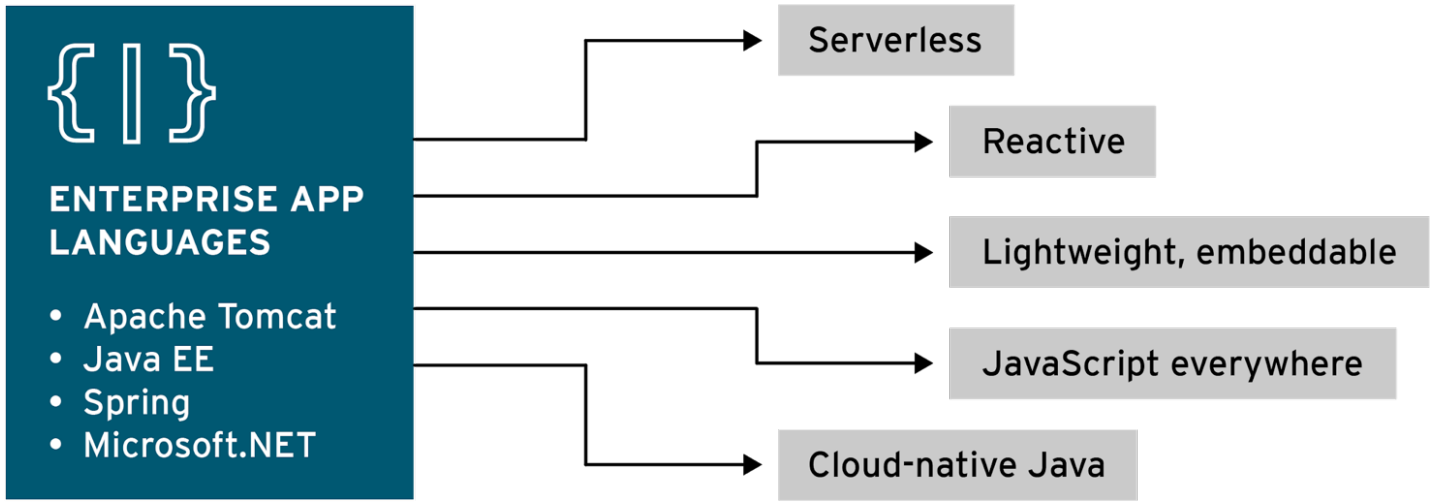

The Future of Enterprise Application Development in the Cloud – Collaborative Research Opportunities
Mark Little, VP, Red Hat

# Aims of this presentation

- To identify potential research opportunities between Red Hat and Boston University
  - In the context of Red Hat Enterprise Middleware
- Not all research areas of interest to Red Hat but we have to start somewhere
- Follow up interactions between our teams asap
- Introduce Boston University to Red Hat communities
  - Much of this work is going to be practice and experience driven
  - Many of these activities are going to be high profile, with interactions across many vendors and communities

# DEVELOPERS DEMAND MORE OPTIONS

## ENTERPRISES EXPAND USE OF LANGUAGES, FRAMEWORKS, & RUNTIMES



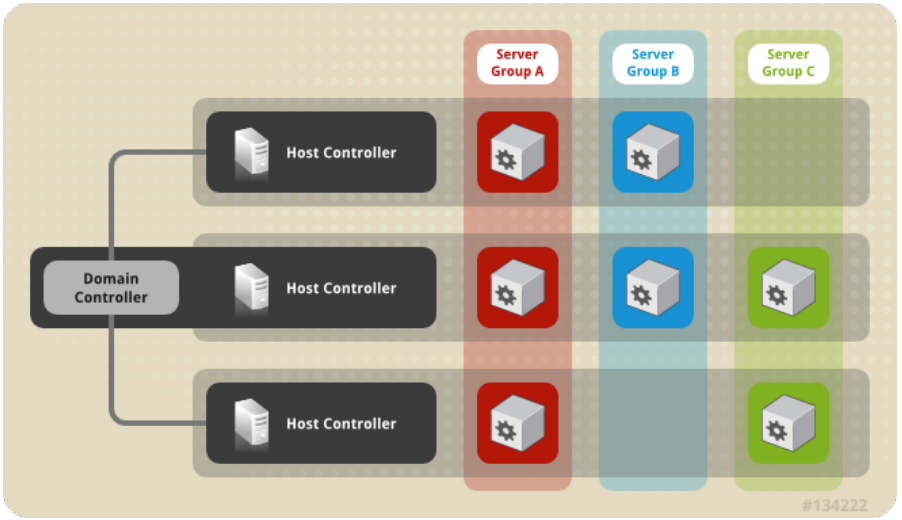
# Traditional Enterprise Applications


## Monolithic Applications

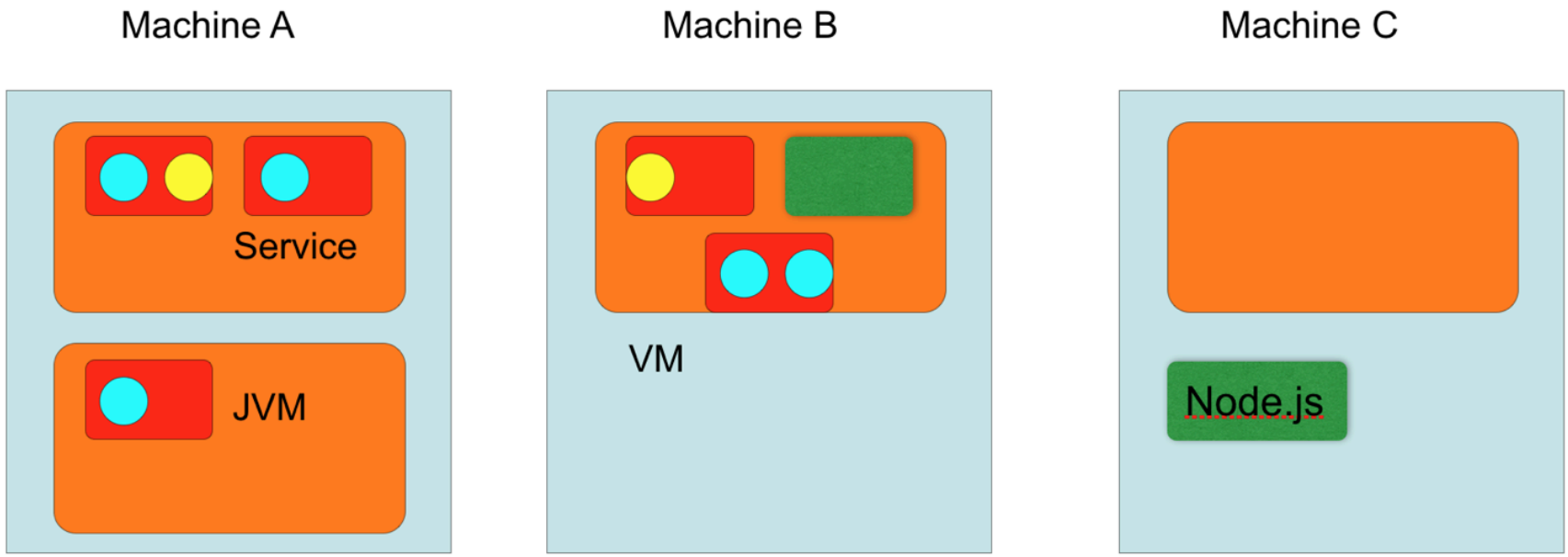
- Java Only
- Focus on business logic
- Appserver “services”. Ex:
  - Configuration
  - Resource abstraction
  - Strict Dev/Ops separation

## Provided Infrastructure

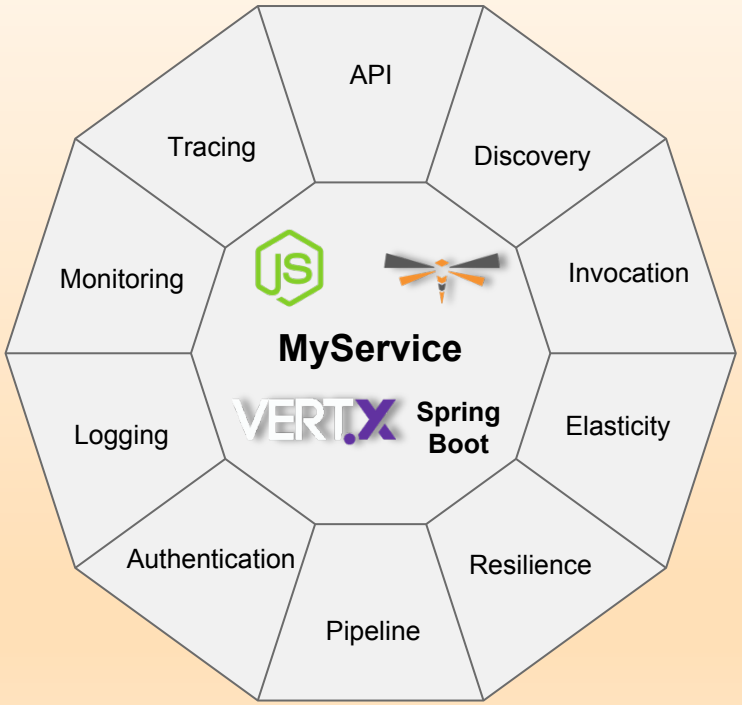
- Centralized logs
- AppServer lifecycle mgmt
- Application lifecycle mgmt



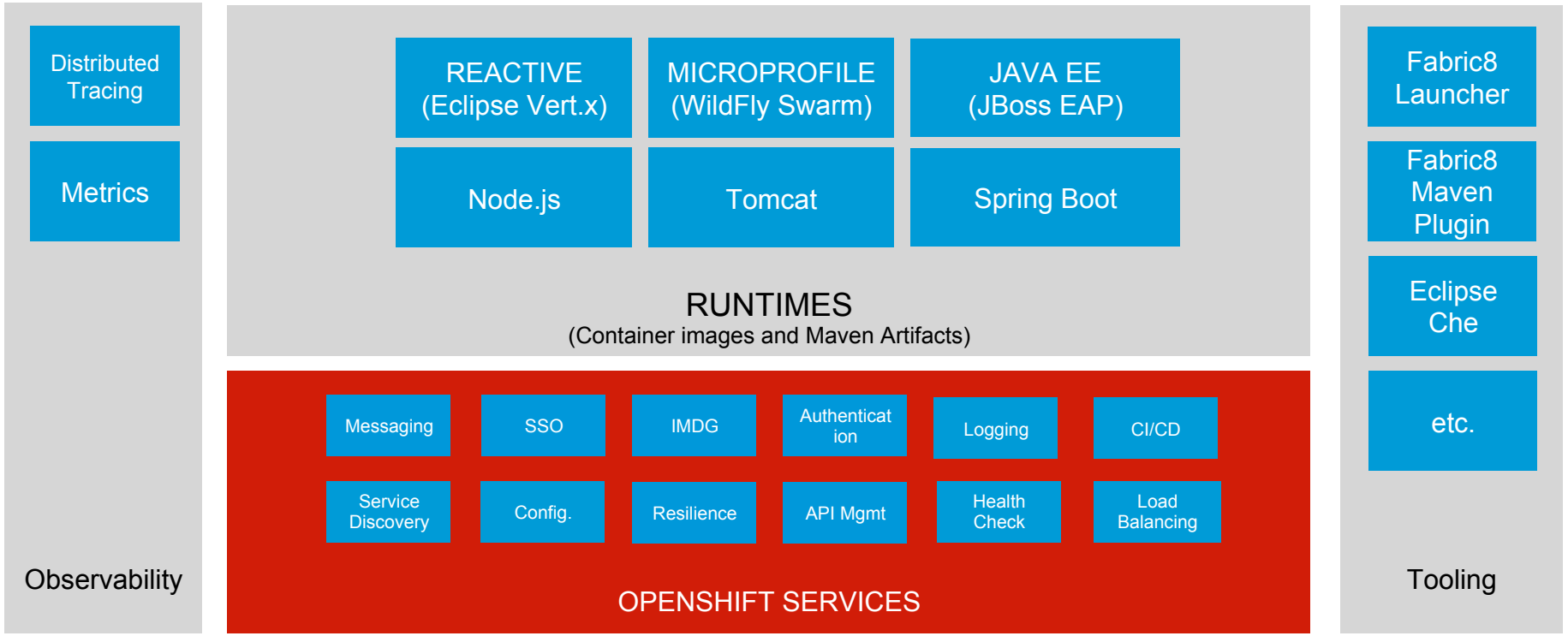
# Decomposing of services, SOA and beyond



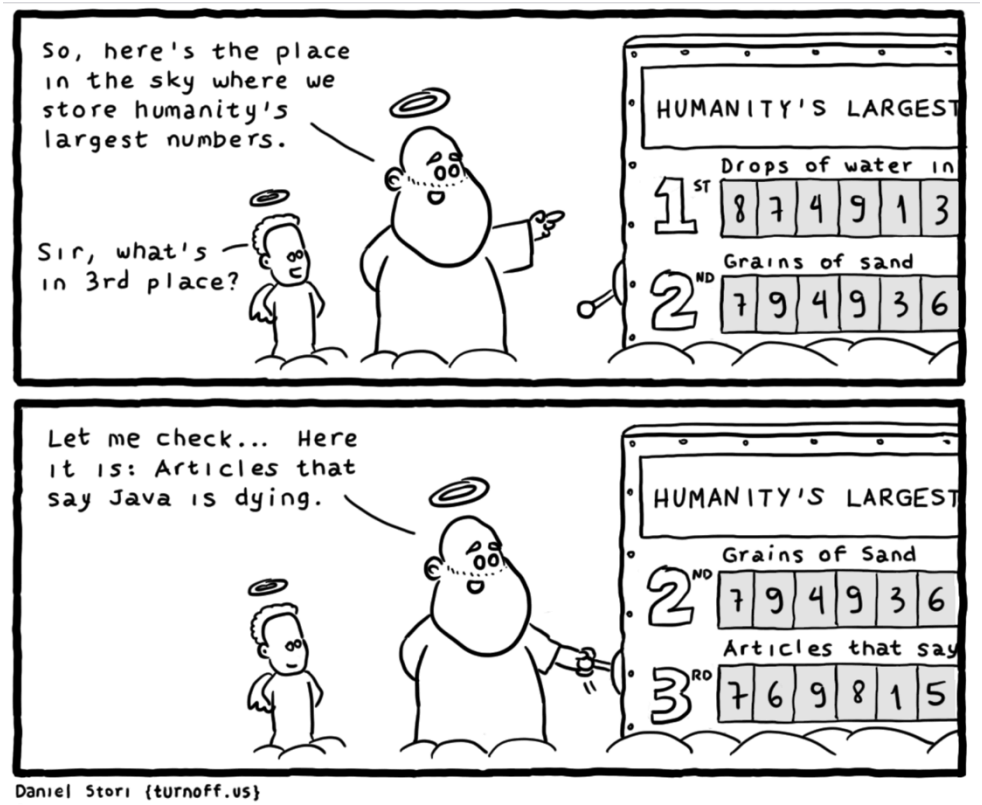
# Cloud-native “application server”: what, why & how?



# RHOAR – background context for research



https://dzone.com/articles/big-numbers-comic





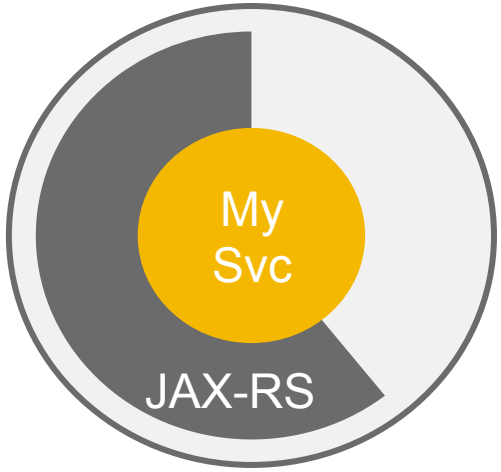
- Defines **open source** Java **microservices** specifications
- Industry Collaboration - Red Hat, IBM, Payara, Tomitribe, London Java Community, SouJava, Oracle, Hazelcast, Fujitsu, SmartBear...
- **WildFly Swarm** is **Red Hat's** implementation
- Minimum footprint for Enterprise Java cloud-native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

| | | | | | |
|---|---|---|---|---|---|
| JSON-P 1.0 | Common Annotations | JWT Propagation | Config | OpenAPI | REST Client |
| CDI 1.2 | JAX-RS 2.0 | Fault Tolerance | Metrics | Open Tracing | Health Check |



## Java EE microservices

- Leverage Java EE expertise
- Open Standard
- Microservices focus
- Optimized for OpenShift
- Super Lightweight
- uber-jar and war support




```
$ java -jar my_microservice.jar
                or
$ java -jar custom-runtime.jar myapp.war
```


Strategic Members

FUJITSU IBM ORACLE payara redhat Tomitribe

Participating Members

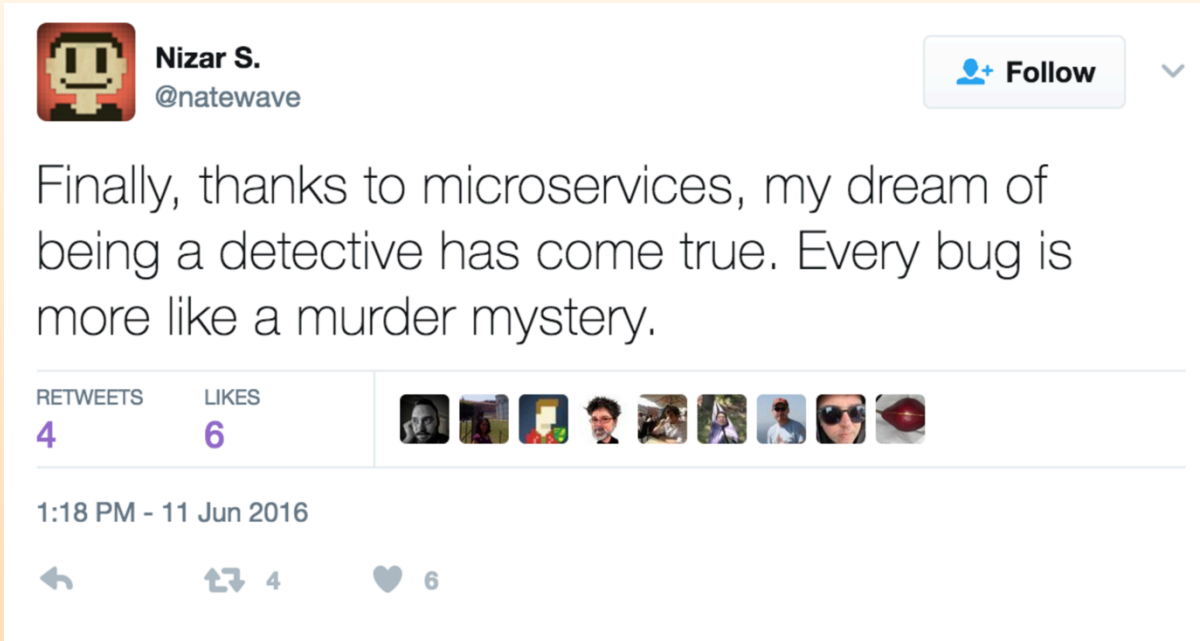
DOCDOKU Genuitec IncQueryLabs Lightbend Microsoft Pivotal SAP

vaadin webtide

https://jakarta.ee/

# This is not the Java EE you are looking for ...

# Microservices

# Observability

Jaeger
Distributed
Tracing

Prometheus
Metrics

elastic

fluentd

kibana

EFK Logging

# Asychronous, event-driven programming

- Asynchronous APIs allow for more efficient use of process/thread
  - But make programming harder
- Asynchronous communication has significant impact on fault tolerance
- However, we have yet to make parallel programming easy
- Failure detection impossibility
  - Fischer, Lynch, Patterson (FLP) Result
- Synchronous programming is still the default in most organisations and standards
  - With multi-threading, of course

# What Can Reactive Streams Offer EE4J?

Interested in what Reactive Streams can bring but aren't sure where they'd fit into your enterprise ecosystem? Here are some tips for EE4J projects.

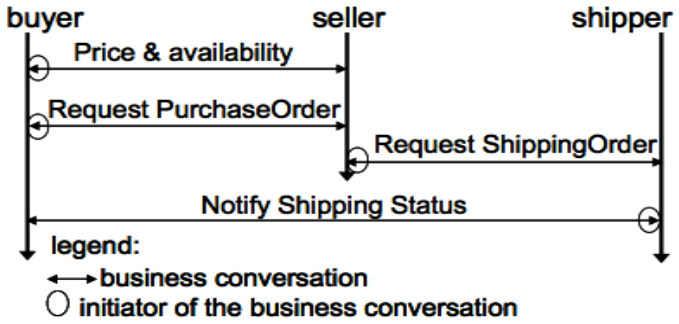
by James Roper MVB · Feb. 21, 18 · Java Zone · Tutorial

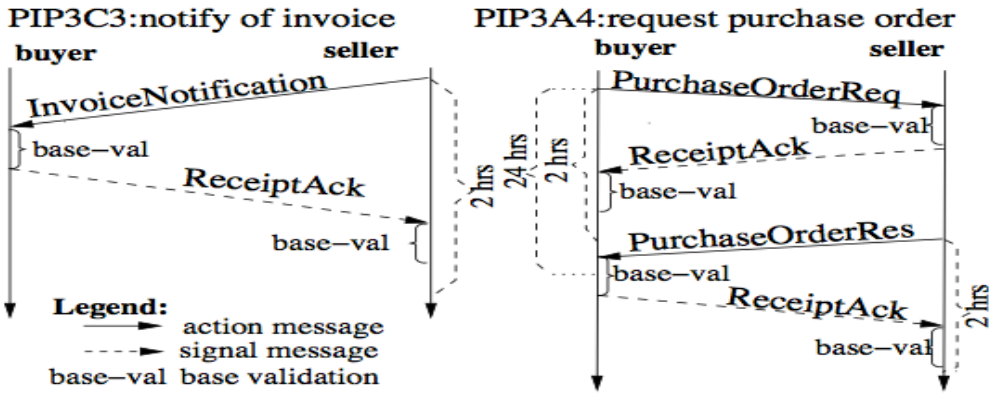


Take 60 minutes to understand the Power of the Actor Model with "Designing Reactive Systems: The Role Of Actors In Distributed Architecture". Brought to you in partnership with Lightbend.

In my current role at Lightbend, I'm investigating and pursuing opportunities where Reactive Streams can make the lives of EE4J (the new Java EE) developers better. In this blog post, I'm going to share some of the ideas that we've had for Reactive Streams in EE4J, and how these ideas will benefit developers.

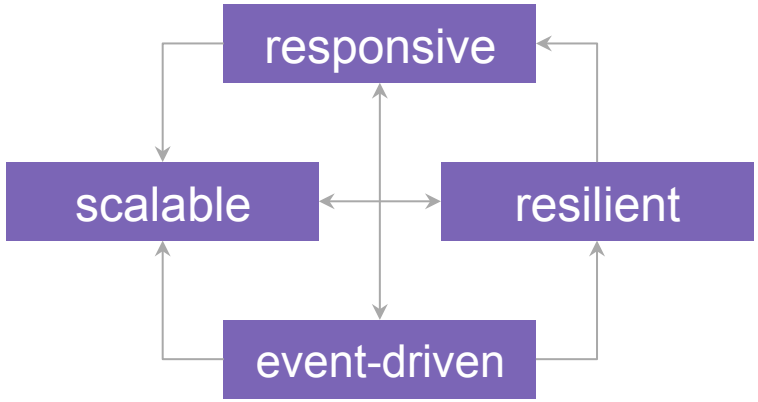
# Message-based interactions

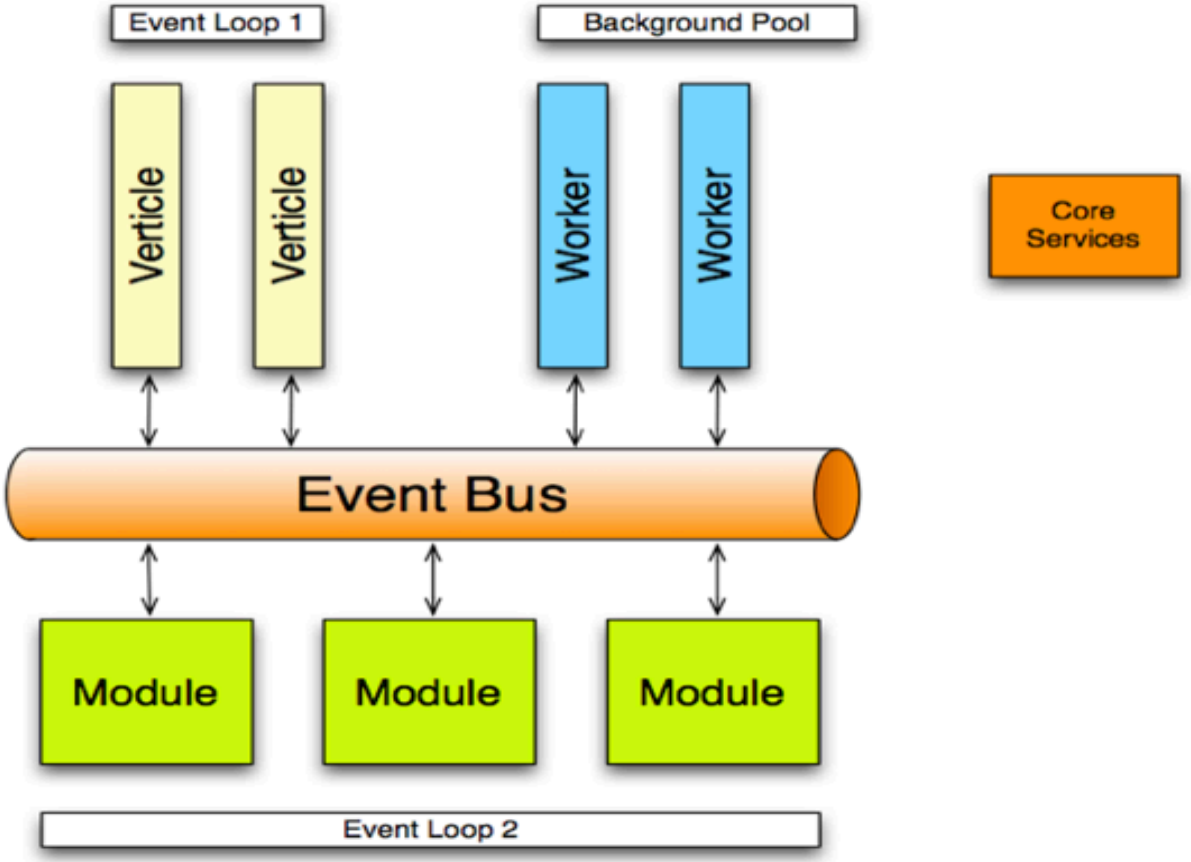


# VERT.X

- Path for Reactive Microservices for JVM
- Event Driven Non-Blocking I/O
- Ideal for High Concurrency and Low Latency Services
- Lightweight Messaging
- OpenShift / Middleware Integration
- 2014 JAX Innovations Award Winner



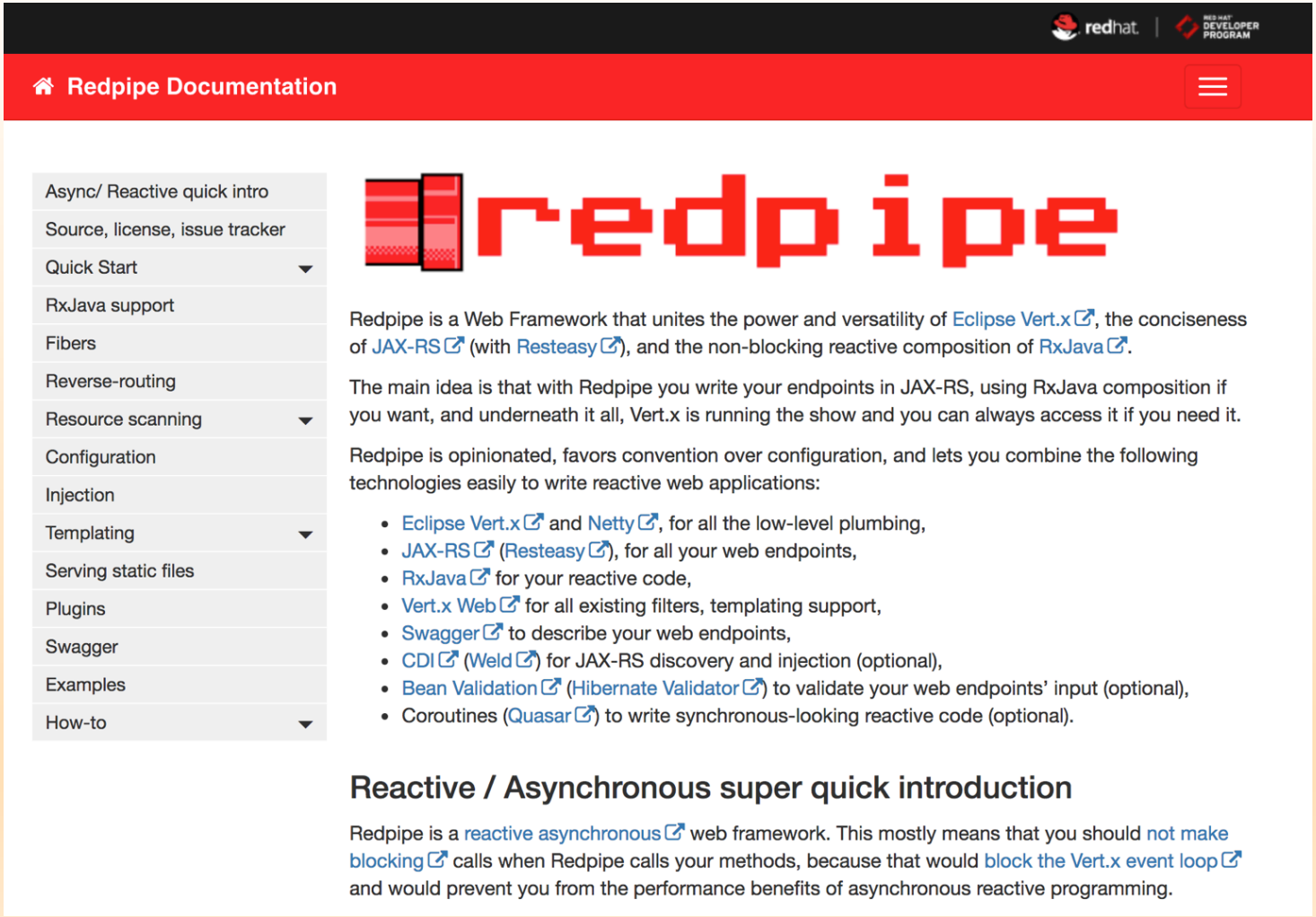

Event Loop 1
Background Pool
Verticle
Verticle
Worker
Worker
Core Services
Event Bus
Module
Module
Module
Event Loop 2


- Eclipse Vert.x 3.5.3
  - vertx-rx-java2 - RxJava 2, Improved reactive support
  - MongoDB / Mongo Client
  - vert.x-sockjs-service-proxy
  - Event Bus on OpenShift
  - vertx-proton - AMQ Client
  - gRPC
  - MQTT module (client and server)
  - Kafka client, Prometheus client, JUnit 5, and HashiCorp Vault Tech Preview

- Improved reactive support
- Improved enterprise and device connectivity
- Improved OpenShift Support

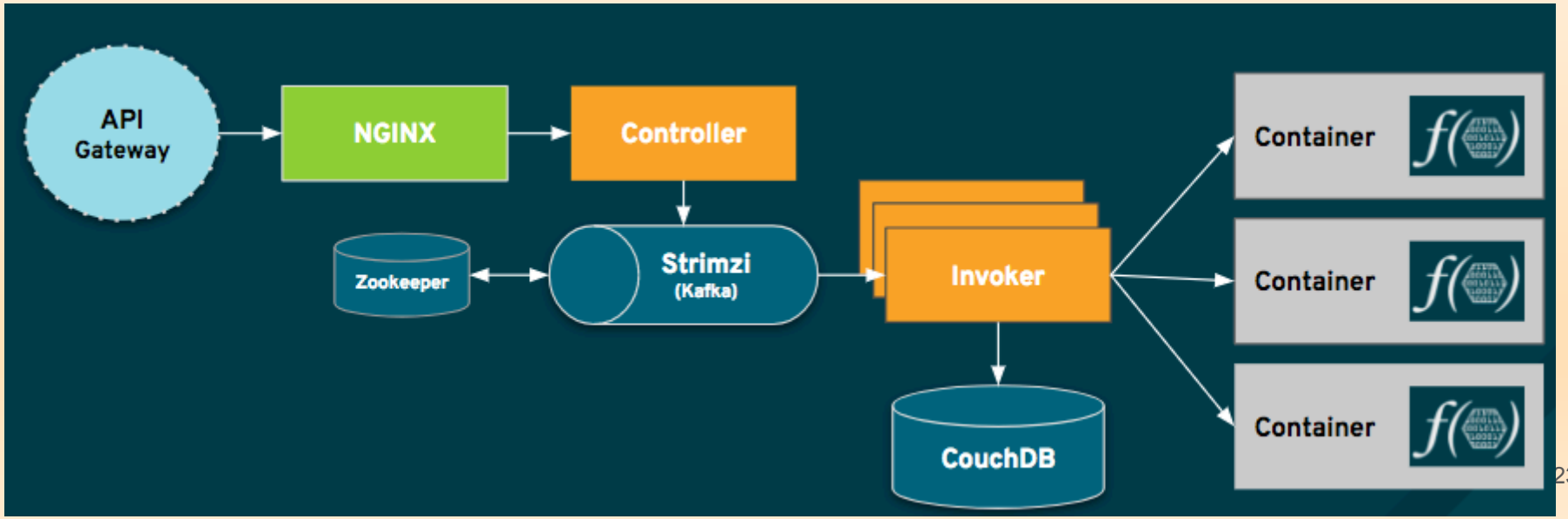
# Project Loom and Fibres

Project Loom's mission is to make it easier to write, debug, profile and maintain concurrent applications meeting today's requirements. Threads, provided by Java from its first day, are a natural and convenient concurrency construct (putting aside the separate question of communication among threads) which is being supplanted by less convenient abstractions because their current implementation as OS kernel threads is insufficient for meeting modern demands, and wasteful in computing resources that are particularly valuable in the cloud. Project Loom will introduce fibers as lightweight, efficient threads managed by the Java Virtual Machine, that let developers use the same simple abstraction but with better performance and lower footprint. We want to make concurrency simple(r) again! A fiber is made of two components — a continuation and a scheduler. As Java already has an excellent scheduler in the form of `ForkJoinPool`, fibers will be implemented by adding continuations to the JVM.


redhat.
DEVELOPER PROGRAM
Redpipe Documentation
Async/ Reactive quick intro
Source, license, issue tracker
Quick Start
RxJava support
Fibers
Reverse-routing
Resource scanning
Configuration
Injection
Templating
Serving static files
Plugins
Swagger
Examples
How-to
redpipe
Redpipe is a Web Framework that unites the power and versatility of Eclipse Vert.x, the conciseness of JAX-RS (with Resteasy), and the non-blocking reactive composition of RxJava.
The main idea is that with Redpipe you write your endpoints in JAX-RS, using RxJava composition if you want, and underneath it all, Vert.x is running the show and you can always access it if you need it.
Redpipe is opinionated, favors convention over configuration, and lets you combine the following technologies easily to write reactive web applications:
Eclipse Vert.x and Netty, for all the low-level plumbing,
JAX-RS (Resteasy), for all your web endpoints,
RxJava for your reactive code,
Vert.x Web for all existing filters, templating support,
Swagger to describe your web endpoints,
CDI (Weld) for JAX-RS discovery and injection (optional),
Bean Validation (Hibernate Validator) to validate your web endpoints' input (optional),
Coroutines (Quasar) to write synchronous-looking reactive code (optional).
Reactive / Asynchronous super quick introduction
Redpipe is a reactive asynchronous web framework. This mostly means that you should not make blocking calls when Redpipe calls your methods, because that would block the Vert.x event loop and would prevent you from the performance benefits of asynchronous reactive programming.

# Apache OpenWhisk

- Complete Serverless solution – event driven!
- Incubating project under Apache Software Foundation
- Started by IBM but with Adobe and Red Hat as contributors
  - But now there's also knative



# Fault tolerance is a key area

- Machines and software fail
- Fundamental universal law (entropy increases)
- Things get better with each generation, but still statistically significant
- Failures of centralized systems difficult to handle
- Failures of distributed systems are much more difficult

# Fault tolerance techniques

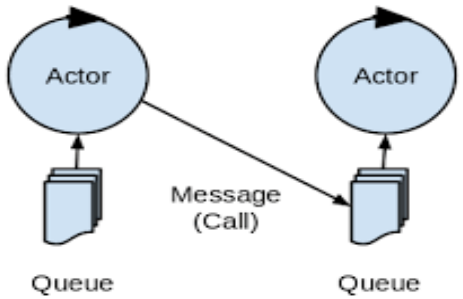
- Replication of resources
  - Increase availability
  - Probability is that a critical number of resources remain operational
  - “Guarantee” forward progress
  - Tolerate programmer errors by heterogeneous implementations
- Spheres of control
  - “Guarantee” no partial completion of work in the presence of failures
- Often a duality
  - “Understanding the Role of Atomic Transactions and Group Communications in Implementing Persistent Replicated”, Proceedings of the 8th International Workshop on Persistent Object Systems, California, USA,1998

# Software Transactional Memory

- Hardware Transactional Memory around since the 1980's
  - An alternative to lock-based synchronization
- Software Transactional Memory (STM) proposed in 1995
  - Still an active area of research
- STM is about ease of use and reliability
  - Access shared state, either for reading of writing, within atomic blocks
  - All code inside an atomic block executes as if there were no other threads
  - Some implementations can be lock free (optimistic vs pessimistic, timestamp)

# The Actor Based Programming Model

- The Actor Based Programming Model
- Actors and CSP have been around for decades
  - CSP from Hoare, 1985
  - Actor model from Hewitt et al, 1973
- But popular ways to model primitives for concurrent computations
- Distributed computations communicate via message passing
  - No shared state



# Transactions and Actors and async … oh my!

- A stateful actor may go through multiple state transactions upon receipt of a message
  - Actors share state through message passing
- Computational failures may occur
- Hardware and software failures may occur
- Consistency of state important
- Composition of actors
- The combination of STM and Actors is fairly natural
  - But complexity still arises when combined in microservices and the cloud

# Transaction models for cloud-native

- ACID transactions implicitly assume
  - Closely coupled environment
  - All entities involved in a transaction span a LAN, for example
- Short-duration activities
  - Must be able to cope with resources being locked for periods
- Therefore, do not work well in
  - Loosely coupled environments
  - Long duration activities

# What characteristics are right?

- Need to be able to relax the strict ACID properties
  - Need to put control of some into hands of service
- Is consistency (or consensus) important?
- May need a notion of a central coordinator
  - But may not!
  - Or something with a fuzzy idea of what's going on
- "A comparison of Web services transaction protocols", IBM Developer Works, 2003.
- Relaxing atomicity, isolation and consistency
- Yes, CAP but …
  - https://www.cl.cam.ac.uk/research/dtg/www/files/publications/public/mk428/cap-critique.pdf

# Heisenberg's Uncertainty Principle

- Cannot accurately measure both position and momentum of sub-atomic particles
  - Can know one with certainty, but not the other
  - Non-deterministic measurements
- Large-scale/loosely-coupled transactional applications suffer the same effect
  - Can know that all services will eventually see same state, just not when
  - Or at known time can determine state within model/application specific degree of uncertainty
- Or another way of thinking about it …
  - No such thing as simultaneity in data space as there isn't in space-time
    - *"Data on the Outside vs. Data on the Inside", by Pat Helland*

# No global consensus

- Split transactions into domains of consistency
  - Strong consistency within domains
  - Some level of (known) consistency between domains
    - See *“A method for combining replication and caching”, Proceedings of International Workshop on Reliable Middleware Systems, October 1999*.
    - *OASIS WS-BusinessProcess specification*, part of OASIS WS-CAF, 2003.
  - Resolve inconsistencies at the business level
    - Don’t try and run consensus protocols between domains
- Consistency related to isolation
  - Put into the control of the service and application developers

Insights and news on Red Hat developer tools, platforms and more

ALL MICROSERVICES CONTAINERS RHEL CLOUD DEVOPS LANGUAGES/COMPILERS

# Transactions for Microservices? Really?

*By tom jenkinson September 25, 2017*

(No Ratings Yet)


At the upcoming JavaOne 2017, which is being held at the Moscone Center in San Francisco, CA during October 1-5, we will be hosting a session on transactions for microservices.

LOGIN TO ADMIN YOUR BLOG

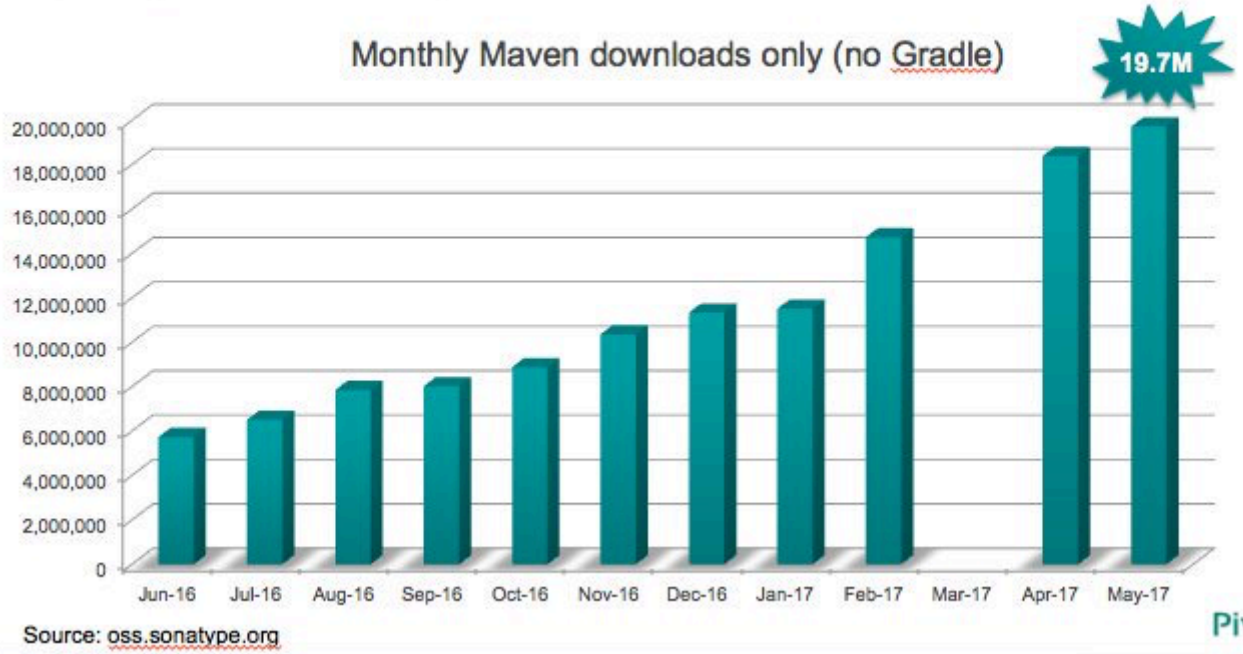
## RECENT POSTS

- Red Hat OpenShift Container Platform Load Testing Tips
- March 2018 ISO C++ Meeting Trip Report (Core Language)
- Some Rest with Vert.x
- Analyzing Changes to the Binary Interface Between the Linux Kernel and its Modules
- Integrating Intercede RapID with Red Hat Mobile and OpenShift

# AI and ML – intelligent applications

- 20 years ago developers needed to understand a lot of RDBMS to use them efficiently
- 16 years ago Hibernate came on the scene and changed things, democratising data
- JPA 1.0 released in 2006
- 2002 Spring Framework released
  - Re-packaged a lot of existing implementations and standards
  - Made J2EE more easily accessible to the average developer
- 2014 Spring Boot 1.0 released
  - Prescriptive approach
  - 20 million downloads from maven each month

# Spring Boot Adoption




Source: oss.sonatype.org

Pivotal

# “Democratize” Data+Analytics/ML

- There will be more developers looking to develop “intelligent apps” than data scientists who can help them
  - Enable the developer to become “just enough” of a data scientist to build apps
- Consider whether we can even isolate Spark as just an implementation detail
- Drive adoption through upstream communities
  - WildFly Swarm, Vert.x, Node.js, Spring Boot

# Conclusions and next steps

- There are a number of hopefully relevant research areas
- All can have significant impact on open source developer communities
- All variable term activities
- Identify those of interest by this team and designate Red Hat contact
  - Define specific work item(s)
  - Face-to-face meetings may help
- Success!
- Systems Research Challenges Workshop 2018
  - http://sysws.org.uk/workshop/2018/cfp/