MONOLITHS TO MICROSERVICES: APP TRANSFORMATION

Hands-on Technical Workshop

Thomas Qvarnström
Sr. Technical Marketing Manager
Middleware BU

James Falkner
Sr. Technical Marketing Manager
Middleware BU

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

WHY MONOLITH TO MICROSERVICES

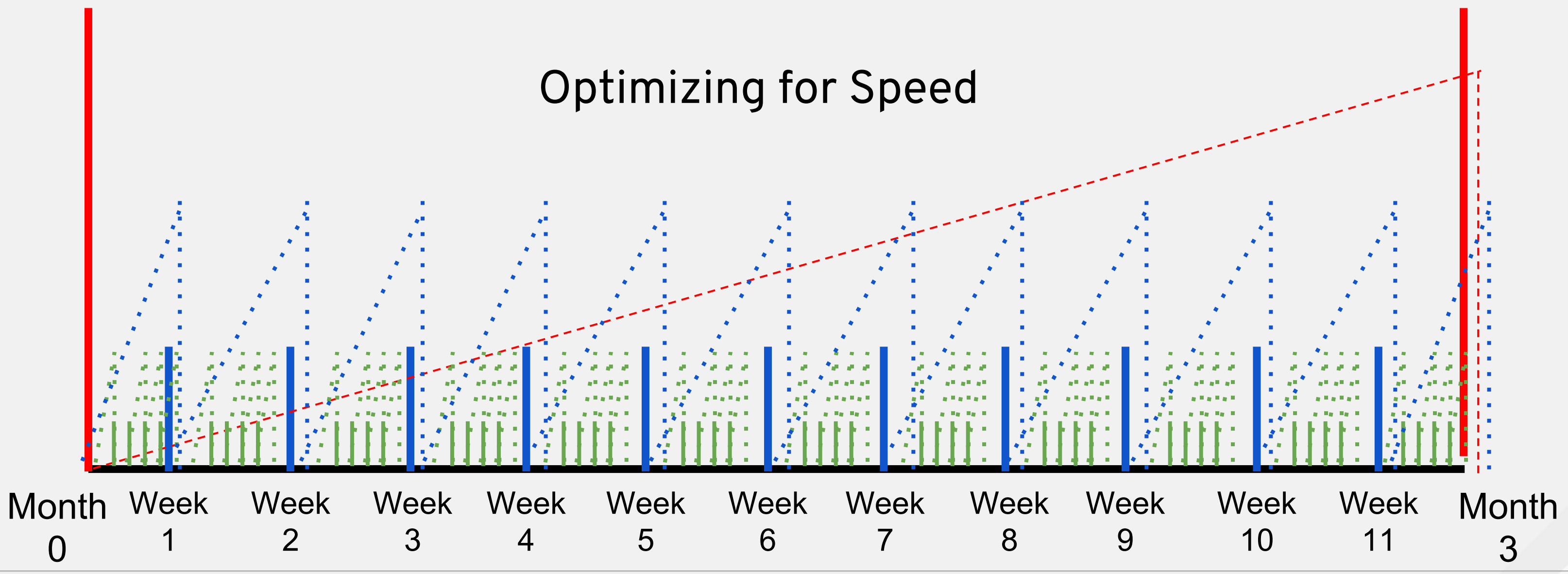
Break things down (organizations, teams, IT systems, etc) down into smaller pieces for greater parallelization and autonomy and focus on reducing time to value.

REDUCING TIME TO VALUE

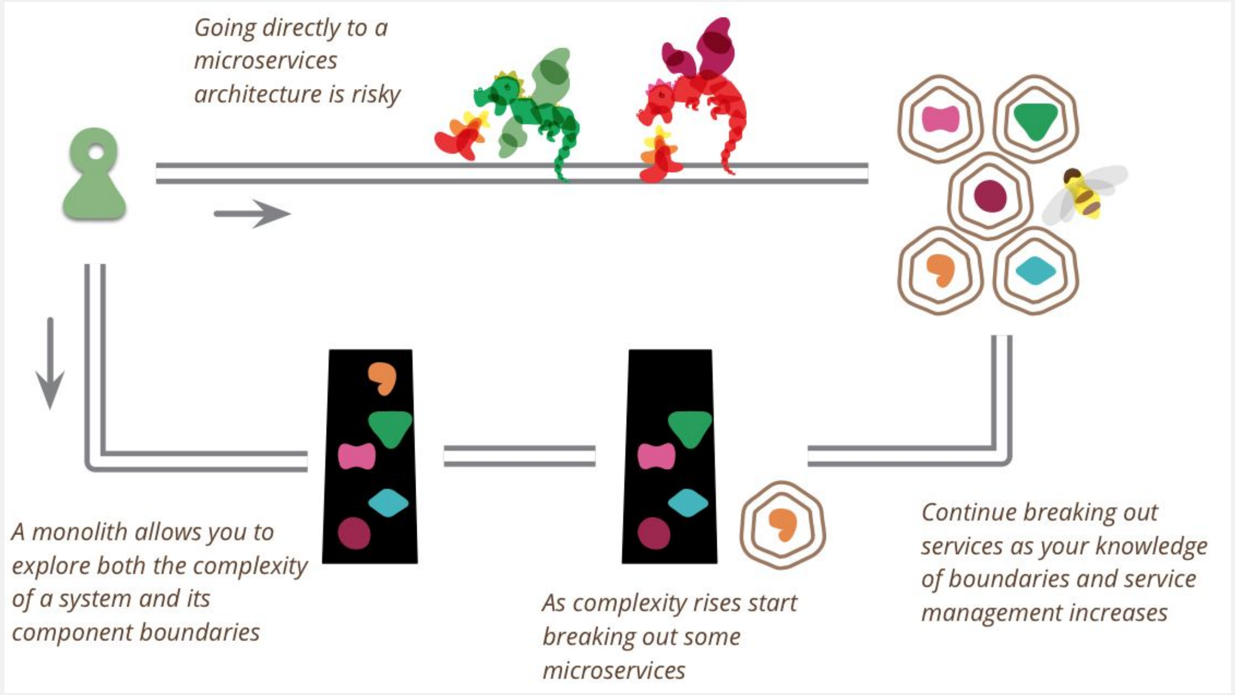
Monolith Lifecycle

Fast Moving Monolith

Microservices



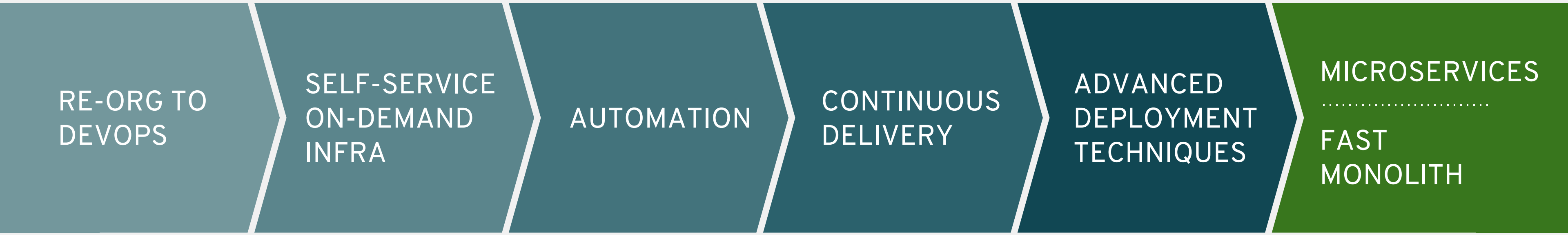
Monolith First?


Going directly to a microservices architecture is risky
A monolith allows you to explore both the complexity of a system and its component boundaries
As complexity rises start breaking out some microservices
Continue breaking out services as your knowledge of boundaries and service management increases

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

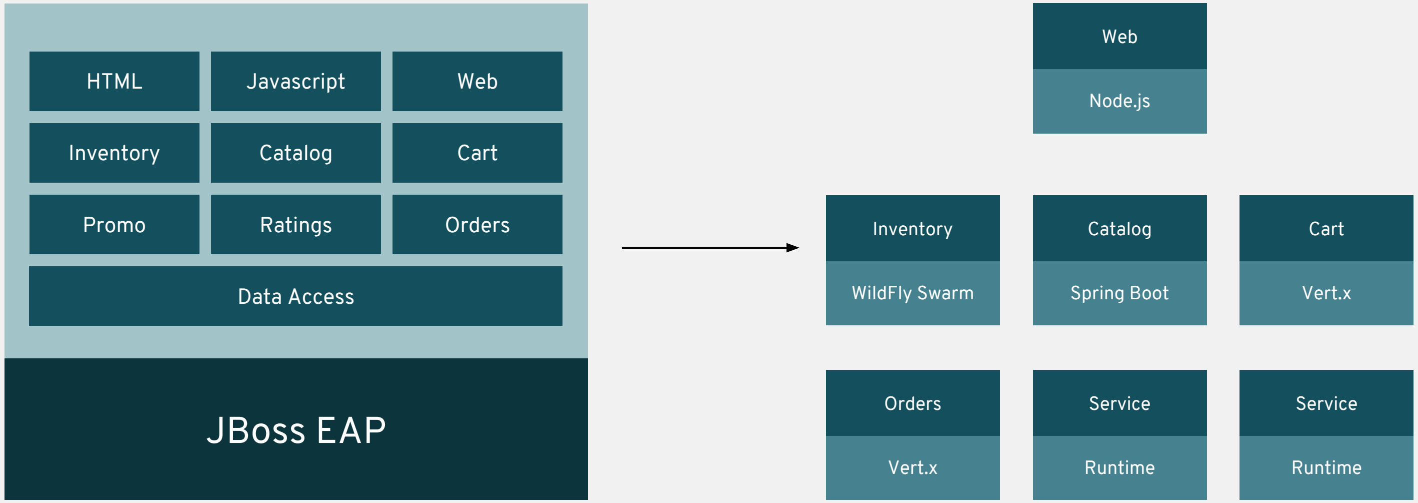
THE BIGGER PICTURE: THE PATH TO CLOUD-NATIVE APPS

A DIGITAL DARWINISM



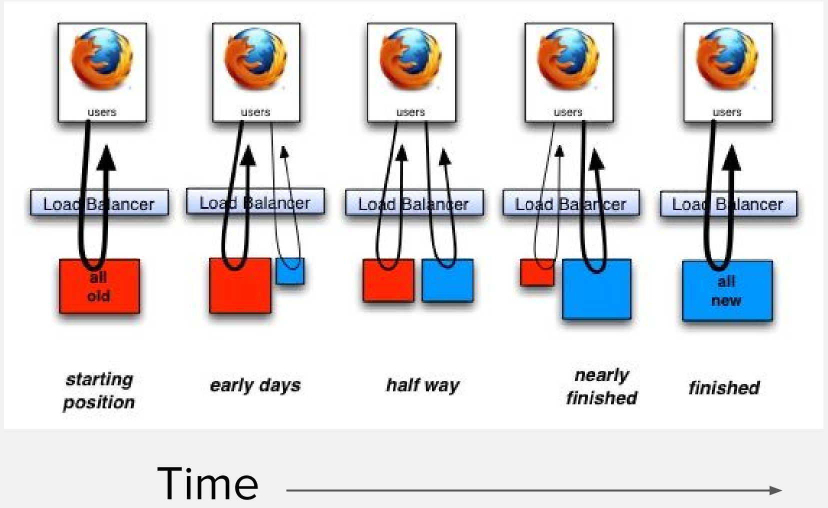
STRANGLING THE MONOLITH

- 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



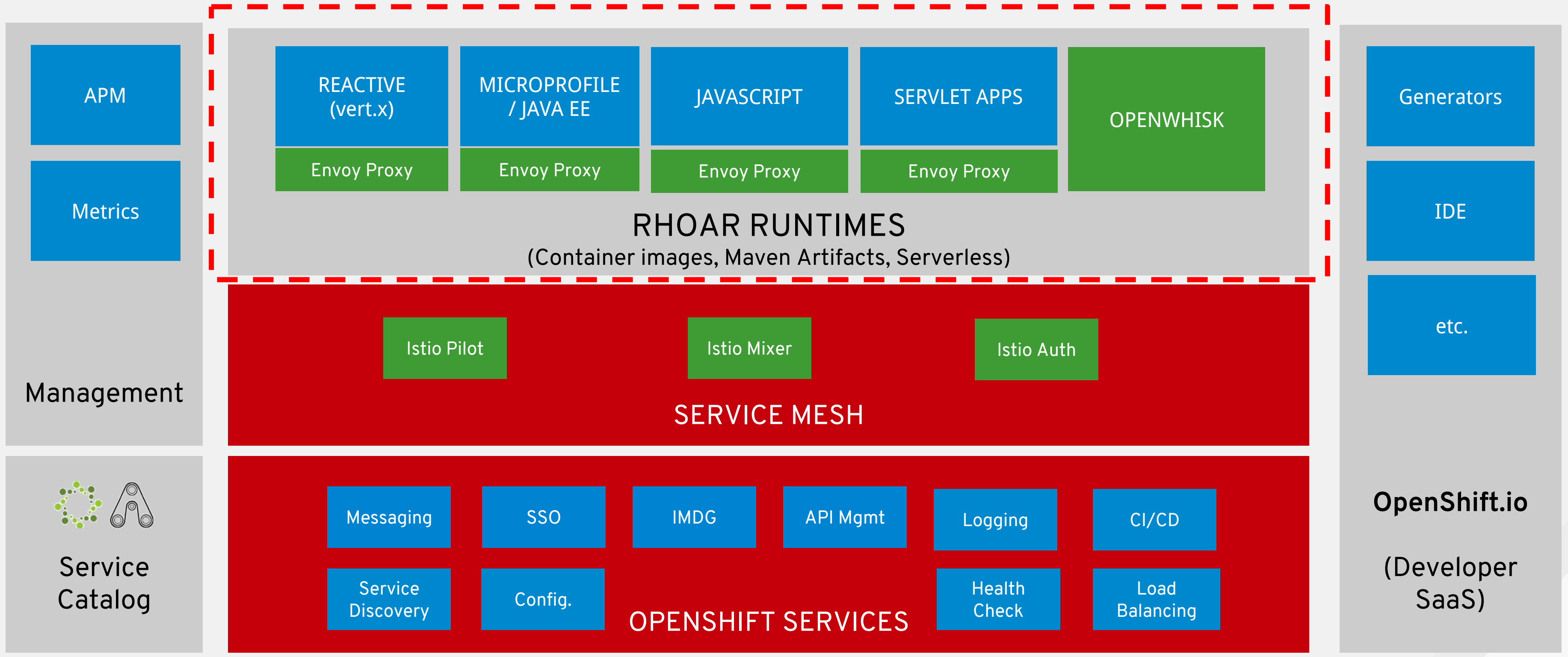
STRANGLING THE MONOLITH

- 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!
- You can optionally include new functionality during strangulation to make it more attractive to business stakeholders.



RED HAT® OPENSHIFT Application Runtimes

RHOAR PRODUCT ARCHITECTURE



🟩 == Future

redhat.

RED HAT® OPENSHIFT
Application Runtimes

ENTERPRISE JAVA

RED HAT® JBOSS® ENTERPRISE APPLICATION PLATFORM

SERVLET APPS

spring

JAVA MICROSERVICES

WildFly SWARM

JAVASCRIPT FLEXIBILITY

node JS®

REACTIVE SYSTEMS

VERT.X

TOMCAT SIMPLICITY

RED HAT® JBOSS® WEB SERVER

SPRING

spring

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

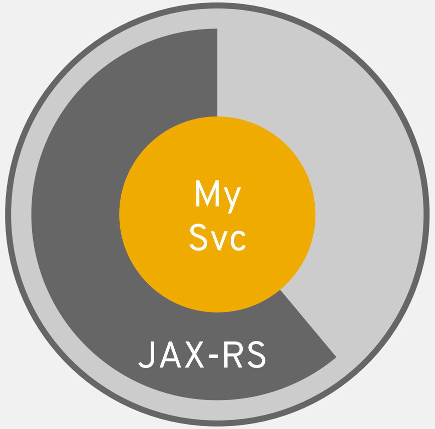
Cloud Native Support in Spring

- 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)

WILDFLY SWARM

Java EE microservices

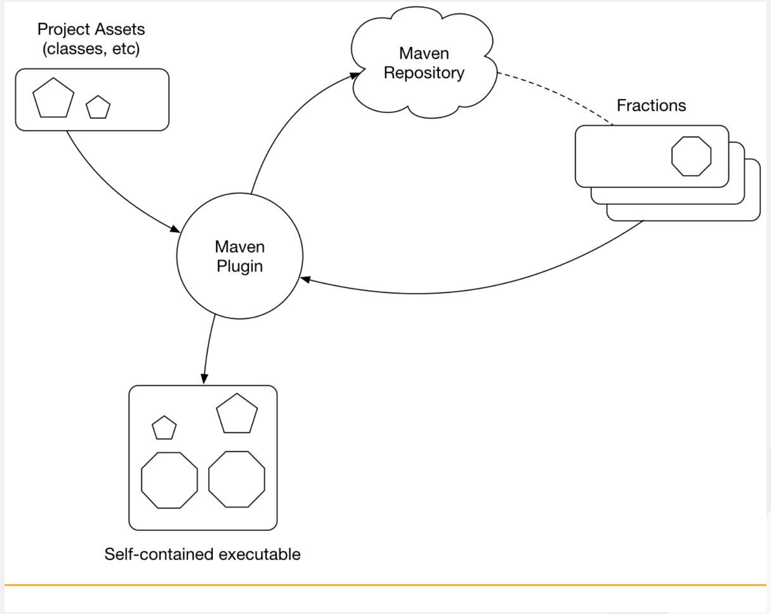
- Leverage Java EE expertise
- Open standard
- Microservices focus
- Optimized for OpenShift
- Super lightweight
- Tooling for Developers
- MicroProfile Implementation



```
$ 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



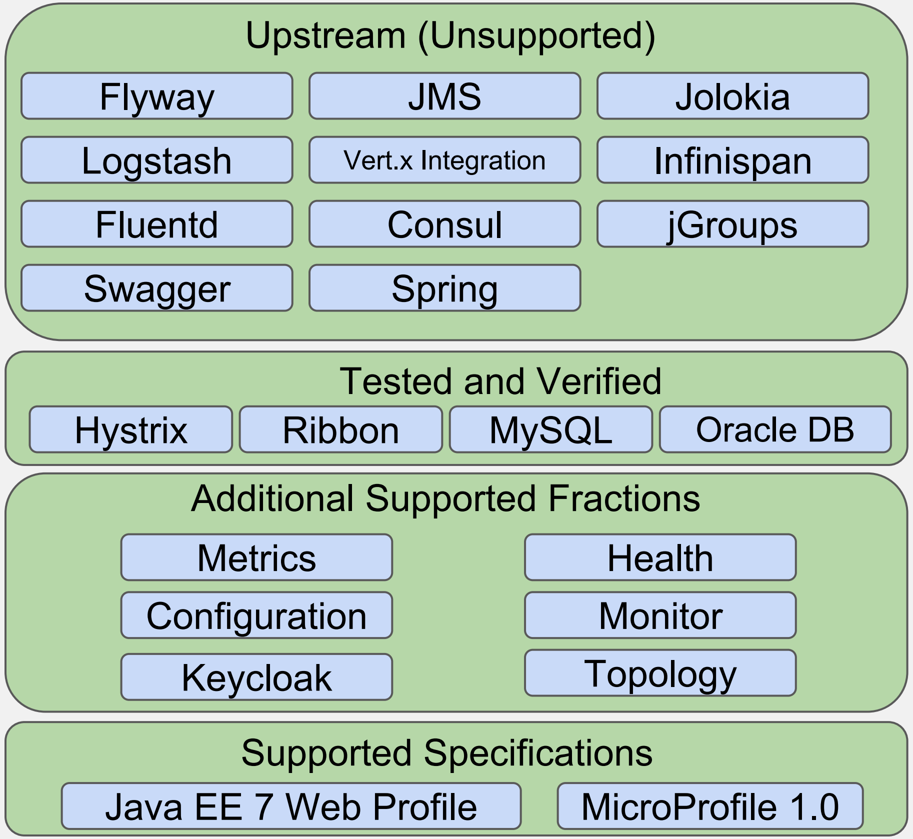
Cloud Native Support in WildFly Swarm

- 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

Build microsevices

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



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) :

JSON-P 1.0

Health Check 1.0

JWT Propagation 1.0

Config 1.1

CDI 1.2

JAX-RS 2.0

Fault Tolerance 1.0

Metrics 1.0

New in 1.3:

OpenTracing 1.0

OpenAPI 1.0

RestClient 1.0

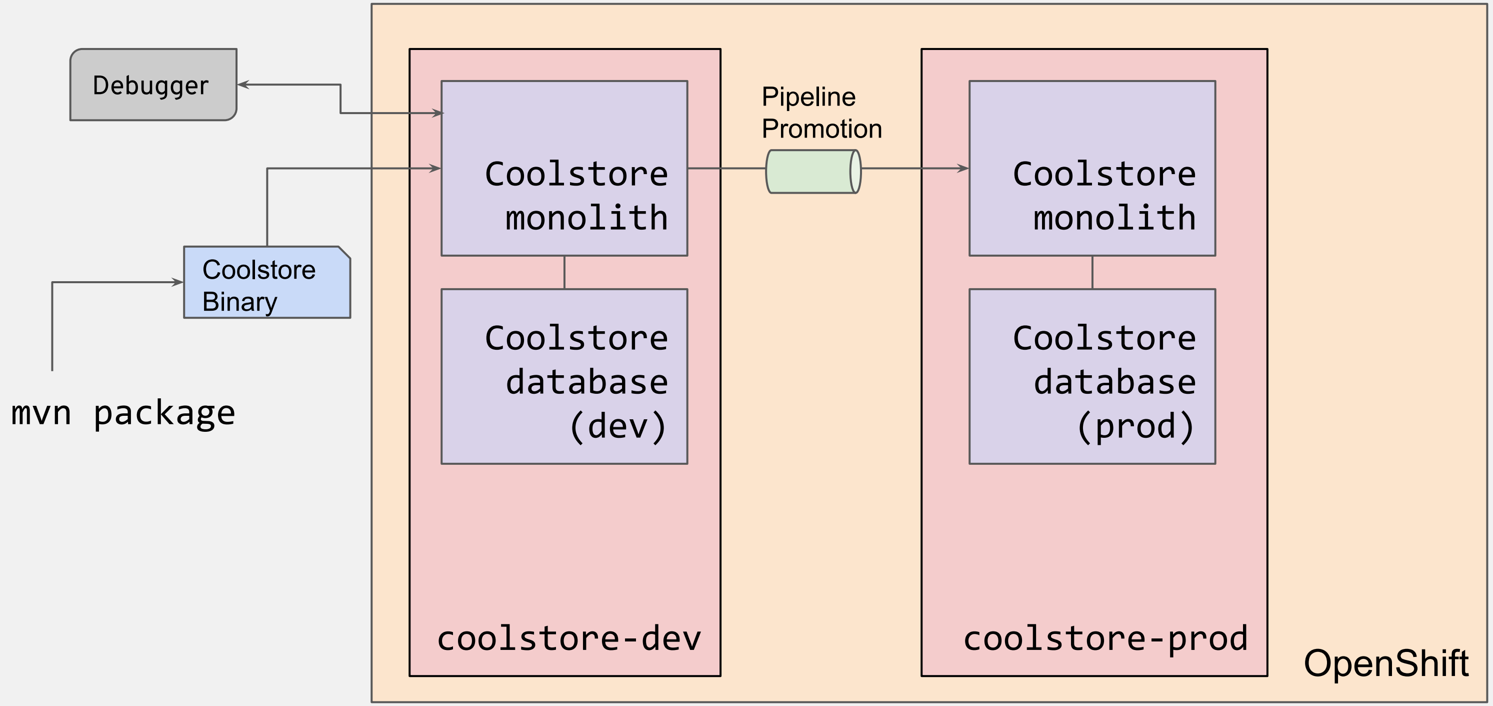
LAB: MONOLITHS TO MICROSERVICES WITH JAVA EE AND SPRING BOOT

GOAL FOR LAB

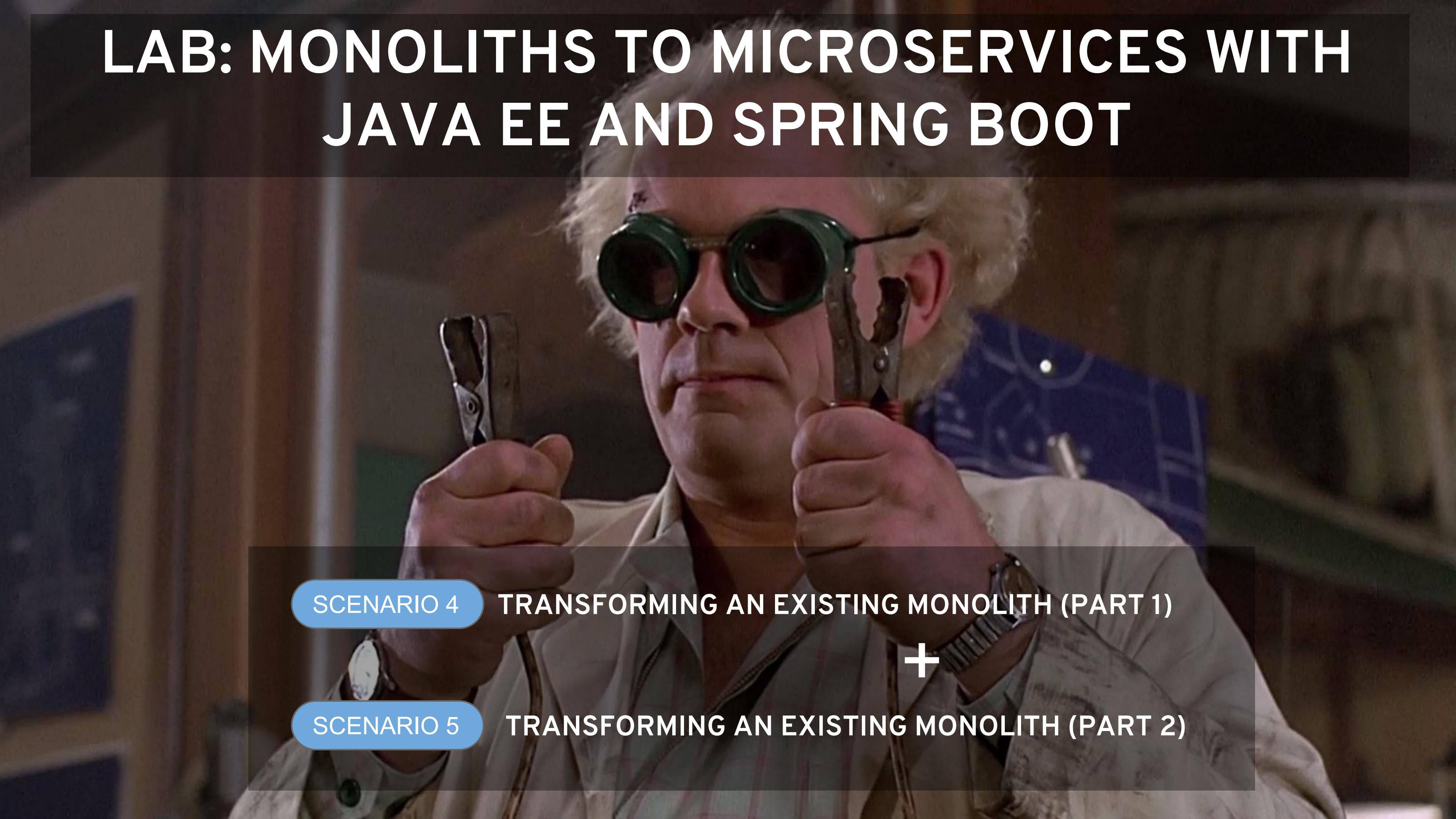
In this lab you will learn:

- 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)

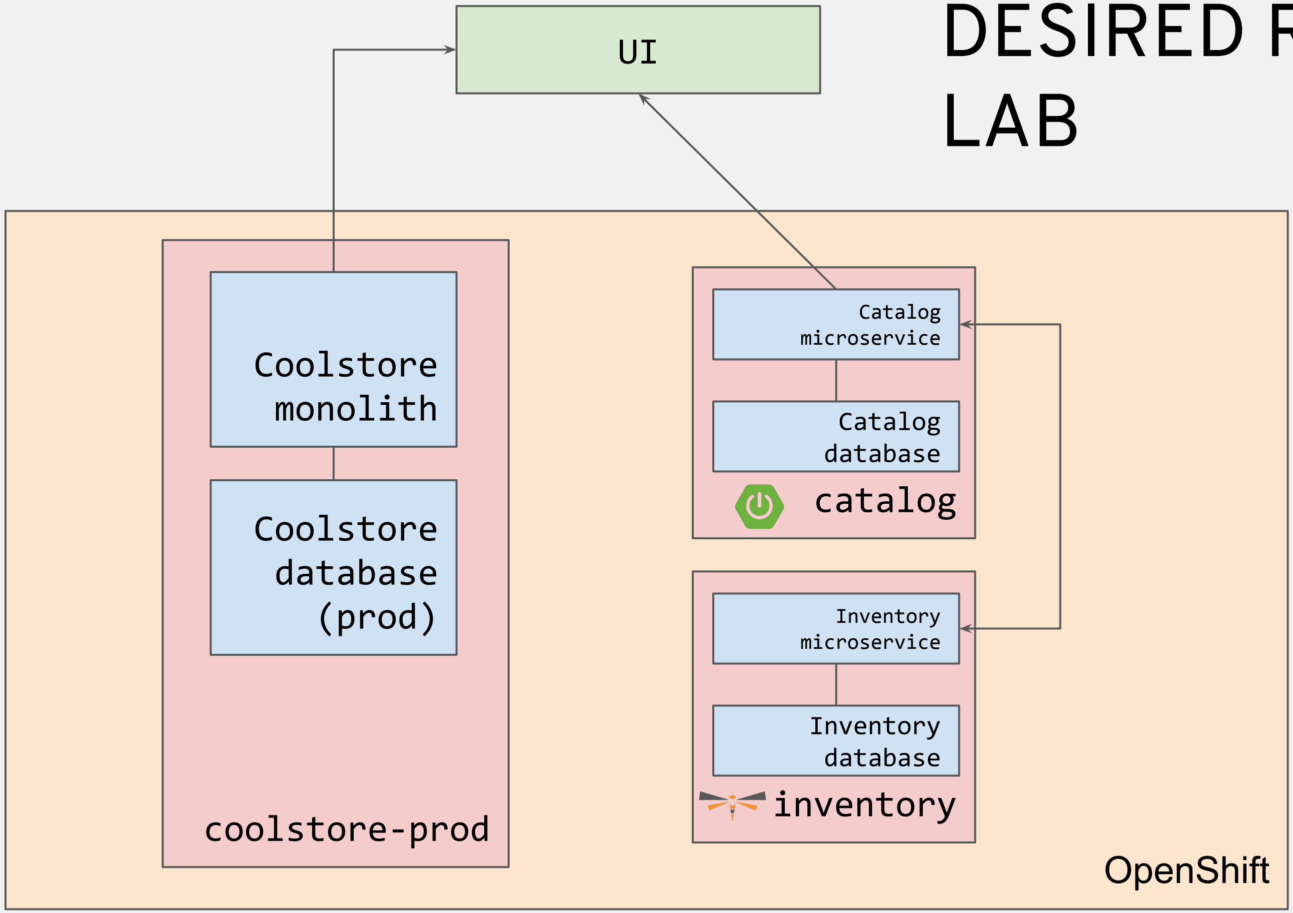
WRAP-UP AND DISCUSSION

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



THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos