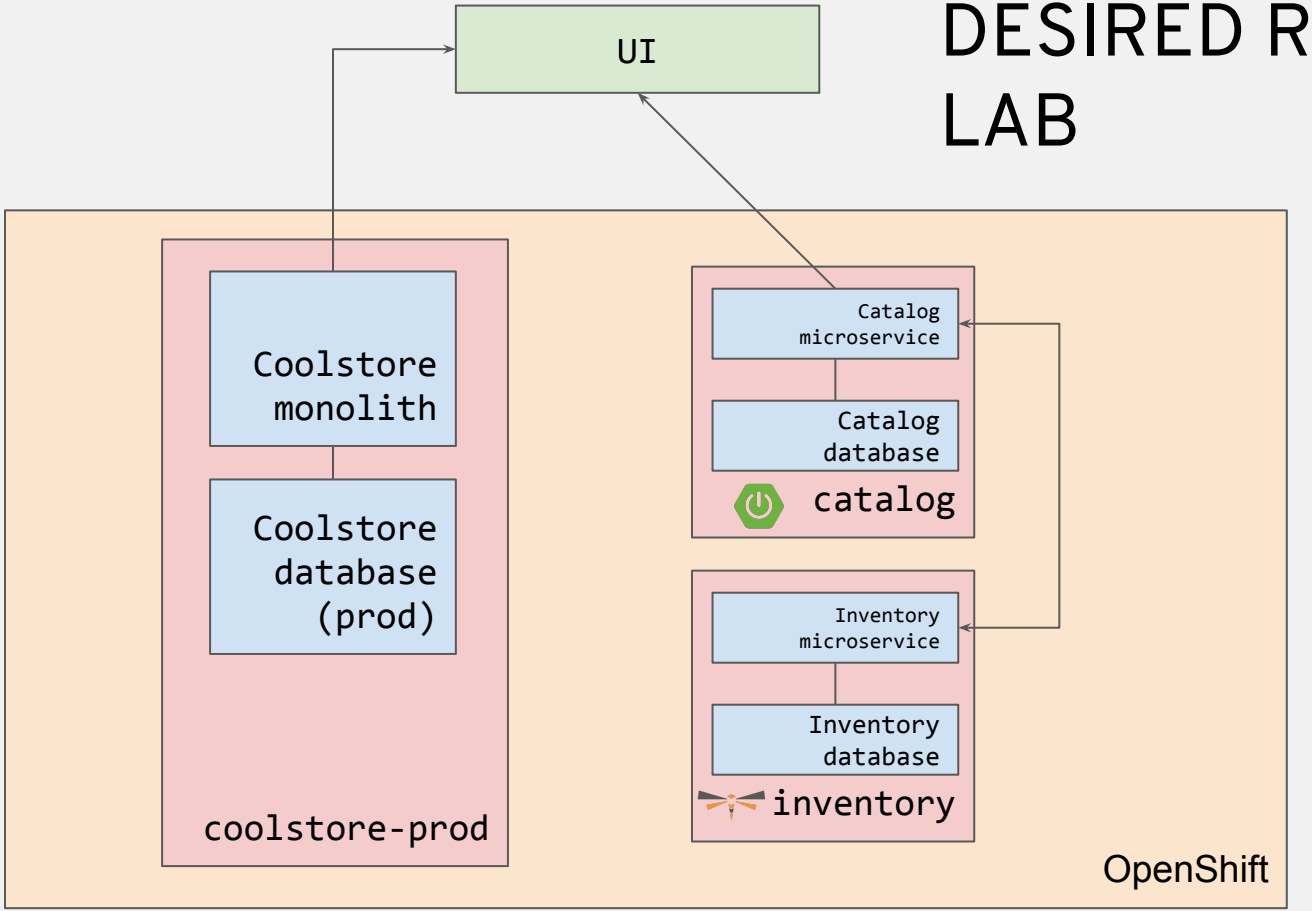
WRAP-UP AND DISCUSSION

RESULT OF LAB

In this lab you learned how to:

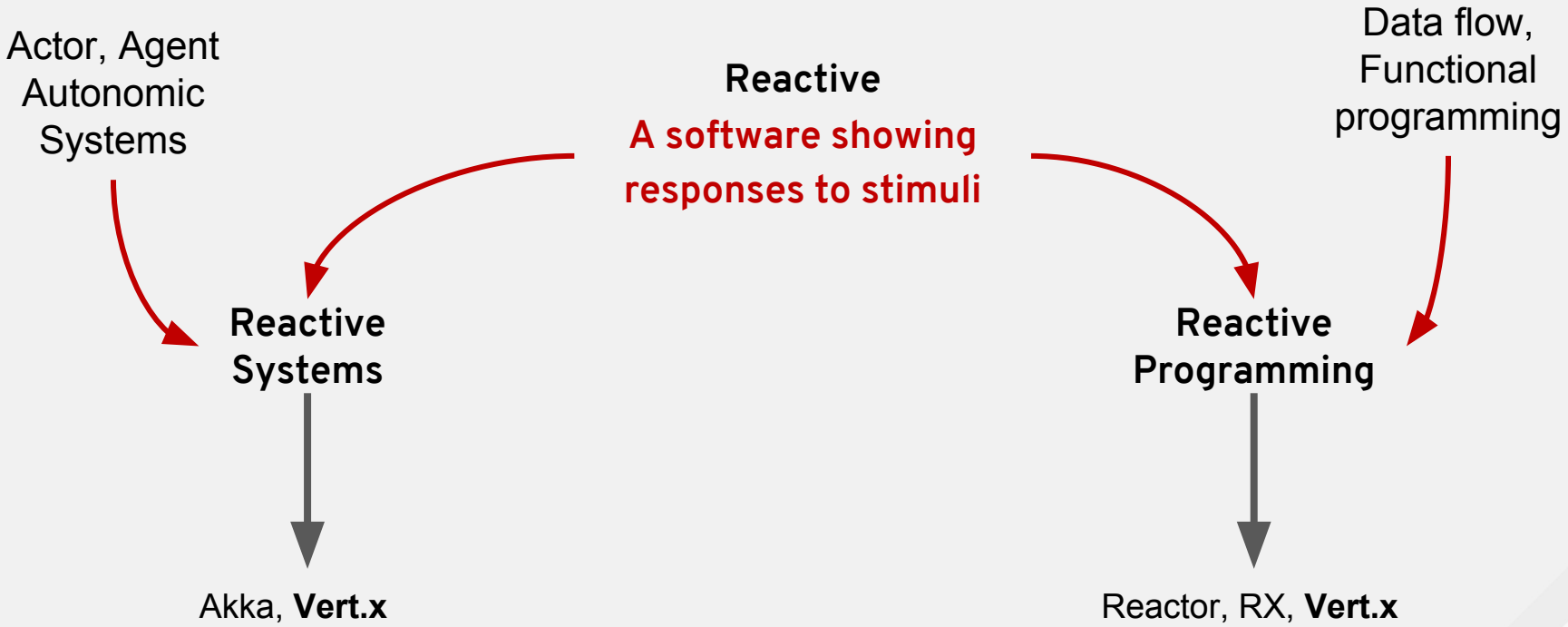
- Implement a Java EE microservice using WildFly Swarm
- Implement a Java EE microservice using Spring Boot
- Develop container-based testing
- Add microservice concerns like Health checks, externalized configuration and circuit breaking
- Use the strangler pattern to slowly migrate functionality from monolith to microservices

DESIRED RESULT OF LAB

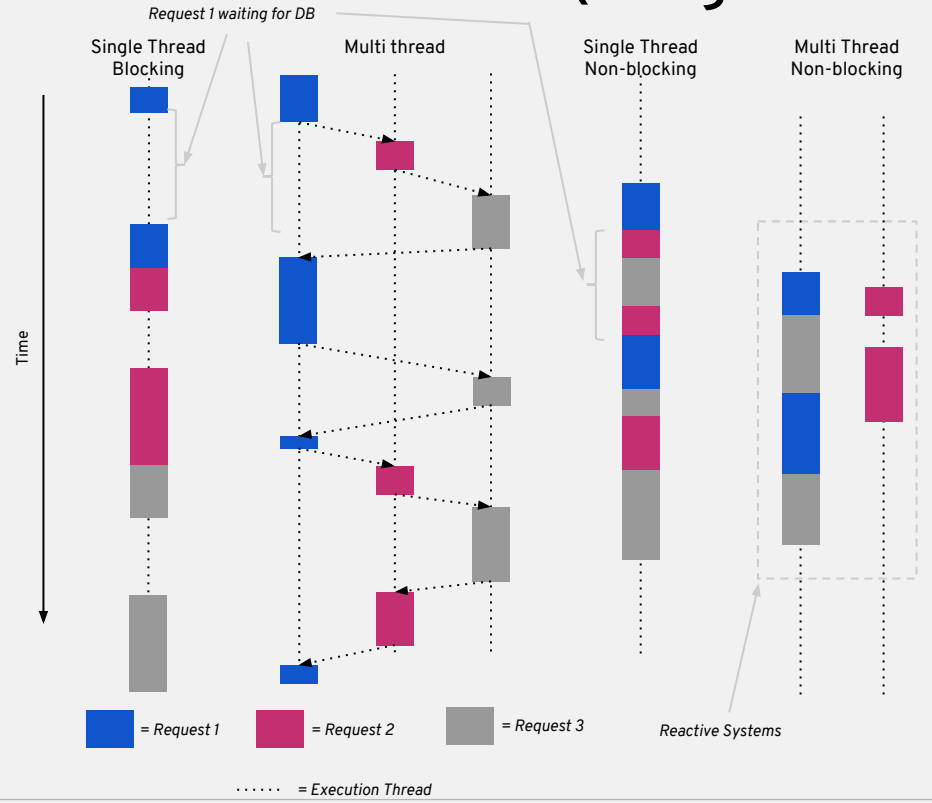


REACTIVE MICROSERVICES

The 2 faces of Reactive



Execution Model (single core)



Blocking

- Example: CGI, early versions of server side JavaScript.
- Can only scale horizontally

Multi thread

- Example: Java EE, Tomcat, Spring (non reactive)
- Scales horizontally and vertically

Non blocking

- Example: NodeJS, Eclipse Vert.x, Akka, Spring reactive
- Scales horizontally and vertically

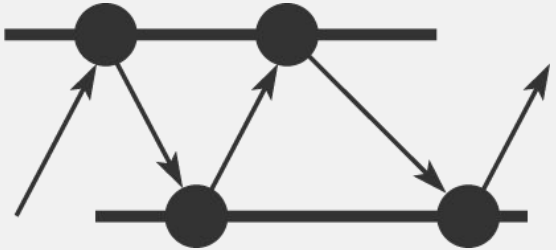
Eclipse Vert.x

VERT.X

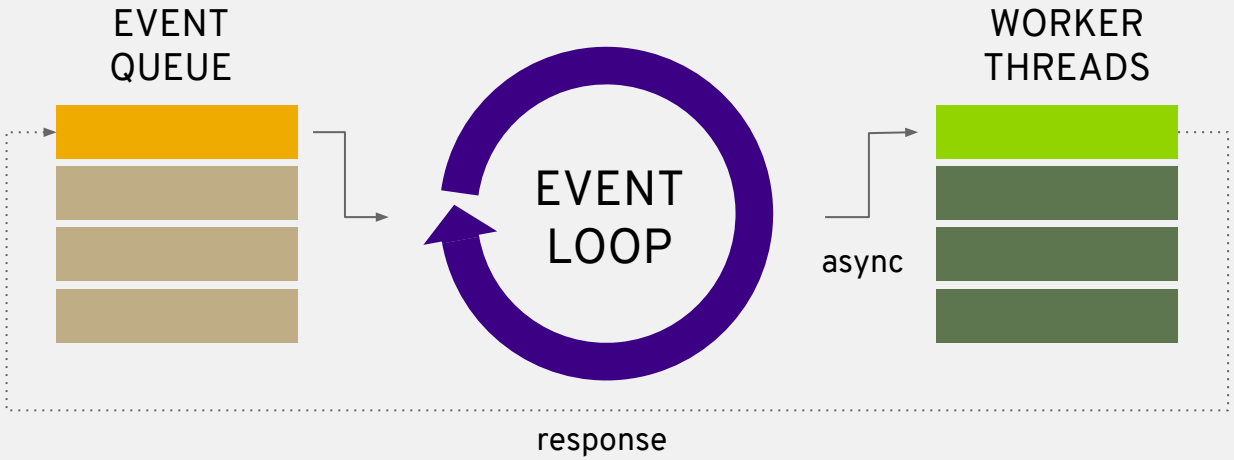
Vert.x is a toolkit to build distributed and reactive systems

- **Asynchronous Non-Blocking development model**
- Simplified concurrency (**event loop**)
- Reactive microservice, Web applications, IOT
- Ideal high-volume, low-latency applications
- Un-opinionated
- Understands clustering in its core architecture

Home - http://www.vertx.io

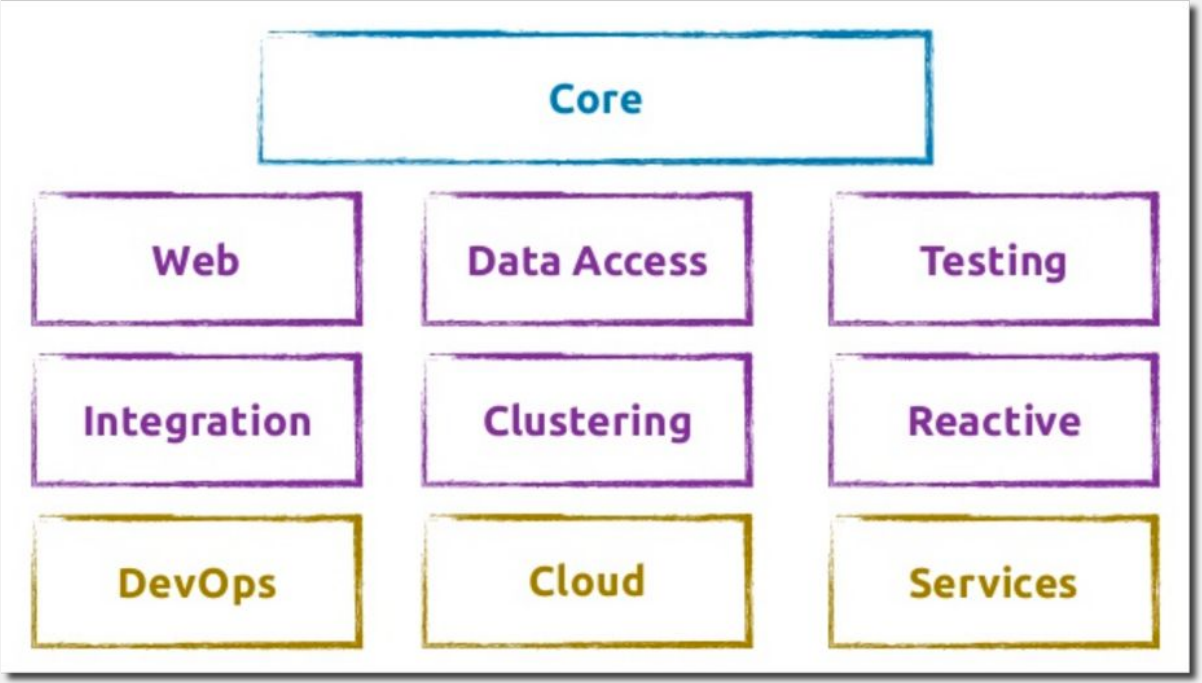


VERT.X EVENT LOOP



Handle Thousands of Requests
With Few Threads

Vert.x Ecosystem



LAB 4: Reactive Microservices with Eclipse Vert.x

- Explore Vert.x Maven project
- Create an API gateway
- Run Vert.x locally
- Deploy Vert.x on OpenShift

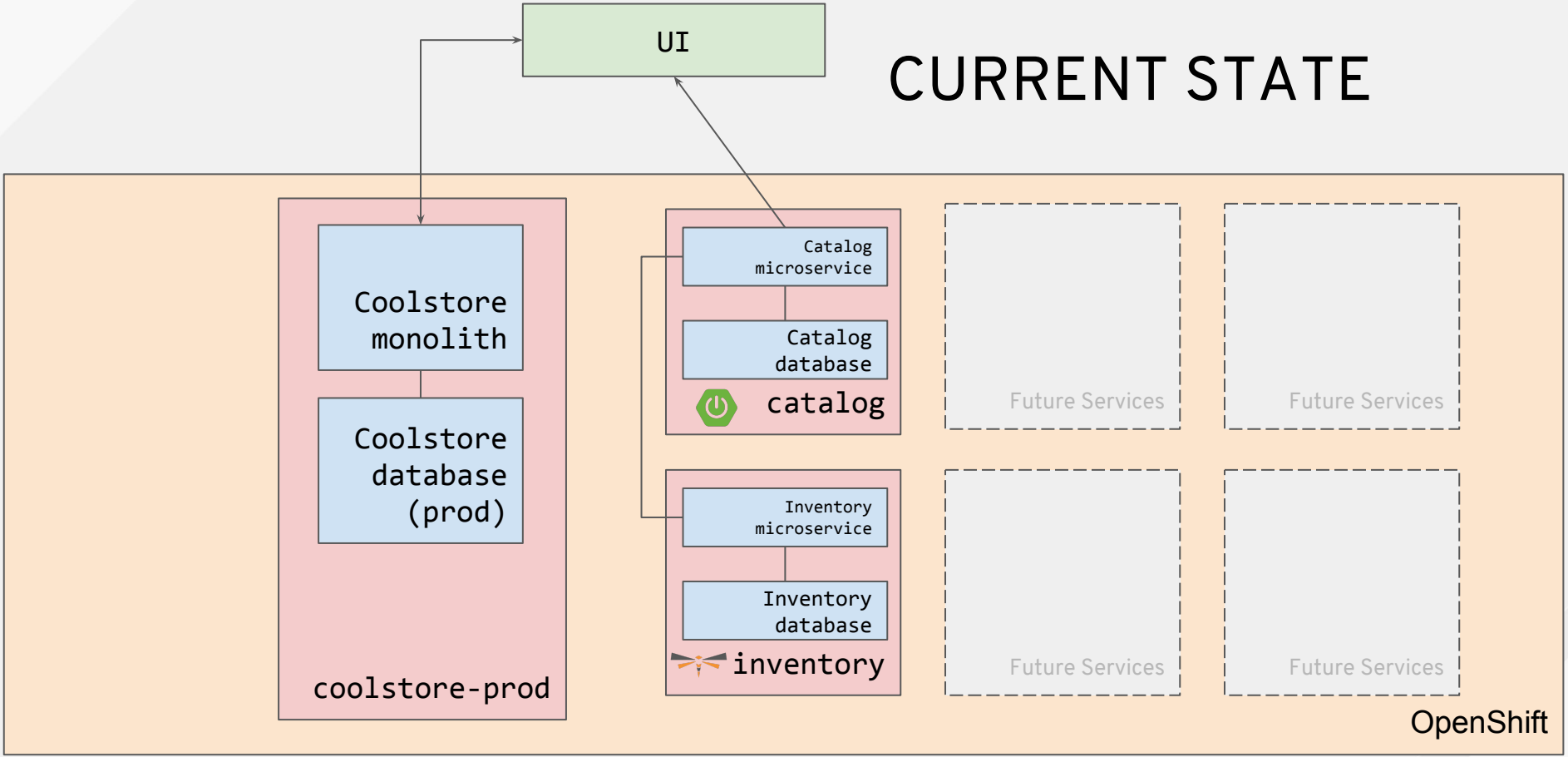
LAB: REACTIVE MICROSERVICES WITH ECLIPSE VERT.X

GOAL FOR LAB

In this lab you will learn:

- How Event-based architectures supercharge microservice apps
- Use cases for reactive applications
- Develop microservices using Eclipse Vert.x
- Interact with other microservices without blocking
- Learn the basics of Reactive programming

CURRENT STATE



LAB: REACTIVE MICROSERVICES

SCENARIO 6 BUILDING REACTIVE MICROSERVICES

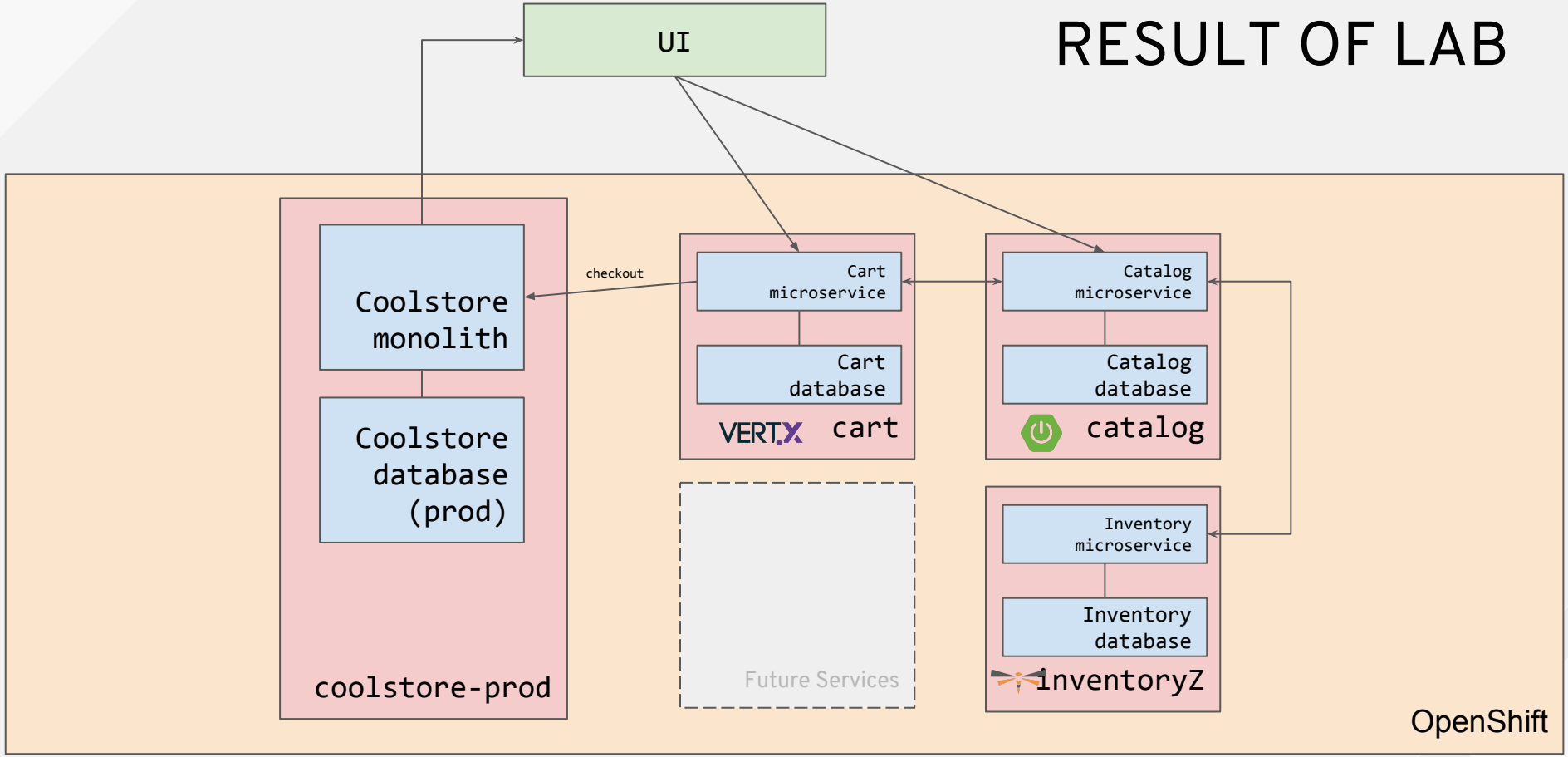
WRAP-UP AND DISCUSSION

RESULT OF LAB

In this lab you learned how to:

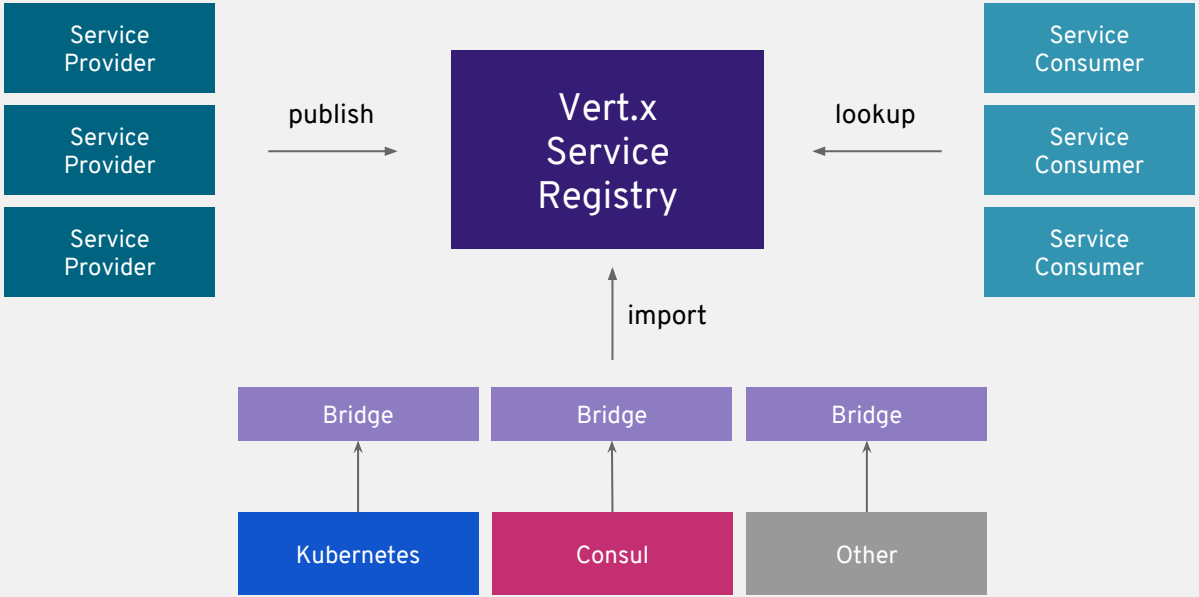
- Build reactive web application that are non-blocking
- Asynchronously call out to external service using Callbacks, Handlers and Futures
- Deploy the application to OpenShift

RESULT OF LAB



ECLIPSE VERT.X OFFER MUCH MORE

SERVICE DISCOVERY



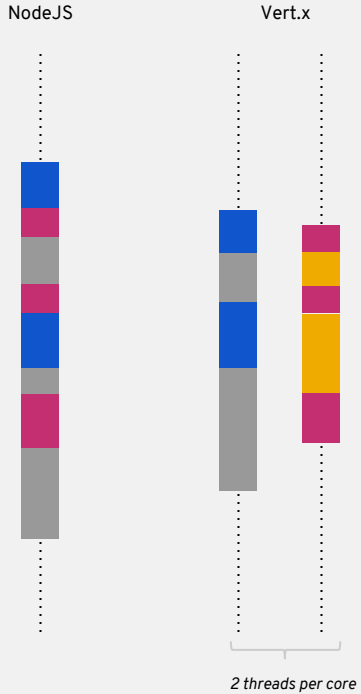
Vert.x vs NodeJS

Vert.x

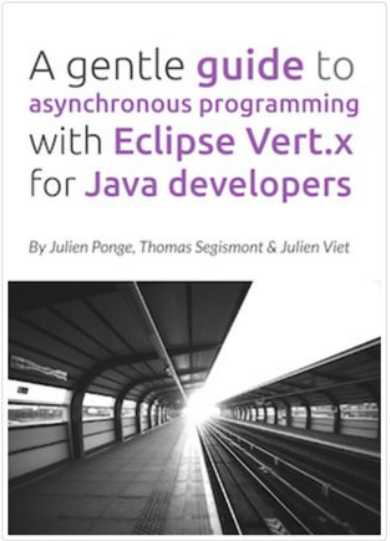
- Multi-threaded
- Polyglot (Java, JavaScript, Scala, and more)
- Supports reactive programming using RxJava, RxJS, etc

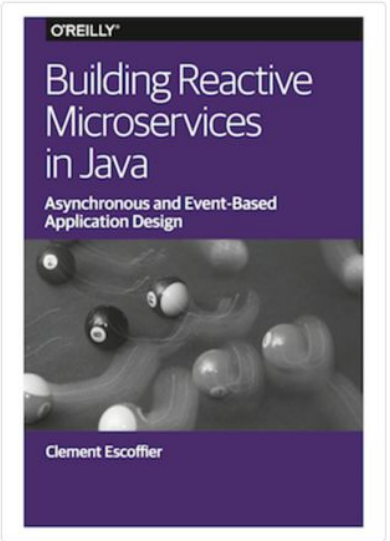
NodeJS

- Single threaded
- JavaScript only
- Support reactive programming using RxJS



FREE E-BOOKS





http://vertx.io/docs/

PART 5: RESILIENT DISTRIBUTED APPS

DISTRIBUTED SERVICES ARCHITECTURES

Benefits (when implemented correctly):

- Performance
- Reliability
- Resiliency
- Extensibility
- Availability
- Robustness

DISTRIBUTED SERVICES ARCHITECTURES

Fallacies of Distributed Computing

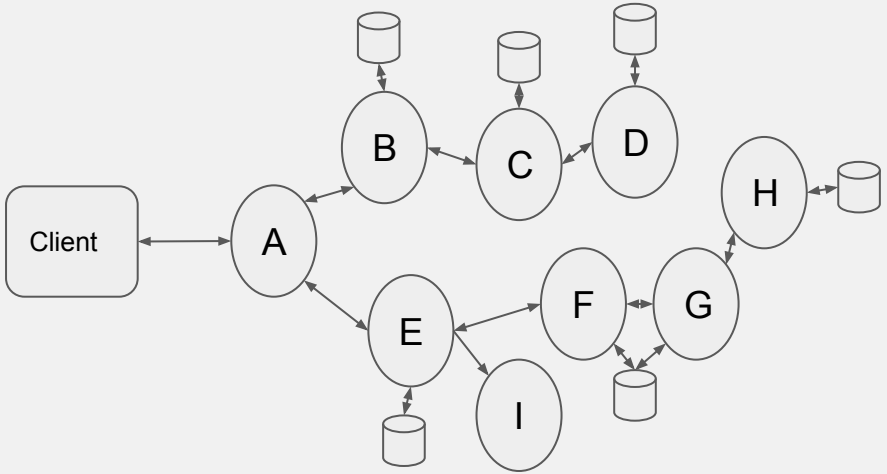
- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

wikipedia.org/wiki/Fallacies_of_distributed_computing

DISTRIBUTED SERVICES ARCHITECTURES

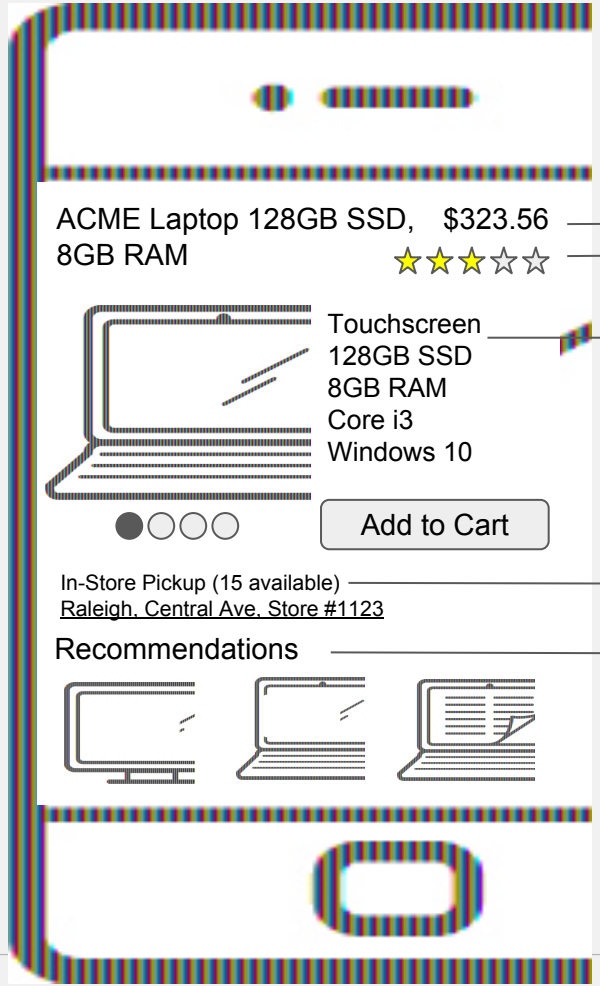
Applications must deal with

- Unpredictable failure modes
- End-to-end application correctness
- System degradation
- Topology changes
- Elastic/ephemeral/transient resources

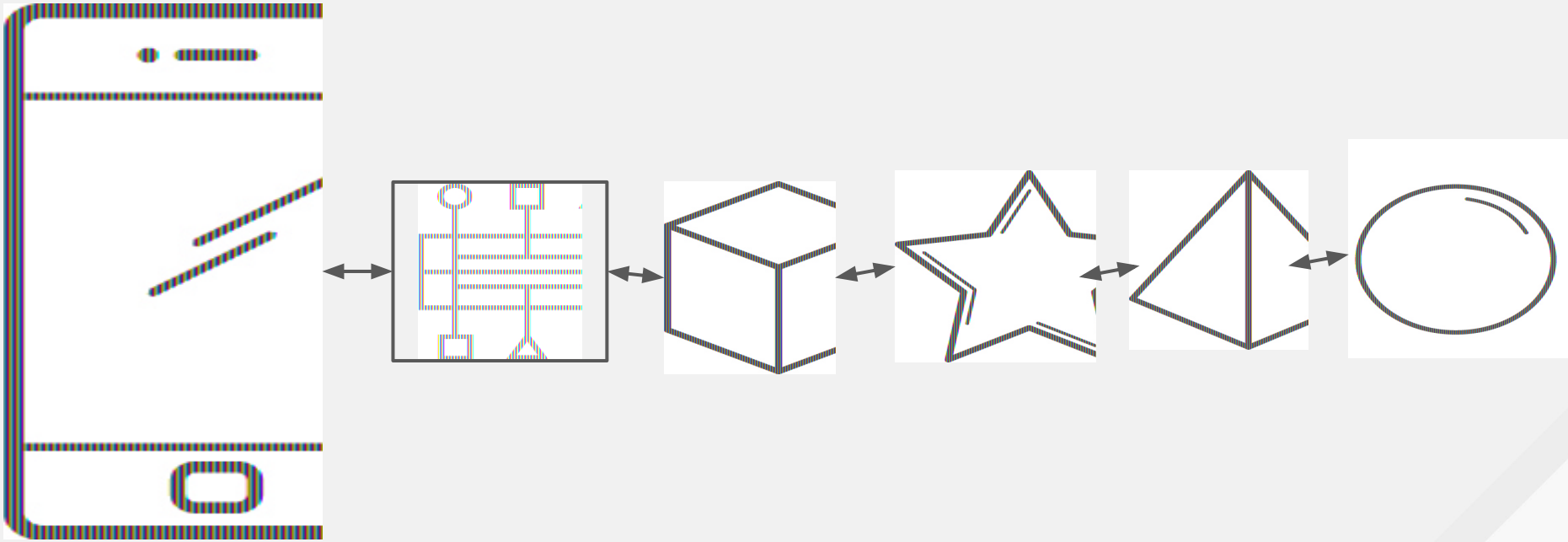


MICROSERVICES == DISTRIBUTED COMPUTING

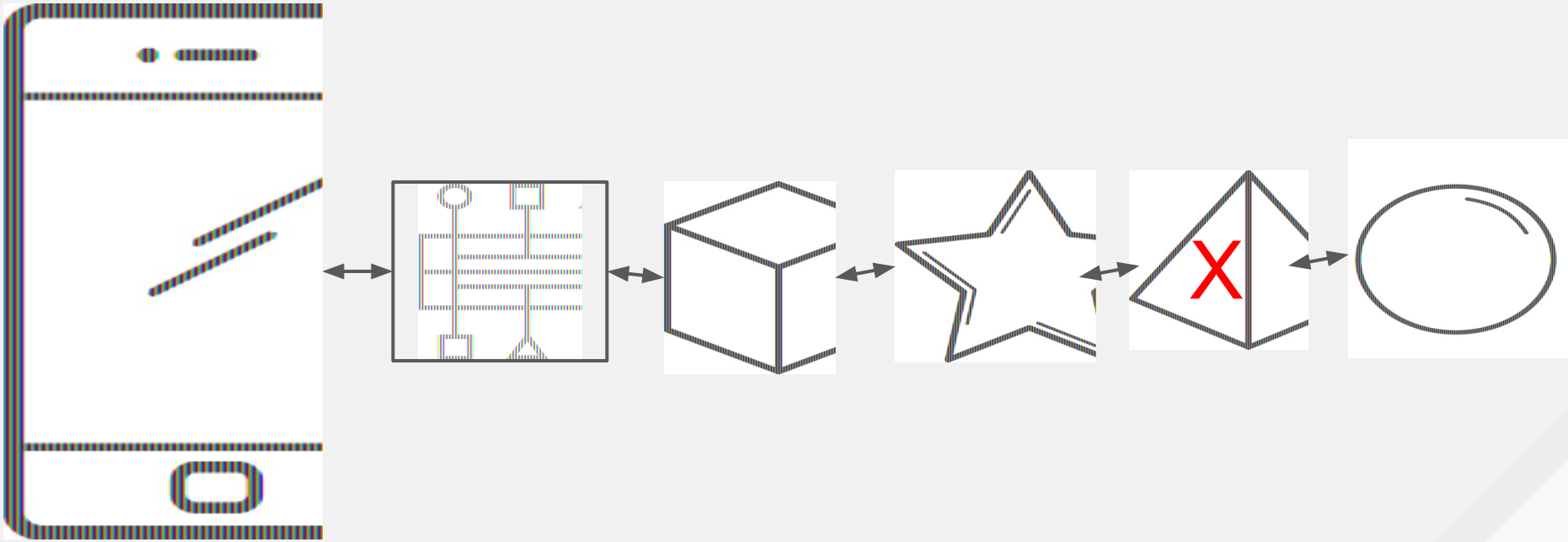
Example



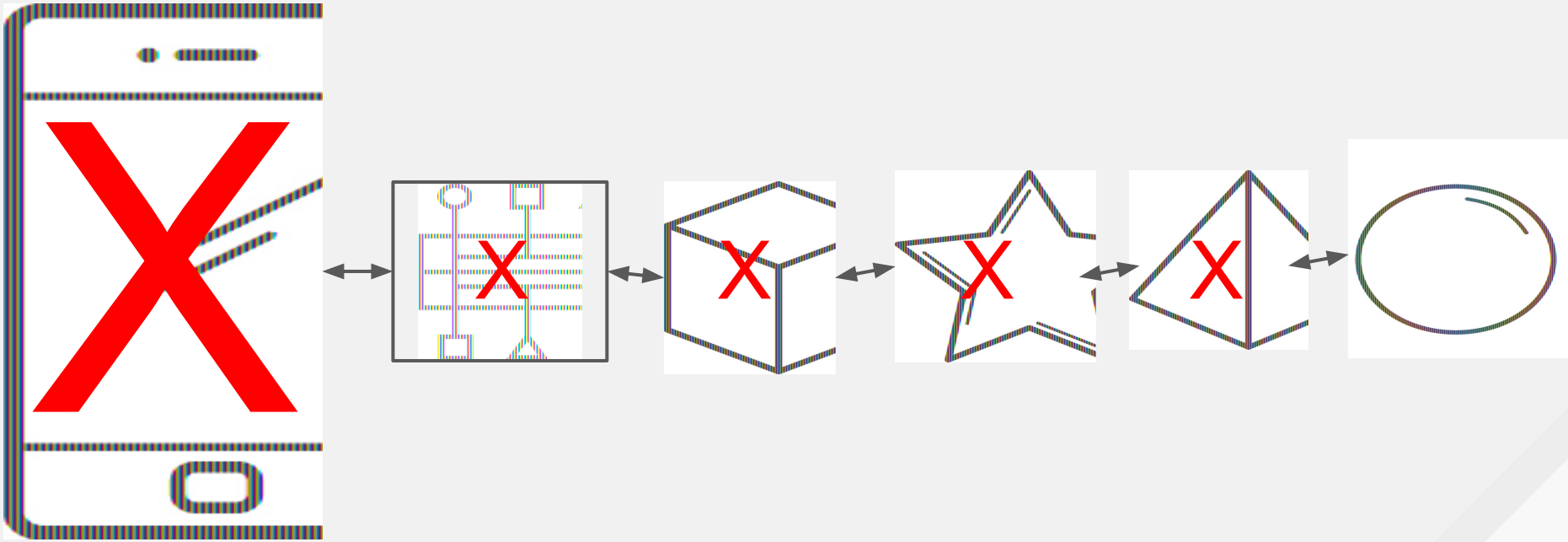
Chaining



Chaining (Fail)



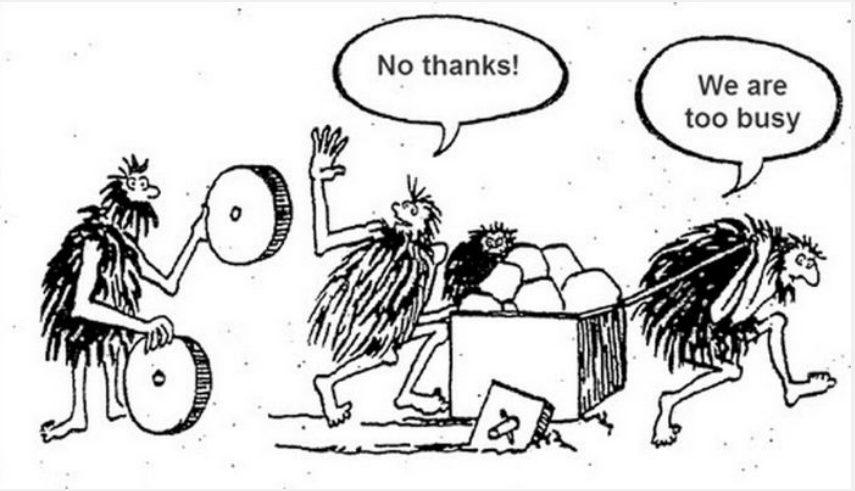
Chaining (Cascading Fail)



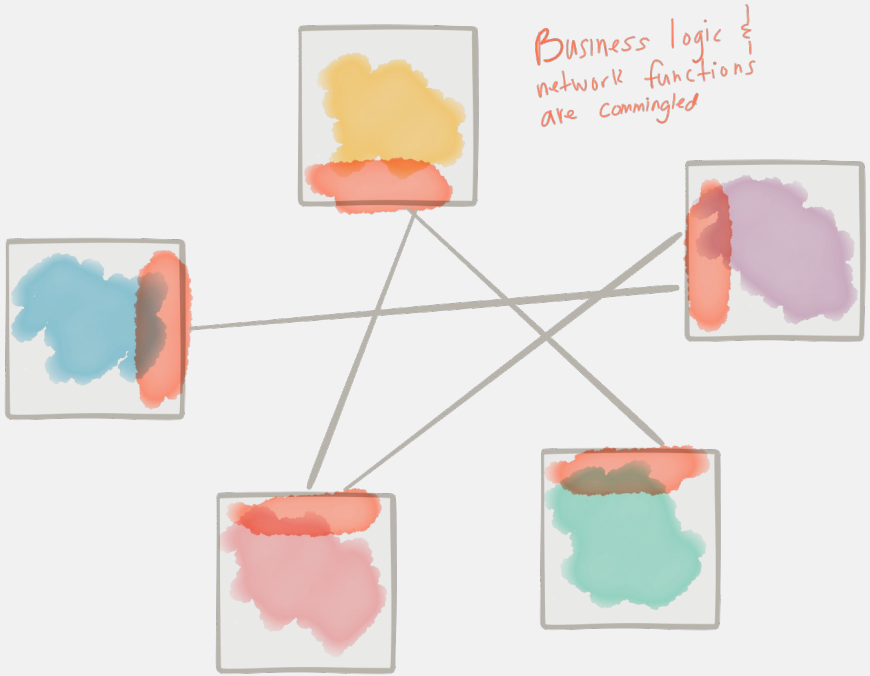
POSSIBLE SOLUTIONS

Today, Developers do this:

- Circuit Breaking
- Bulkheading
- Timeouts/Retries
- Service Discovery
- Client-side Load Balancing



TOO MUCH INFRASTRUCTURE IN BUSINESS LOGIC



BUT I'M USING…


```
spring-cloud-netflix-hystrix
spring-cloud-netflix-zuul
spring-cloud-netflix-eureka-client
spring-cloud-netflix-ribbon
spring-cloud-netflix-atlas
spring-cloud-netflix-spectator
spring-cloud-netflix-hystrix-stream
...
@Enable....150MagicThings
```

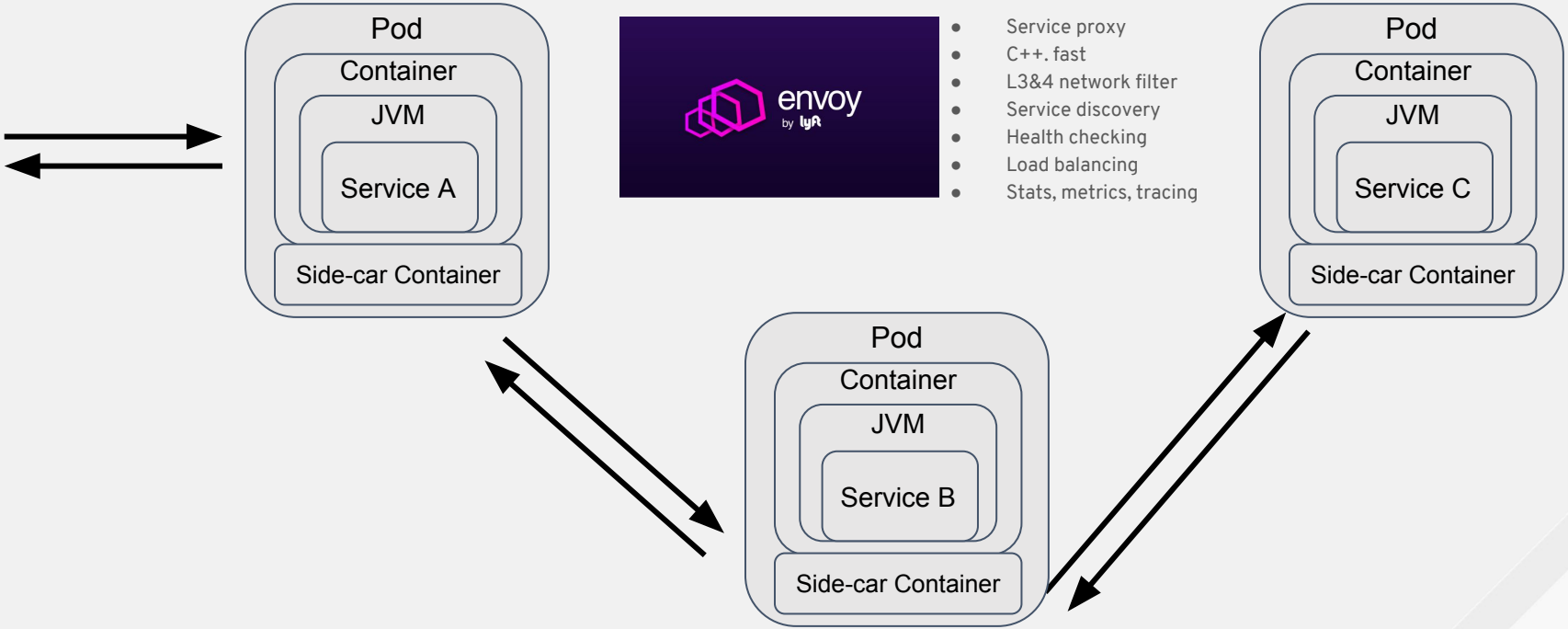

```
org.wildfly.swarm.hystrix
org.wildfly.swarm.ribbon
org.wildfly.swarm.topology
org.wildfly.swarm.camel-zookeeper
org.wildfly.swarm.hystrix
...
```


```
vertx-circuit-breaker
vertx-service-discovery
vertx-dropwizard-metrics
Vertx-zipkin
...
```

+ Node.js
+ Go
+ Python
+ Ruby
+ Perl
+ ....

SIDECARS

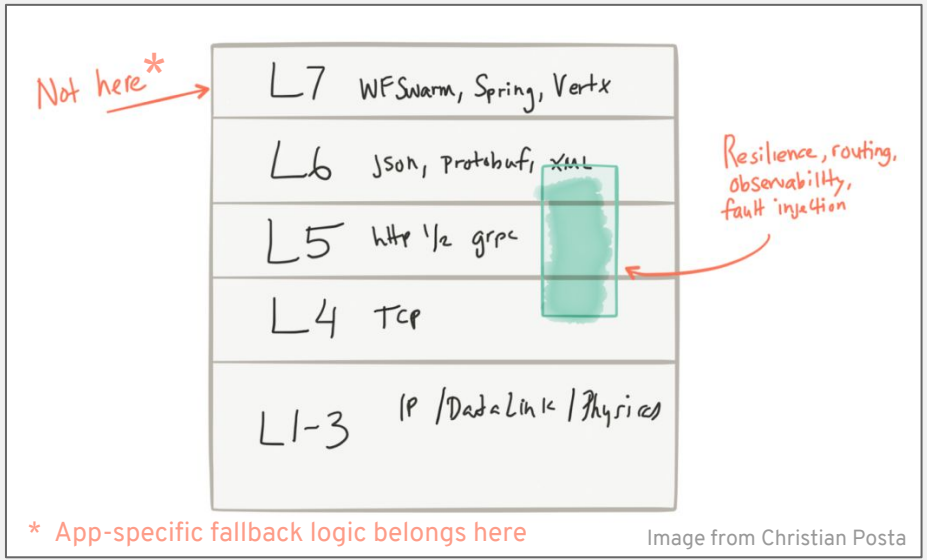
PODS WITH TWO CONTAINERS



Istio - Sail

(Kubernetes - Helmsman or ship's pilot)

ISTIO - A ROBUST SERVICE MESH FOR MICROSERVICES



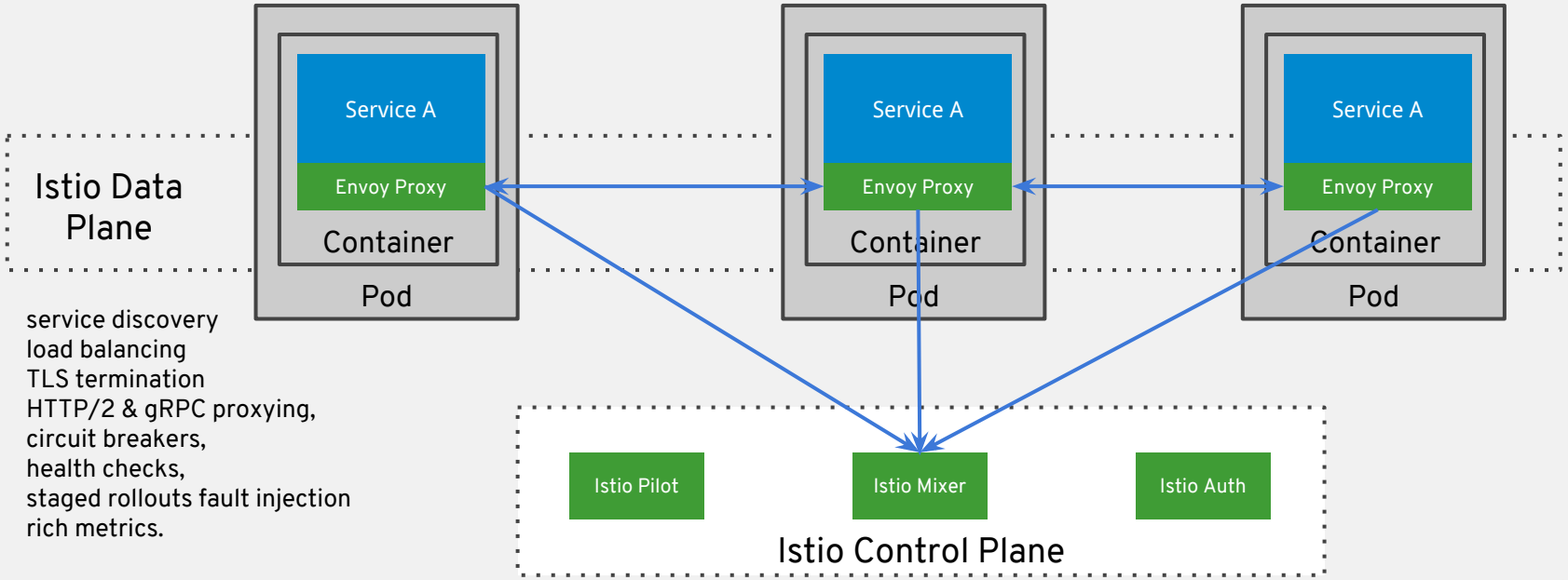
Key Features

- Intelligent routing and load balancing
- Fleet-wide, in-depth observability
- Resiliency across languages and platforms
- Fault injection
- Developer productivity
- Policy driven ops
- Circuit breaking, outlier detection
- Timeouts/retries
- Rate limiting
- Secure by default
- Incremental, unobtrusive adoption

Further Reading :

https://blog.openshift.com/red-hat-istio-launch/
https://istio.io/blog/istio-service-mesh-for-microservices.html
http://blog.christianposta.com/microservices/the-hardest-part-of-microservices-calling-your-services/

ISTIO - A ROBUST SERVICE MESH FOR MICROSERVICES



- service discovery
- load balancing
- TLS termination
- HTTP/2 & gRPC proxying,
- circuit breakers,
- health checks,
- staged rollouts fault injection
- rich metrics.

- traffic mgmt
- discovery

- access control
- usage policies
- telemetry collection

- authentication
- policy enforcement
- Id & credentials

MICROSERVICES 3.0 - SERVICE MESH

Code Independent:

- Intelligent Routing and Load-Balancing
  - A/B Tests
  - Canary Releases
  - Dark Launches
- Distributed Tracing
- Circuit Breakers
- Fine grained Access Control
- Telemetry, metrics and Logs
- Fleet wide policy enforcement

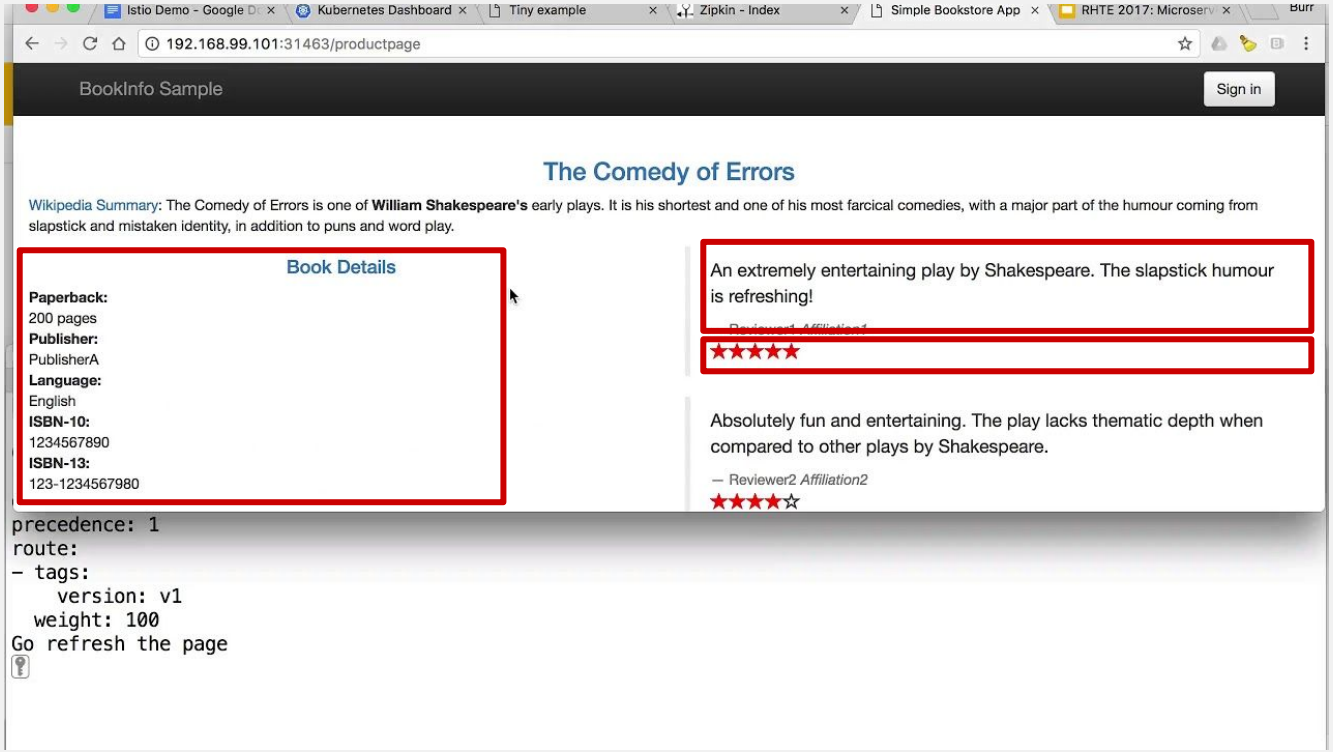
LAB: DETECTING AND PREVENTING ISSUES IN DISTRIBUTED APPS WITH ISTIO

GOAL FOR LAB

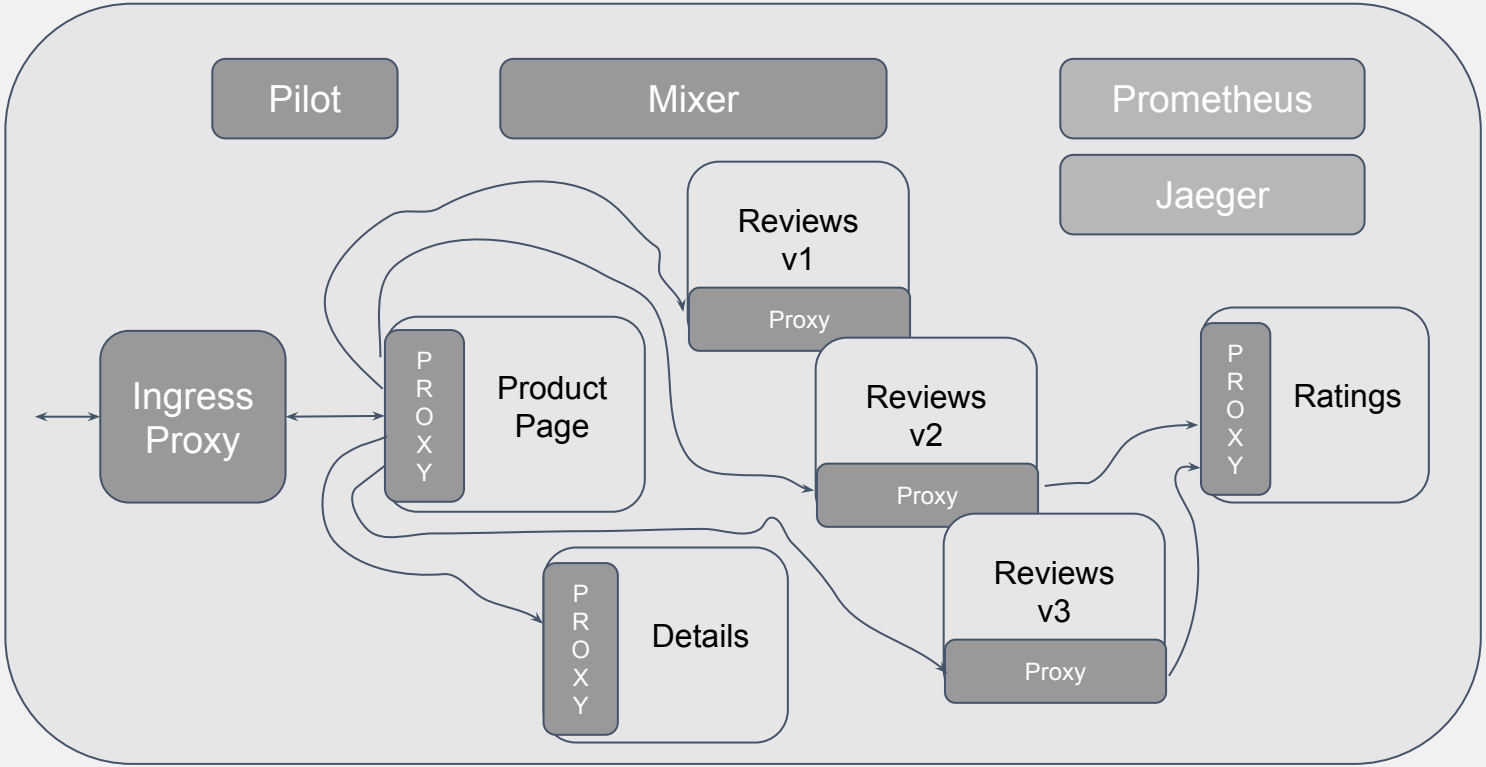
In this lab you will learn:

- How to **install Istio** onto **OpenShift Container Platform**
- How to deploy apps with **sidecar proxies**
- How to generate and visualize **deep metrics** for apps
- How to **alter routing** dynamically
- How to **inject faults** for testing
- How to do **rate limiting**
- How Istio implements **circuit breaking** and **distributed tracing**

SAMPLE APP: “BookInfo”



SAMPLE APP: “BookInfo”



LAB: DETECTING AND PREVENTING ISSUES IN DISTRIBUTED APPS WITH ISTIO

SCENARIO 7

PREVENT AND DETECT ISSUES IN A DISTRIBUTED SYSTEM

WRAP-UP AND DISCUSSION

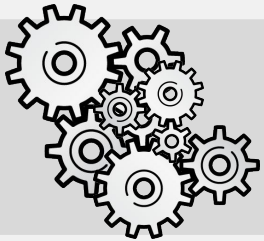
RESULT OF LAB

In this lab you learned:

- How to install Istio onto OpenShift Container Platform
- How to deploy apps with sidecar proxies
- How to generate and visualize deep metrics for apps
- How to alter routing dynamically
- How to inject faults for testing
- How to do rate limiting
- How Istio implements circuit breaking and distributed tracing
- Use cases for service mesh

MICROSERVICES 4.0?

Service



> Autonomous
> Loosely-coupled

Microservice

> Single Purpose
> Stateless
> Independently Scalable
> Automated

Function

> Single Action
> Event-sourced
> Ephemeral

SERVERLESS PROJECTS / SERVICES

syncano

webtask

APACHE OpenWhisk™

Iron.io

APEX | SERVERLESS INFRASTRUCTURE

Microsoft Azure

Back&

SERVERLESS

http://funcatron.org

fission

CLOUD FUNCTIONS BETA

stdlib

serverless-docker

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos