redhat.

MONOLITHS TO MICROSERVICES: APP TRANSFORMATION

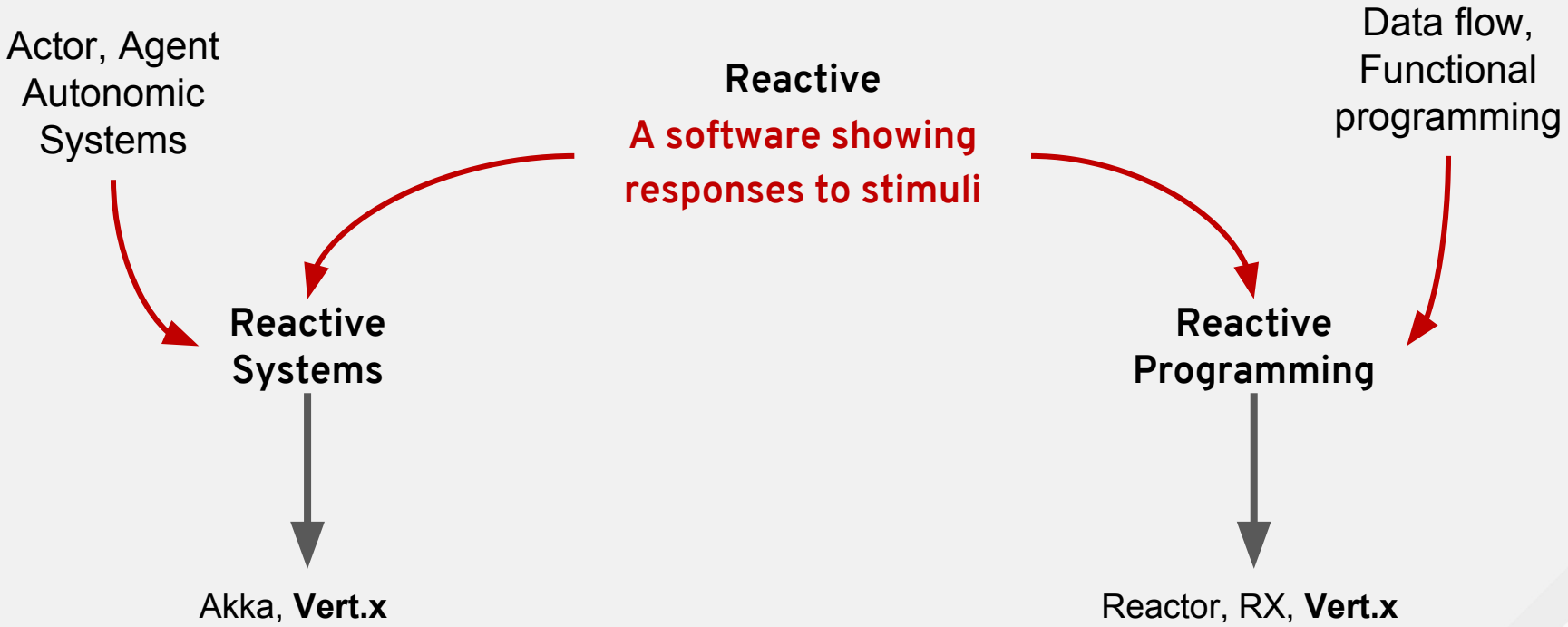
Hands-on Technical Workshop

Daniel Soffner
Andy Yuen
Tom Corcoran

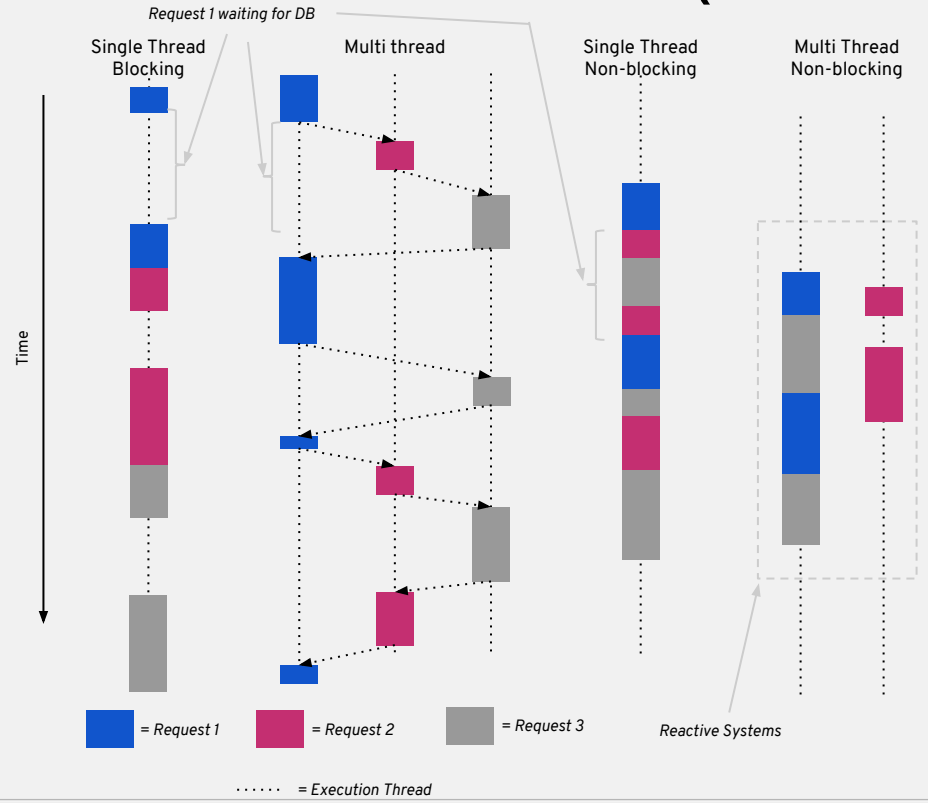
Senior Solution Architects
Red Hat Australia and New Zealand

REACTIVE MICROSERVICES

THE 2 FACES OF REACTIVE



EXECUTION MODEL (SINGLE CORE)



Single thread blocking

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

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

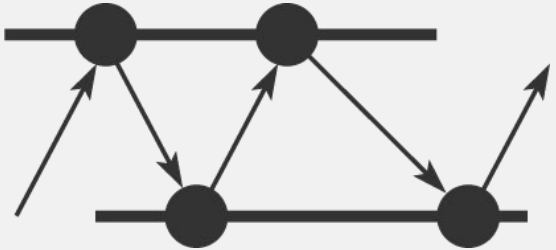
ECLIPSE VERT.X

VERT.X

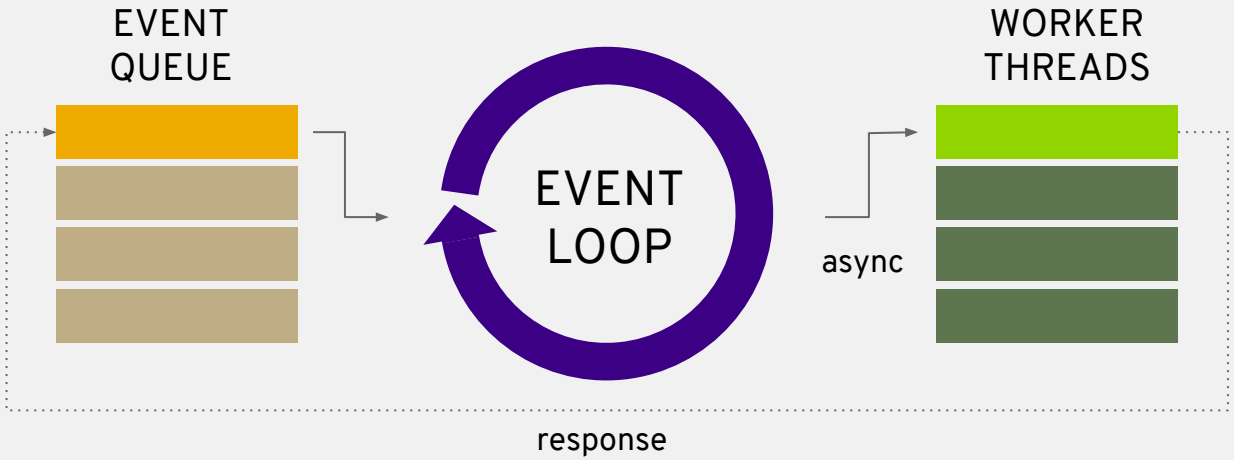
Vert.x is a toolkit to build distributed and reactive systems

- **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

- Vertx is NOT an application server
- Embeddable - just a jar inside the application
- Minimal dependencies
- Modular
  - Vert.x core
  - Vert.x extensions

EVENT LOOP

- Implementation of the Reactor Pattern
- Events are passed to handlers when they become available
- One single thread can handle large number of events
- Vert.x implements the Multi-Reactor Pattern
  - By default starts 2 event loops per core
  - Efficient usage of multi-core servers

DO NOT BLOCK THE EVENT LOOP

- While event loop is blocked, it can’t process new events
- Vert.x APIs are non blocking and won’t block the event loop
  - Example: asynchronous file operations
- Blocking application code should be handled asynchronously
- Blocking code executed on the worker thread pool

EXAMPLES OF BLOCKING CODE

- Thread.sleep()
- Waiting on a lock
- Waiting on a mutex or monitor (e.g. synchronized section)
- Doing a long lived database operation and waiting for a result
- Doing a complex calculation that takes some significant time.
- Spinning in a loop

BLOCKING CODE HANDLERS

- By default blocking handlers are ordered
- The next one won't be executed before the first one has completed if called from the same context (verticle instance)
- If your blocking handlers can be executed in parallel, specify ordered as false

```
vertx.executeBlocking(Handler<Future<T>> blockingCodeHandler,
        boolean ordered, Handler<AsyncResult<T>> resultHandler)
```

ASYNCRESULT AND FUTURE

- Example: vertx.executeBlocking method

```
<T> void executeBlocking(Handler<Future<T>> blockingCodeHandler,
        Handler<AsyncResult<T>> resultHandler);
```

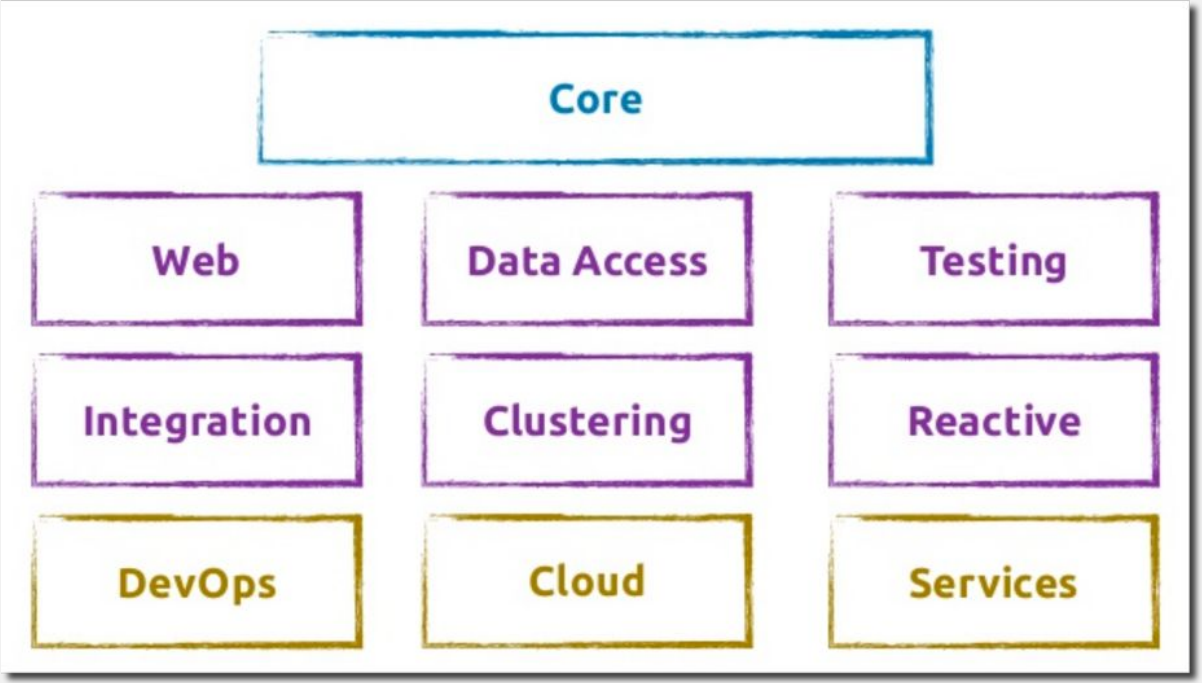
- AsyncResult
  - Encapsulates the result of an asynchronous operation.
  - The result can either have failed or succeeded.

```
if (asyncResult.succeeded()) {
        Object result = asyncResult.result();
} else {
        Throwable t = asyncResult.cause();
}
```

VERT.X VERTICLE

- Verticle = programmable unit within Vertx
- A Verticle is always executed on the same thread, called the Event Loop
  - Simple, actor-like deployment and concurrency model
- A single thread may execute several verticles
- An application would typically be composed of many verticle instances running in the same Vert.x instance at the same time.
- The different verticle instances communicate with each other by sending messages on the event bus.

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

VIDEO: REACTIVE MICROSERVICES WITH ECLIPSE VERT.X

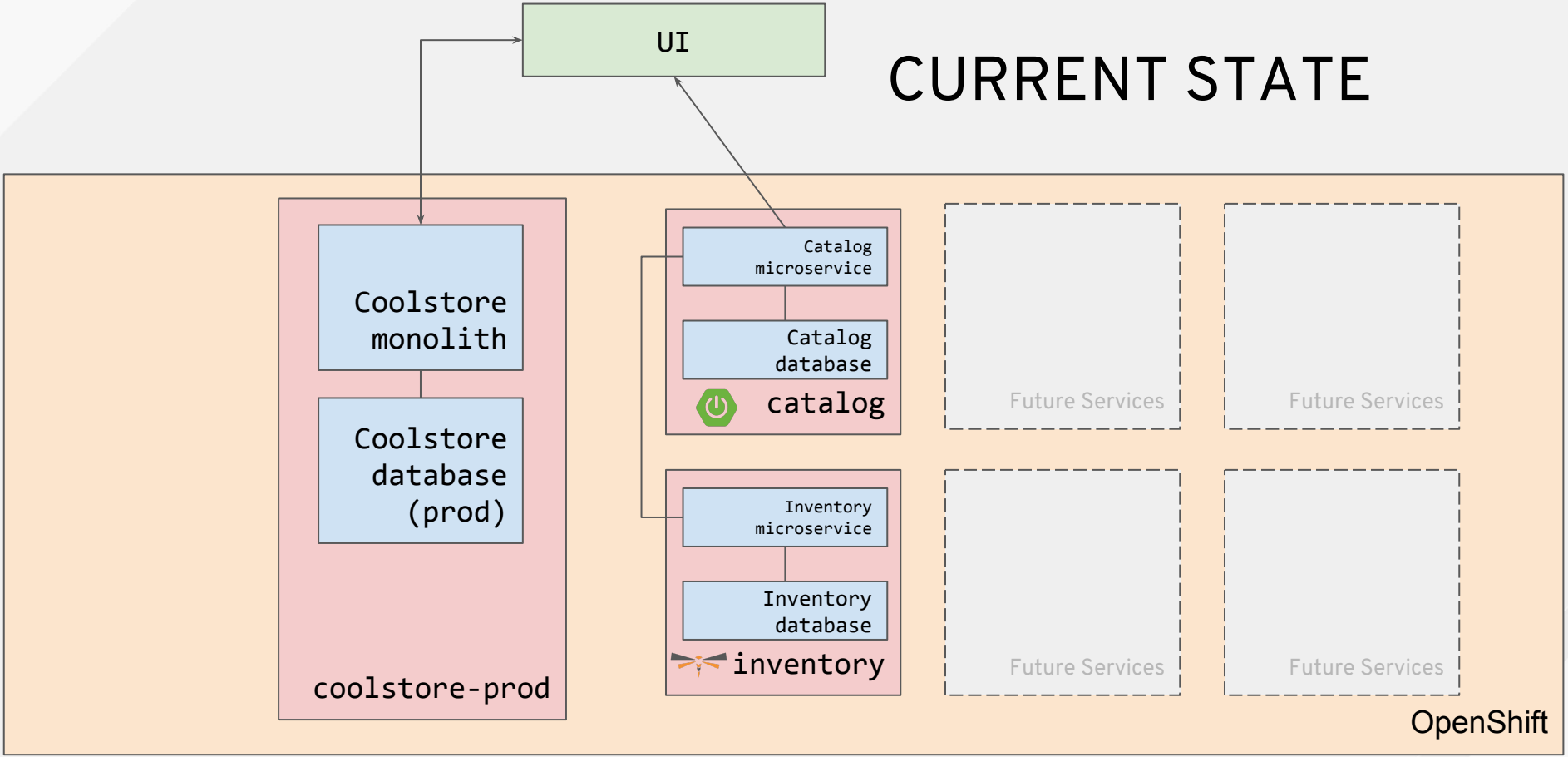
LAB: REACTIVE MICROSERVICES WITH ECLIPSE VERT.X

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

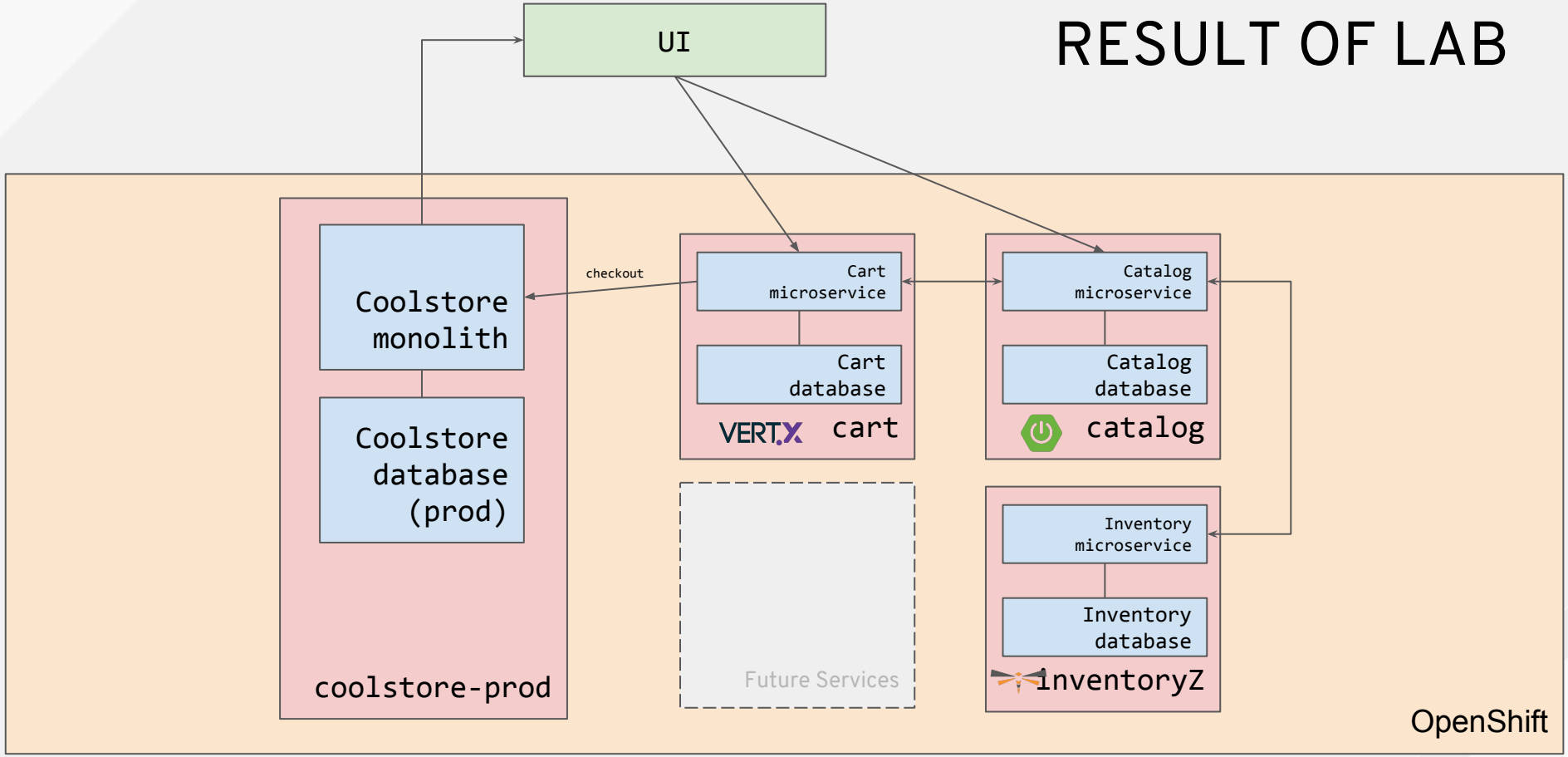
WRAP-UP AND DISCUSSION

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

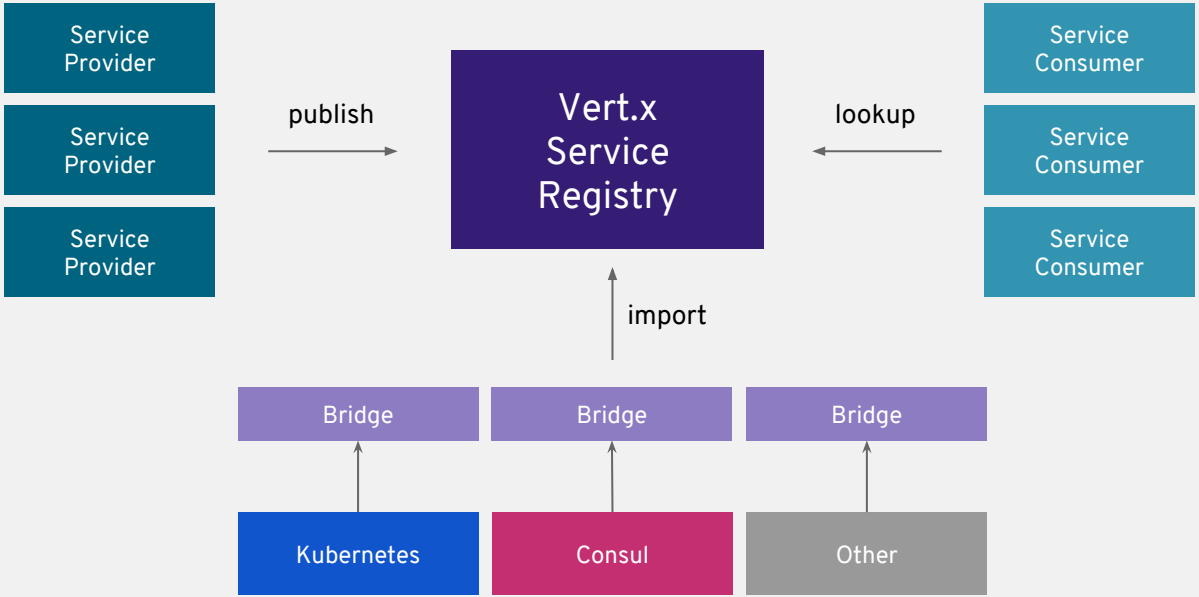
RESULT OF LAB



LAB VIDEO

ECLIPSE VERT.X OFFER MUCH MORE

SERVICE DISCOVERY



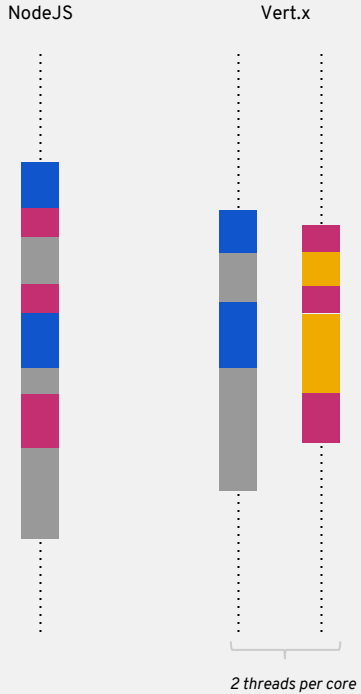
Vert.x vs NodeJS

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



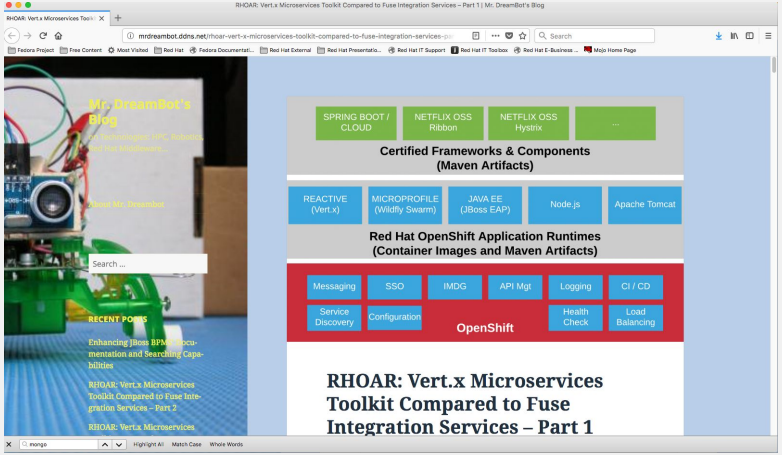
VERT.X ARTICLES ON MY BLOG: Mr. Dreambot's Blog

RHOAR: Vert.x Microservices Toolkit Compared to Fuse Integration Services – Part 1

- Using Dependency Injection with Google Guice
- Using MongoDB Async Client
- Writing JUnit tests
- Deploying to Openshift

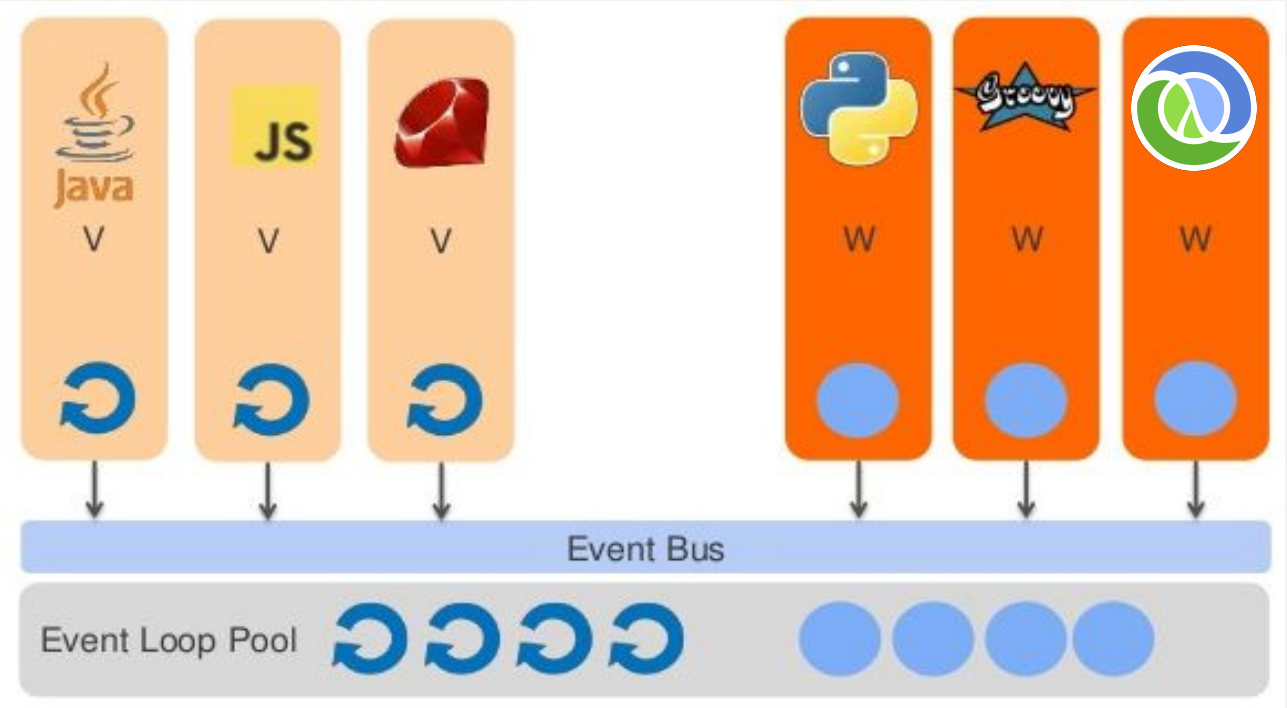
RHOAR: Vert.x Microservices Toolkit Compared to Fuse Integration Services – Part 2

- Using Vert.x EventBus
- Creating Service proxy via code generation to eliminate boilerplate code to use EventBus
- Writing JUnit tests
- Deploying to Openshift

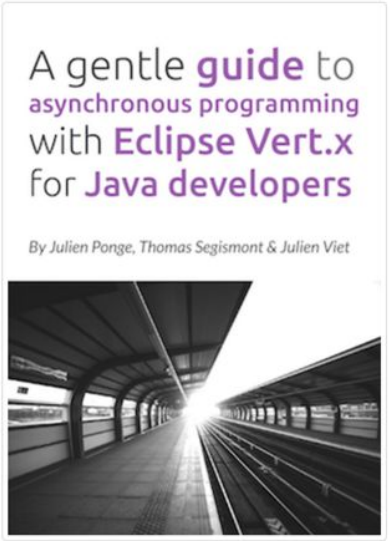


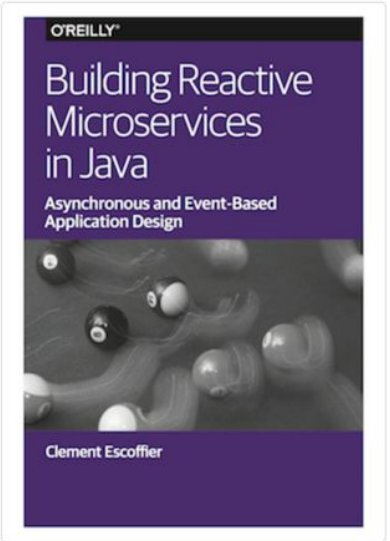
All source code available on Github

EVENT BUS



FREE E-BOOKS





http://vertx.io/docs/

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