redhat.

# MONOLITHS TO MICROSERVICES: APP TRANSFORMATION

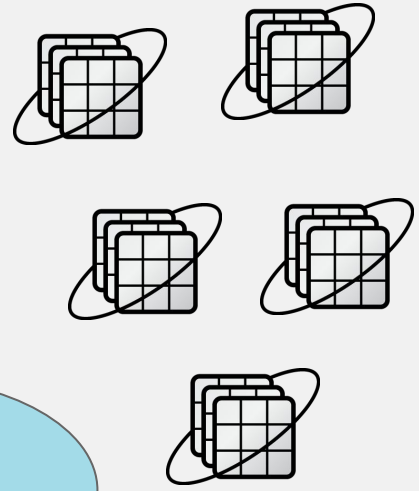
## Hands-on Technical Workshop

Ben Farr
Andy Yuen
Bryon Baker

Senior Solution Architects
Red Hat Australia and New Zealand

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

# MICROSERVICES



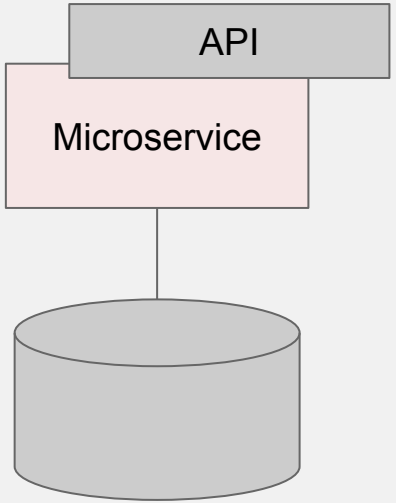
Faster Software Delivery

Resource Scalability

Failure Isolation

# MICROSERVICES

- Simple and small
- Independence
- Boundary and Replaceable



# WHY MONOLITH TO MICROSERVICES

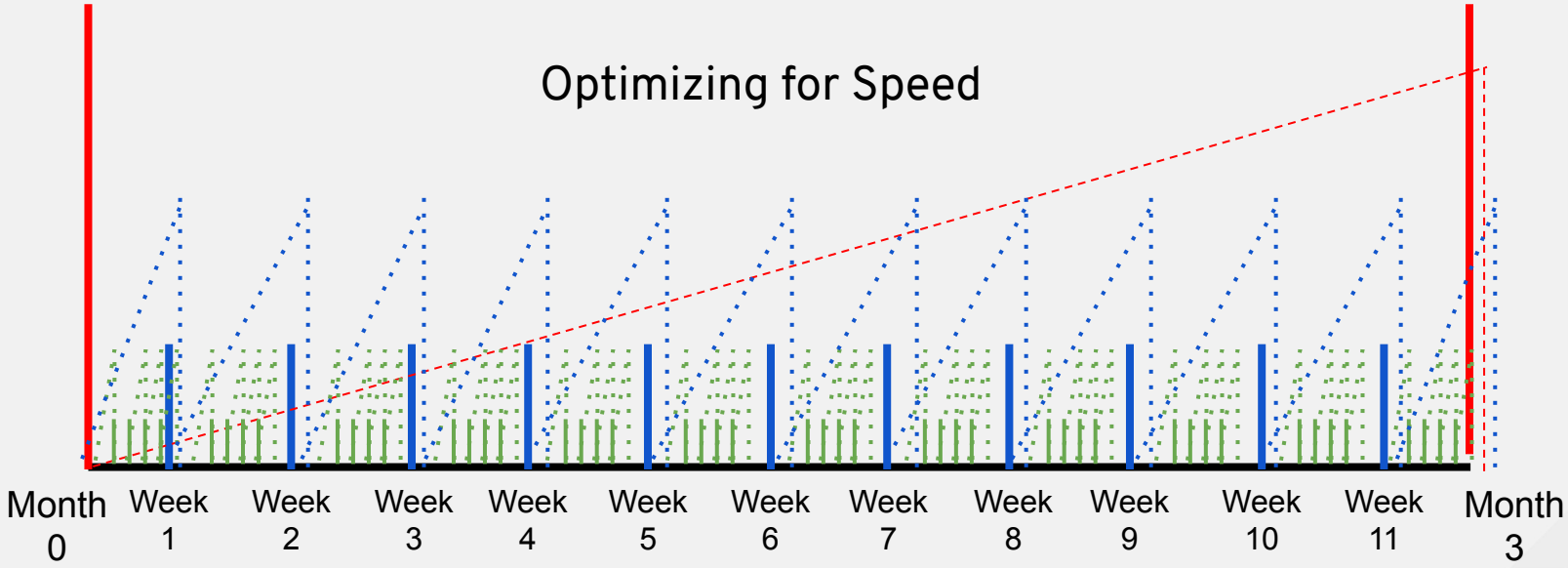
Break things down (organizations, teams, IT systems, etc) down into smaller pieces for greater parallelization and autonomy and focus on reducing time to value.

# REDUCING TIME TO VALUE

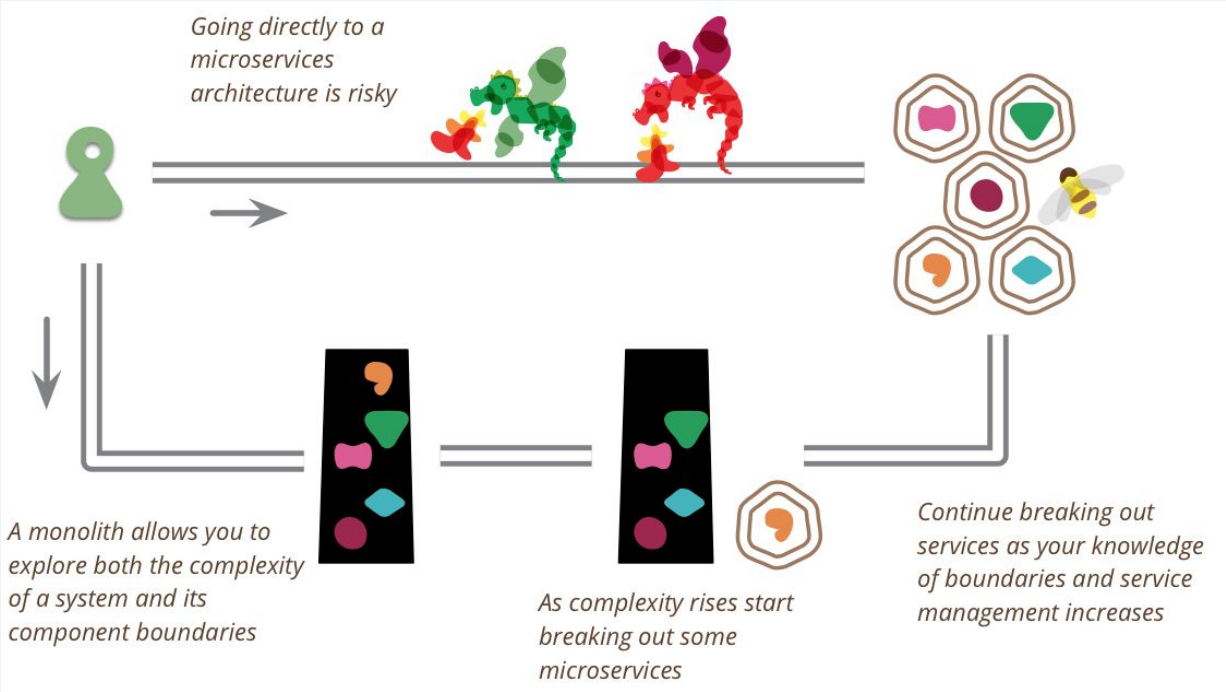
Monolith Lifecycle

Fast Moving Monolith

Microservices



# MONOLITH FIRST ?




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

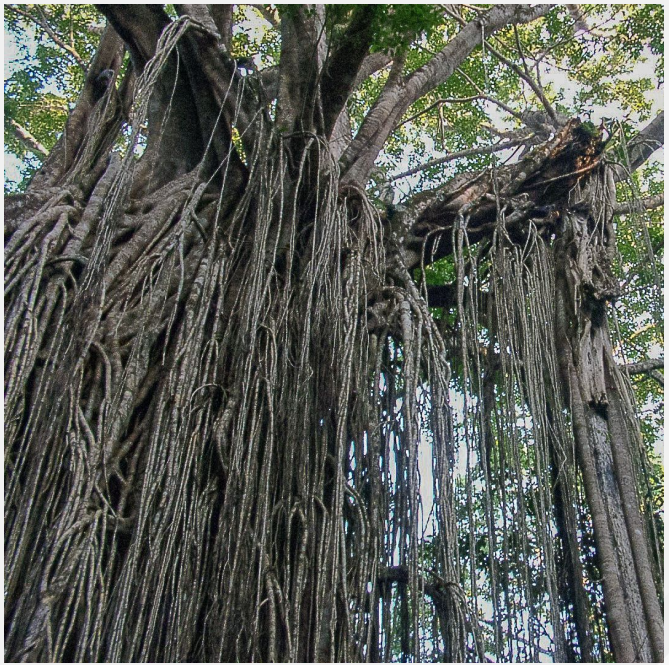
# THE BIGGER PICTURE: THE PATH TO CLOUD-NATIVE APPS

## A DIGITAL DARWINISM

# "STRANGLING" THE MONOLITH

Going directly to a microservice architecture is risky.

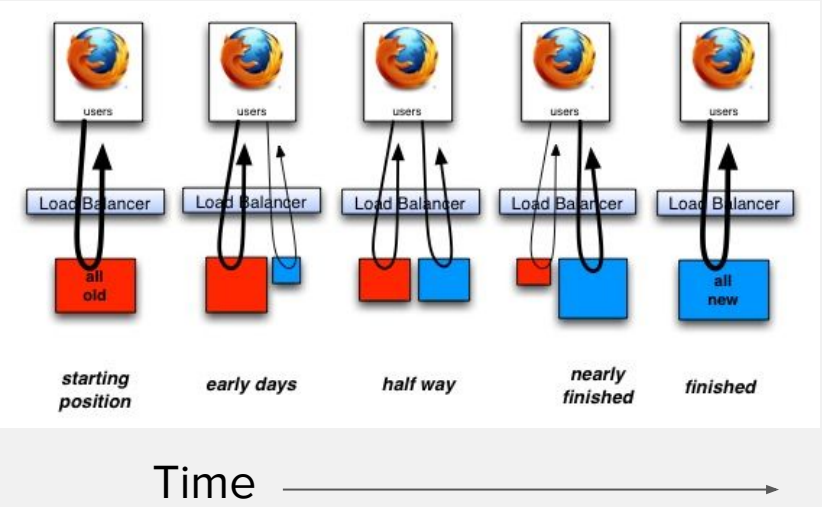
An alternative route is to gradually create a new system around the edges of the old, letting it grow slowly over several years until the old system is strangled.



https://www.martinfowler.com/bliki/StranglerApplication.html

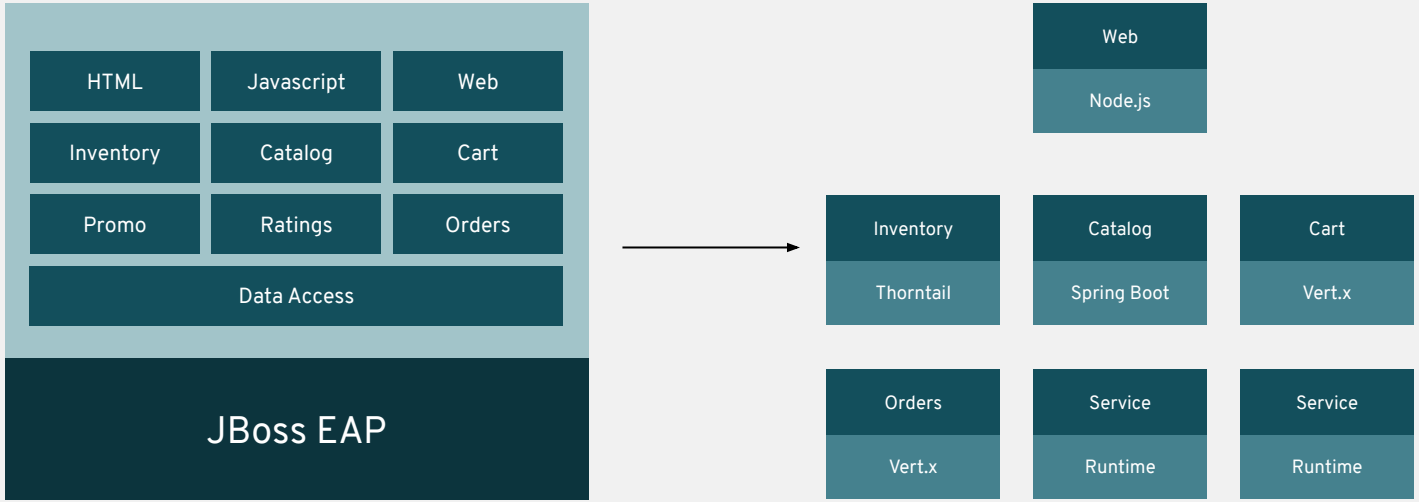
# "STRANGLING" THE MONOLITH

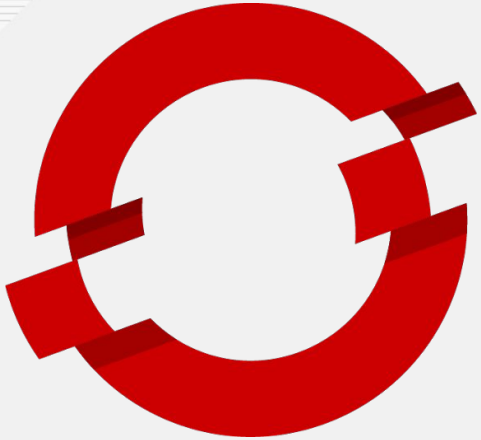
- Strangling - **incrementally** replacing functionality in app with something better (cheaper, faster, easier to maintain).
- As functionality is replaced, “dead” parts of monolith can be removed/retired.
- You can also wait for all functionality to be replaced before retiring anything!
- You can optionally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



# "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

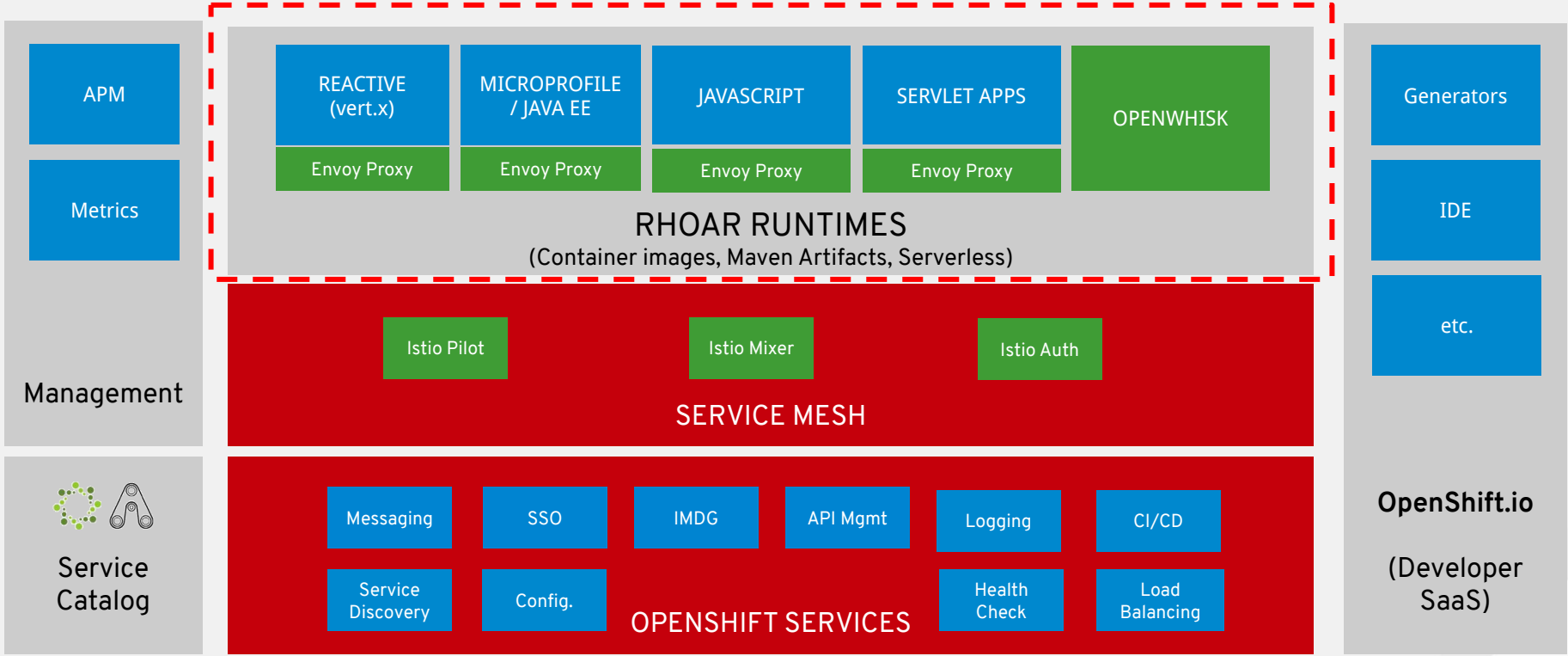




# RED HAT® OPENSHIFT

## Application Runtimes

# RHOAR PRODUCT ARCHITECTURE






== Future


ENTERPRISE JAVA

RED HAT® JBOSS®
ENTERPRISE
APPLICATION PLATFORM

SERVLET APPS


JAVA MICROSERVICES

JAVASCRIPT FLEXIBILITY


REACTIVE SYSTEMS

VERT.X

TOMCAT SIMPLICITY

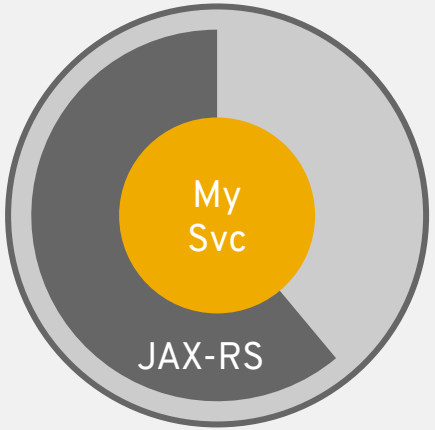
RED HAT® JBOSS®
WEB SERVER

# THORNTAIL

# (WILDFLY SWARM)

# JAVA EE MICROSERVICES

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



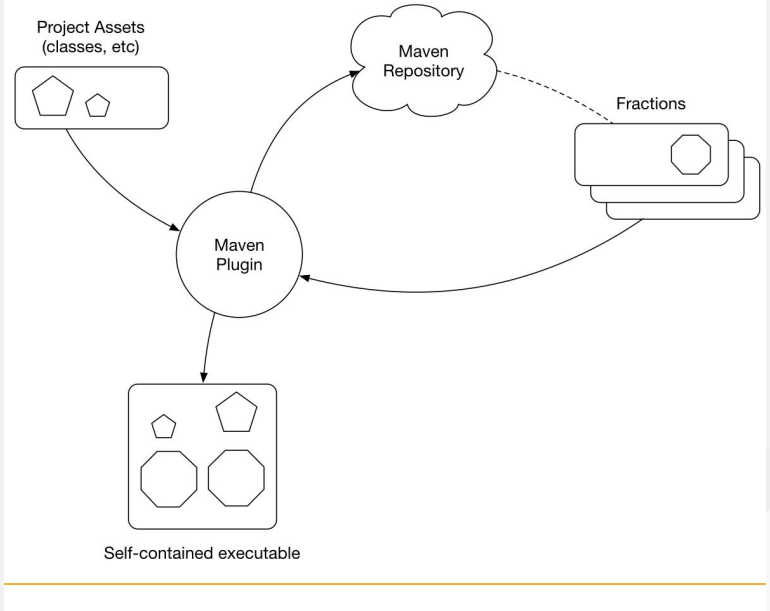

```
$ java -jar my_microservice.jar
```

## WHAT IS THORNTAIL?

- Packaged as an Uber Jar (self-contained, executable Java archive)
- Implementation of MicroProfile specification
- Not intended as JBoss EAP in an uber jar!
  - Focused on Microservice use cases.
  - Not recommended for systems containing a UI.
  - No session replication support.

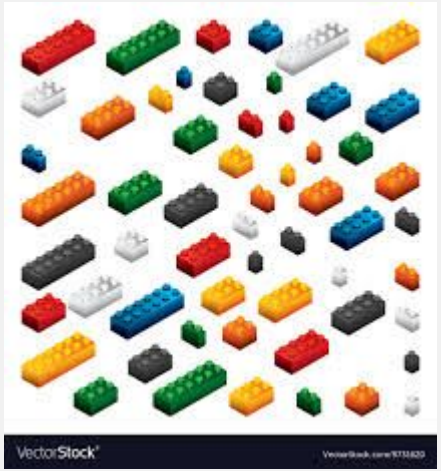
# FRACTIONS

- A tangible unit providing a specific piece of functionality
- Embodied in a maven artifact
- To support the compositional aspect in Thorntail
- 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 THORNTAIL

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



## 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**

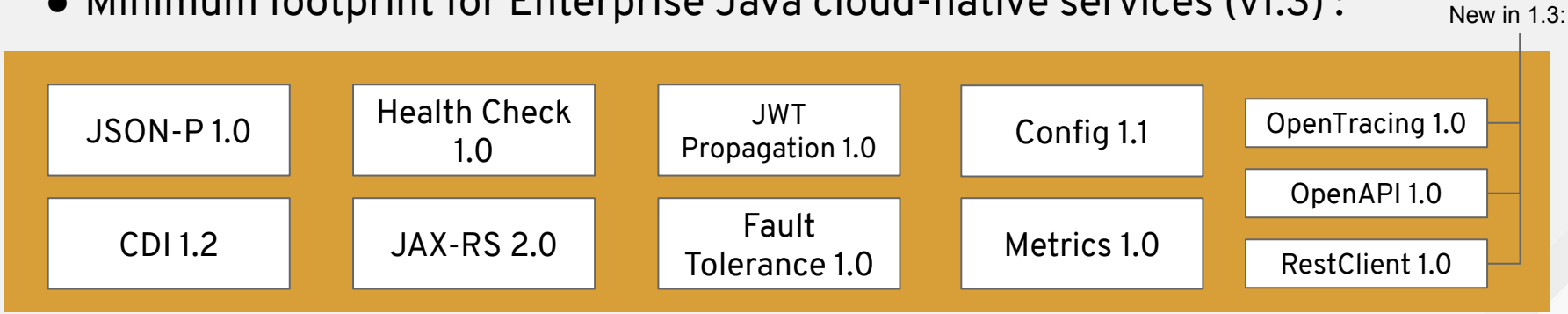
Java EE 7 Web Profile | MicroProfile 1.0

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 and others
- **Thorntail** is **Red Hat's** implementation
- Minimum footprint for Enterprise Java cloud-native services (v1.3) :



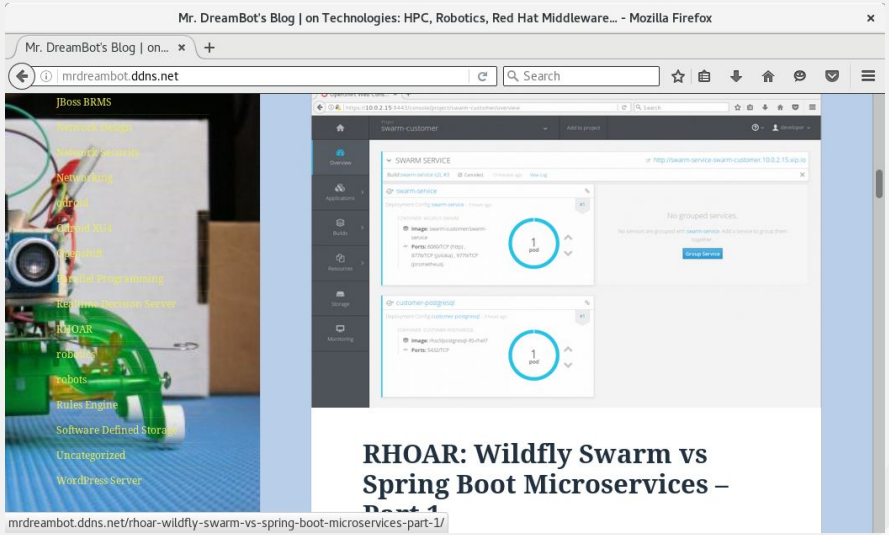



redhat.

# WILDFLY SWARM ARTICLE ON MY BLOG: mrdreambot.ddns.net

[RHOAR: Wildfly Swarm vs Spring Boot Microservices – Part 1]

- Wildfly Swarm
- JEE Developers
- Creating RESTful Services
- JUnit testing
- Deploying to Openshift
- Using ConfigMap



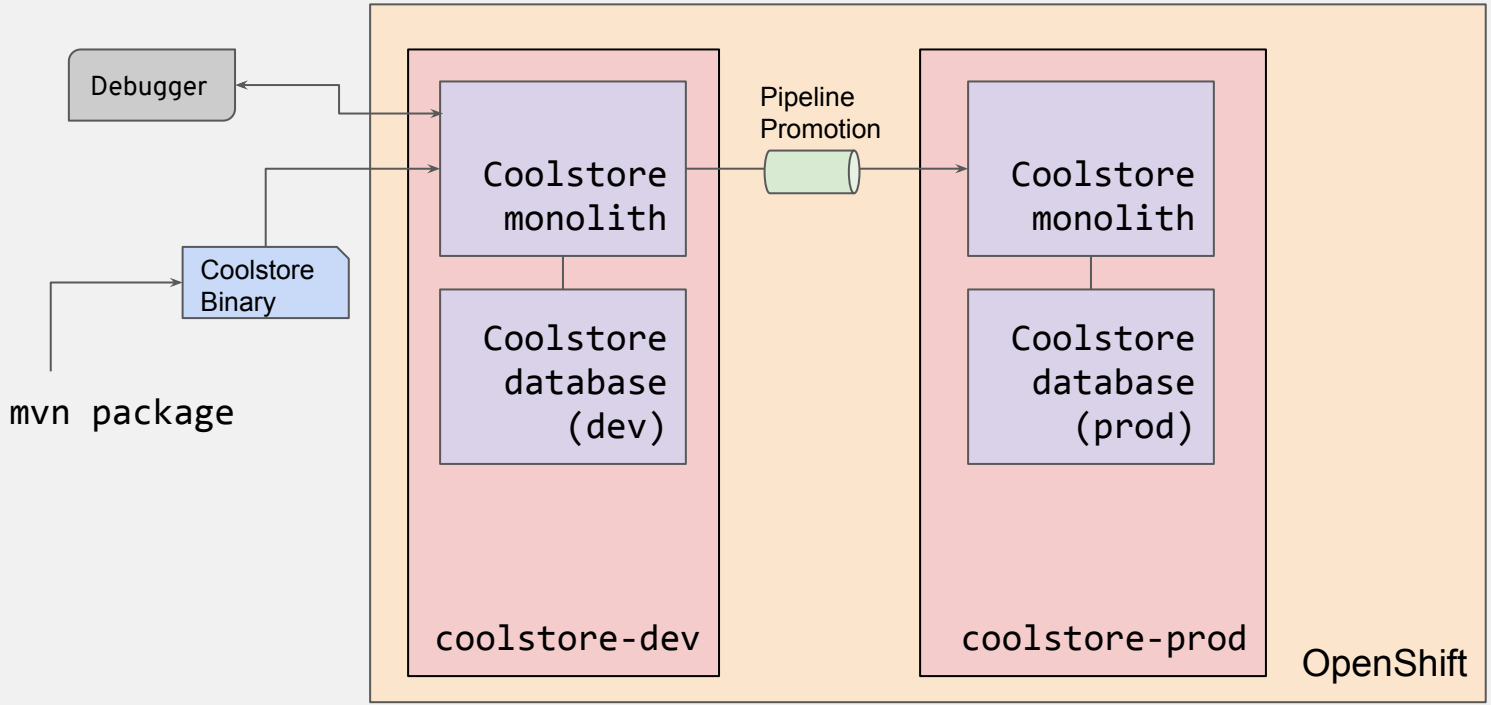
# LAB: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT

# GOAL FOR LAB

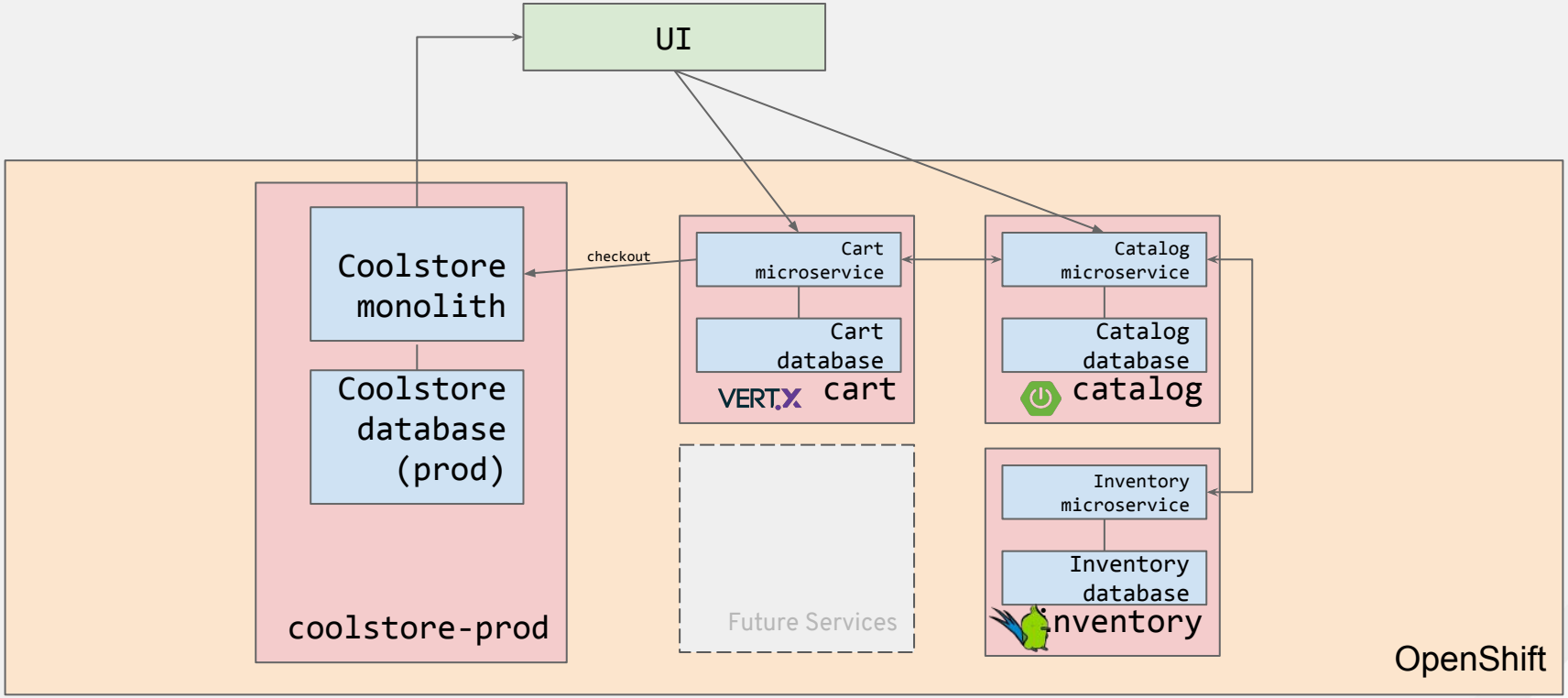
In this lab you will learn:

- 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



# TARGET STATE - MICROSERVICES BASED

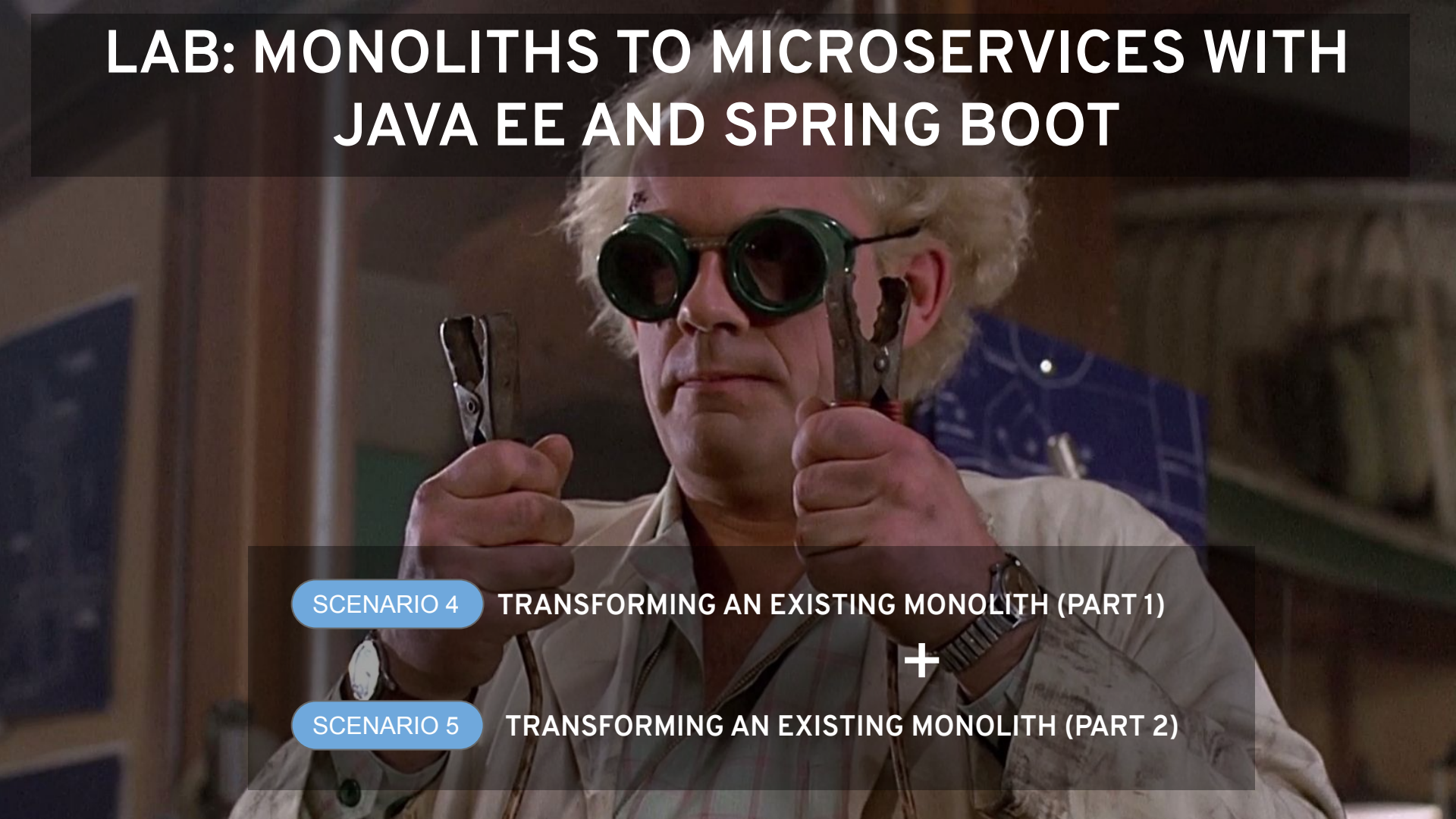


# MICROSERVICES

In the following labs you will develop the following microservices to break up the monolith:

- The Inventory Service using WildFly Swarm
- The Catalog Service using Spring Boot
- The Shopping Cart Service and Shipping Service using Vert.x

# 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)

# SPRING

# SPRING BOOT

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 BOOT


- Provide a radically faster and widely accessible getting started experience for all Spring development.
- Create Java applications that can be started using java -jar (but still allow for more traditional war deployments).
- Take an opinionated view of building Spring applications. Provide default component dependencies and automatic configuration of components. Convention over Configuration.
- Provide non-functional features common to most projects (metrics, health checks, security, externalized configuration, embedded servers)
- No code generation and no requirement for XML configuration.

# SPRING STARTERS


- Dependency descriptors that you can include in your application
- Organized around technologies or features
- Define and manage all the transitive dependencies required for a particular technology
- The complete list of Spring provided starters can be found in the documentation
- Third parties can also provide starters for Spring Boot
  - Apache Camel: camel-spring-boot-starter

# SPRING CLOUD KUBERNETES


- PropertySource Reload to trigger a application reload when changes are detected in ConfigMap
  - Disabled by default
  - Levels of reload: refresh (default), restart_context, shutdown
- Pod Health indicator adds pod-specific health data to Spring Actuator health endpoint
- Kubernetes Profile auto-configuration when running on Kubernetes
- Ribbon discovery for Kubernetes
- Archaius (Netflix OSS configuration management library) ConfigMap property source
- Transparent: does not break when application is running outside of Kubernetes/OpenShift

# 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), Messaging (Rabbit MQ -> AMQ), more

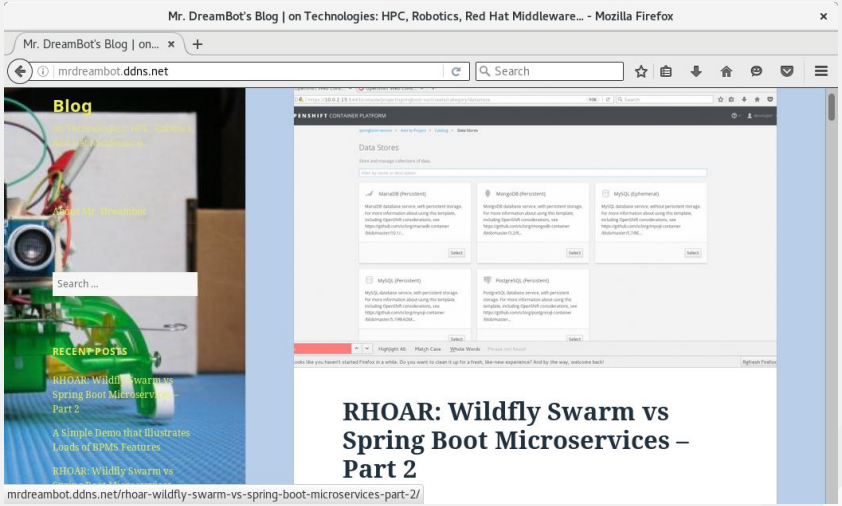
# CLOUD NATIVE SUPPORT IN SPRING

- 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)

# SPRING BOOT ARTICLE ON MY BLOG: mrdreambot.ddns.net

[RHOAR: Wildfly Swarm vs Spring Boot Microservices – Part 2]

- Spring Boot
- Spring Developers
- JPA and JAX-RS
- Creating RESTful Services
- JUnit testing
- Deploying to Openshift
- Using ConfigMap
- Comparing Wildfly Swarm and Spring Boot



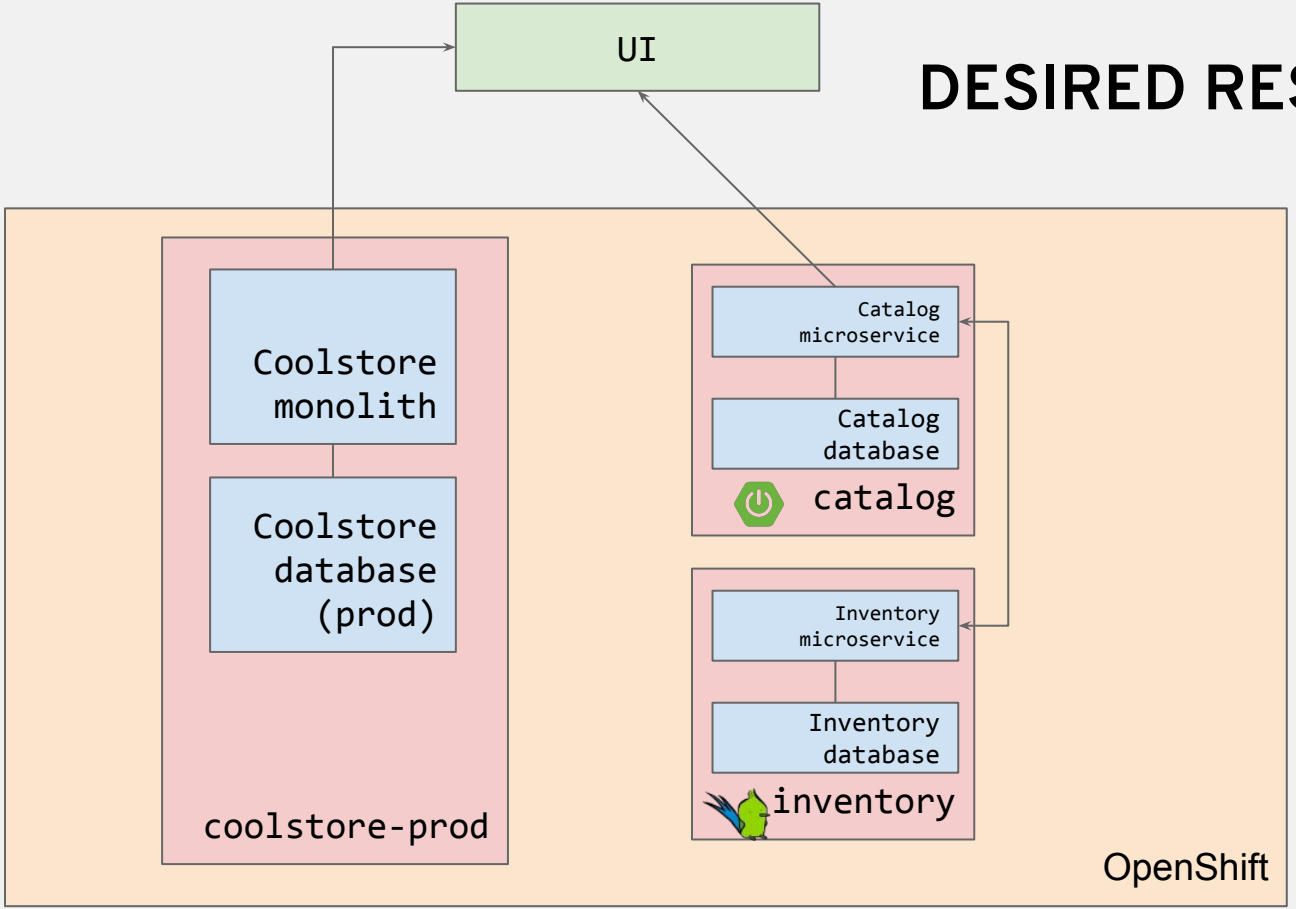
# LAB: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT

# RESULT OF LAB

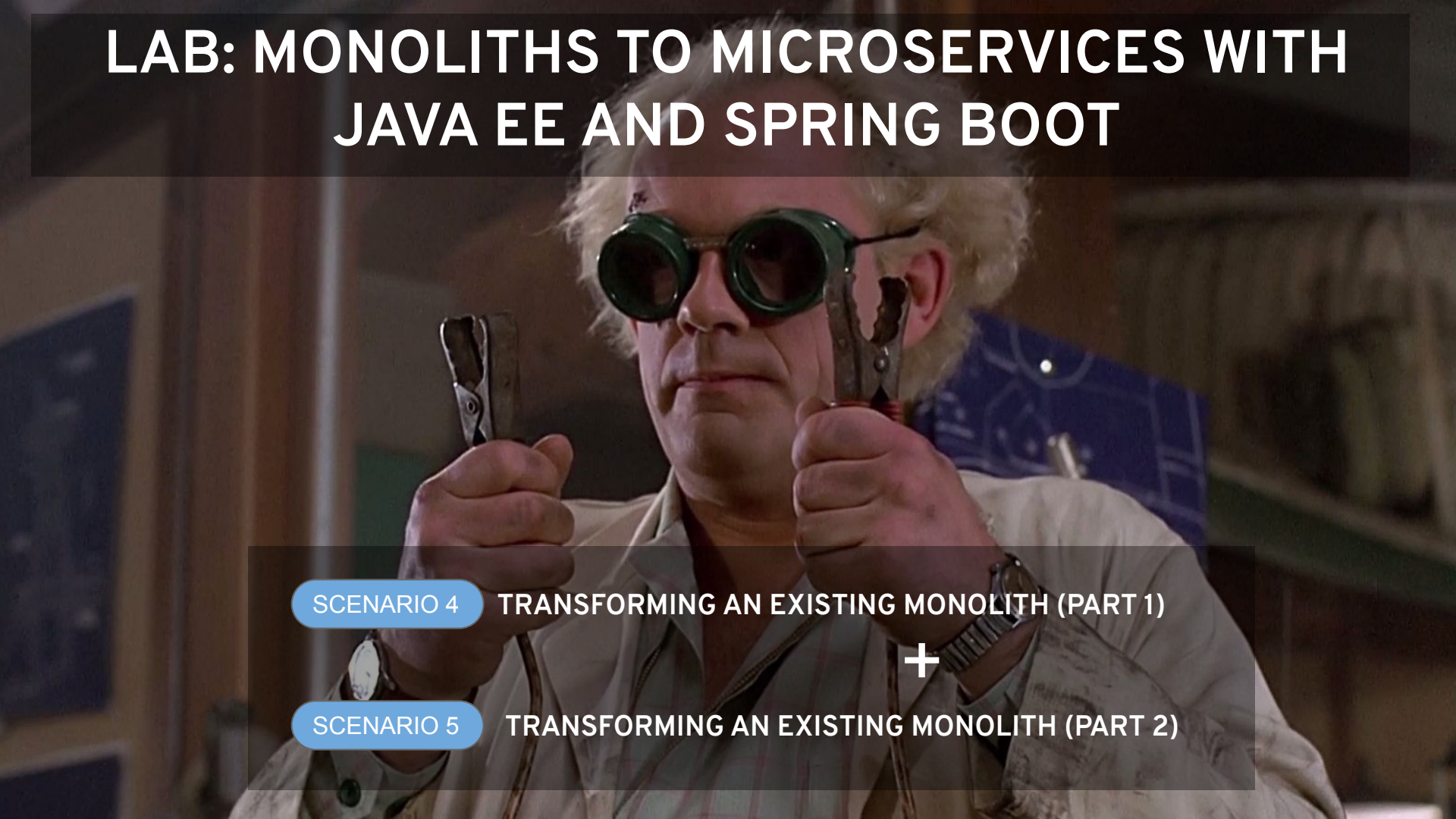
In this lab you learned how to:

- 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



# 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)

# TRAINING COURSES

- **DO180**: Introduction to Containers, Kubernetes, and Red Hat OpenShift
- **JB183**: Red Hat Application Development I: Programming in Java EE
- **DO288**: Red Hat OpenShift Development I: Containerizing Applications
- **JB283**: Red Hat Application Development II: Implementing MicroServices Architectures and Red Hat Certified Enterprise MicroServices Developer - scheduled to be released in May
- **DO292**: Red Hat OpenShift Development II: Creating MicroServices with Red Hat OpenShift Application Runtimes (RHOAR) - scheduled to be released in July

# NEXT STEPS - SELF LEARNING

- App Modernisation
- Openshift
- Monoliths to Microservices 1
- Monoliths to Microservices 2
- App Resiliency and Istio.
  - https://learn.openshift.com/servicemesh/
  - https://github.com/VeerMuchandi/istio-on-openshift

redhat.

# THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos