Stay Productive While Slicing Up the Monolith

Markus Eisele

Lightbend

JavaOne

DEVOXX the java community conference

JavaZone

DEVNATION

25 JAHRE OOP 2016

DEVNEXUS

DEVCONF.CZ DEFINE FUTURE

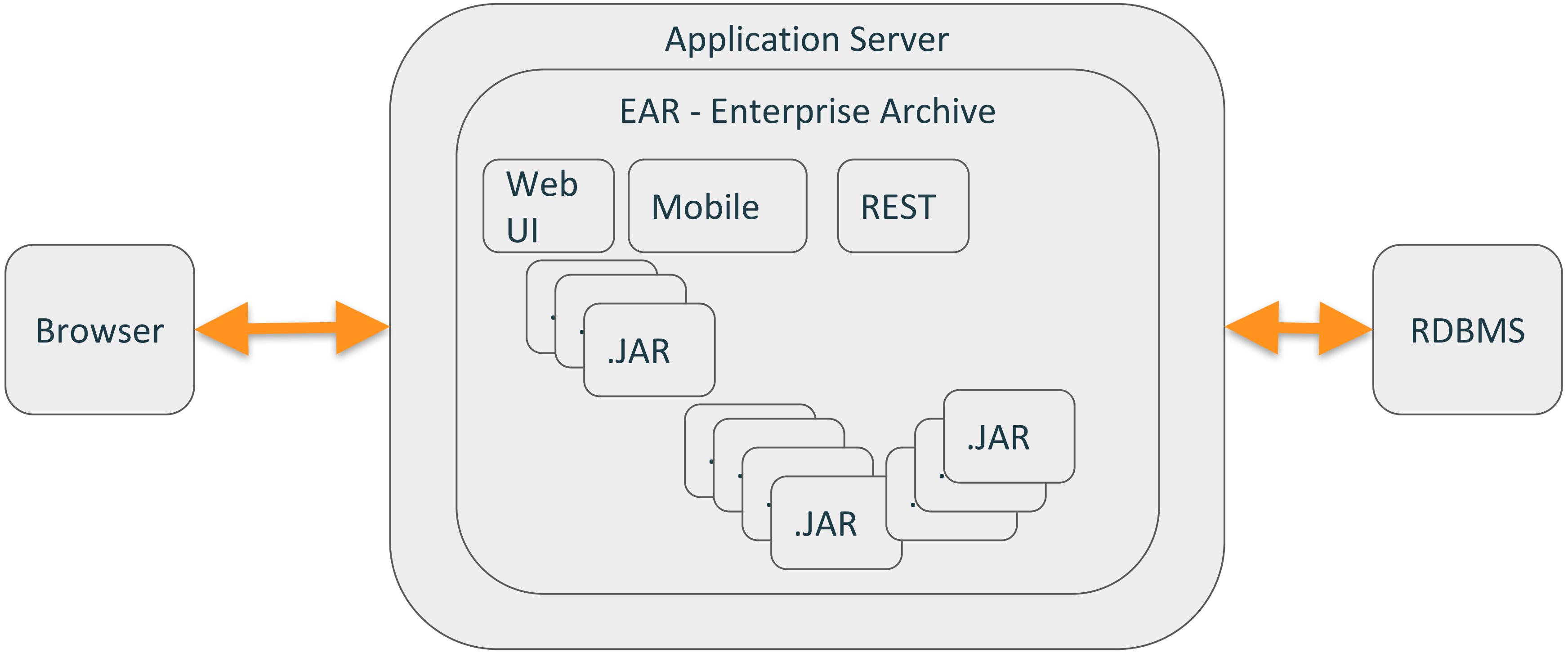
W-JAX

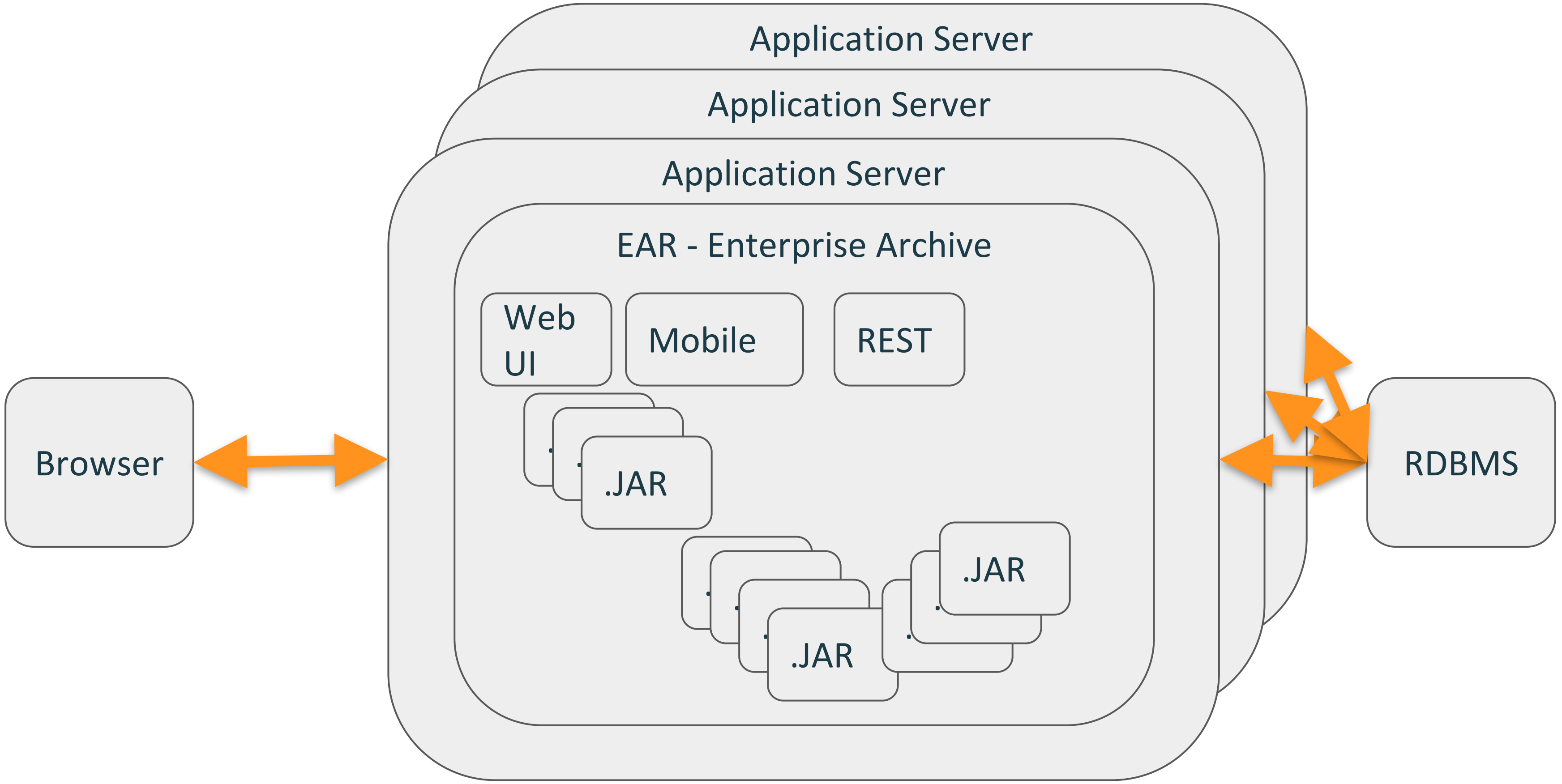
vJUGgernaut Award

Java Champions

JavaLand

Classical Architectures?

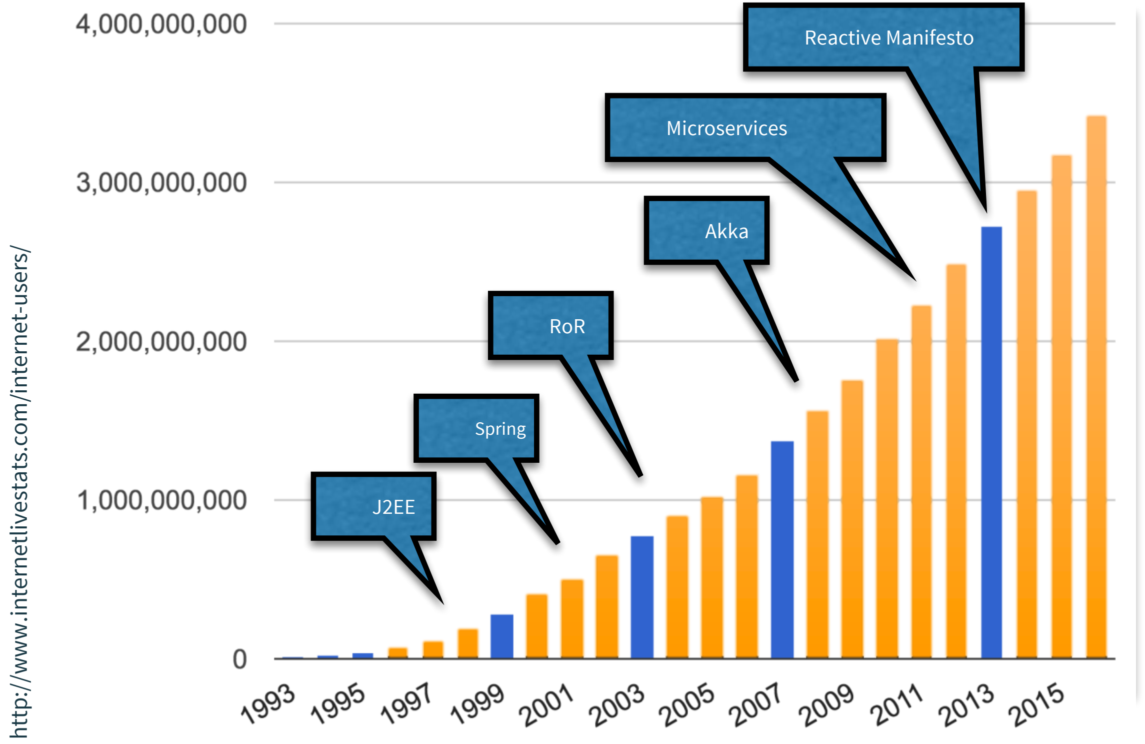

Application Server
EAR - Enterprise Archive
Web UI
Mobile
REST
.JAR
.JAR
.JAR
Browser
RDBMS


Application Server
Application Server
Application Server
EAR - Enterprise Archive
Web UI
Mobile
REST
.JAR
.JAR
.JAR
Browser
RDBMS

LL: Building and Scaling Monoliths

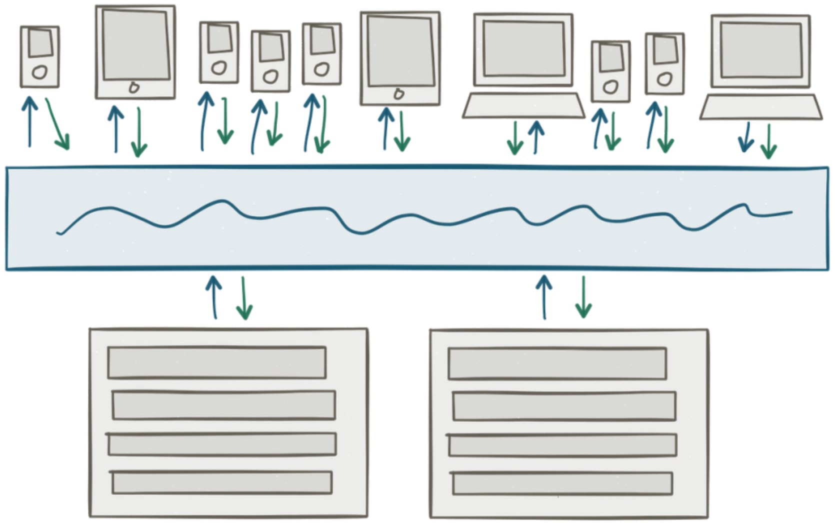
- Monolithic application – everything is package into a single .ear
- Reuse primarily by sharing .jars
- A “big” push to production once or twice a year
- Single database schema for the entire application
- >= 500k loc
- >= Heavyweight Infrastructure
- Thousands of Testcases
- Barely New Testcases
- >= 20 Team Member
- The single .ear requiring a multi-month test cycle /
- Huge bug and feature databases
- User Acceptance Undefined
- Technical Design Approach
- Barely Business Components or Domains
- Requiring multiple team involvement & significant oversight
- Technical Dept
- Outdated Runtimes (Licenses, Complex updates)
- Grown applications

More users



Lightbend

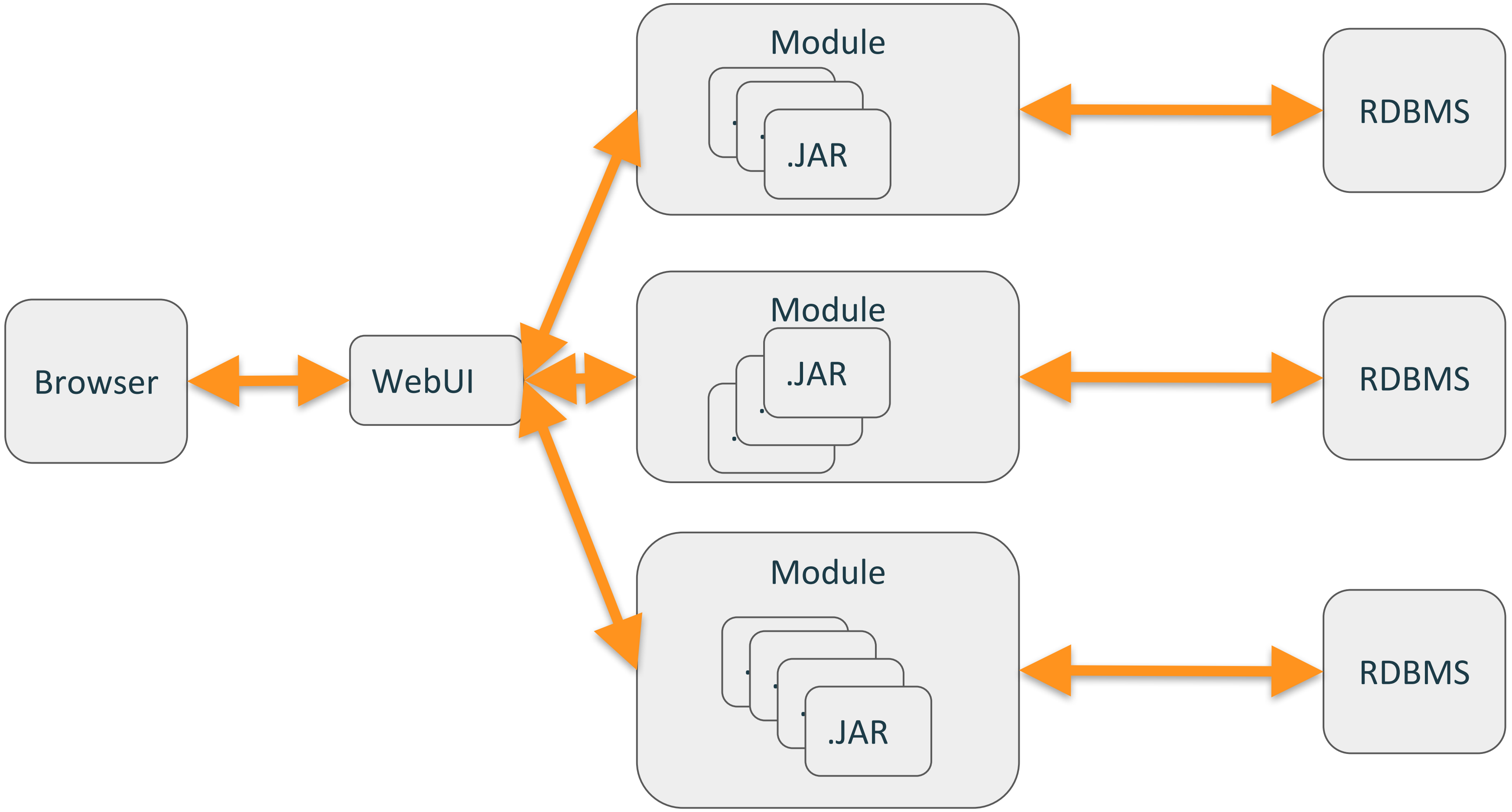
New requirements

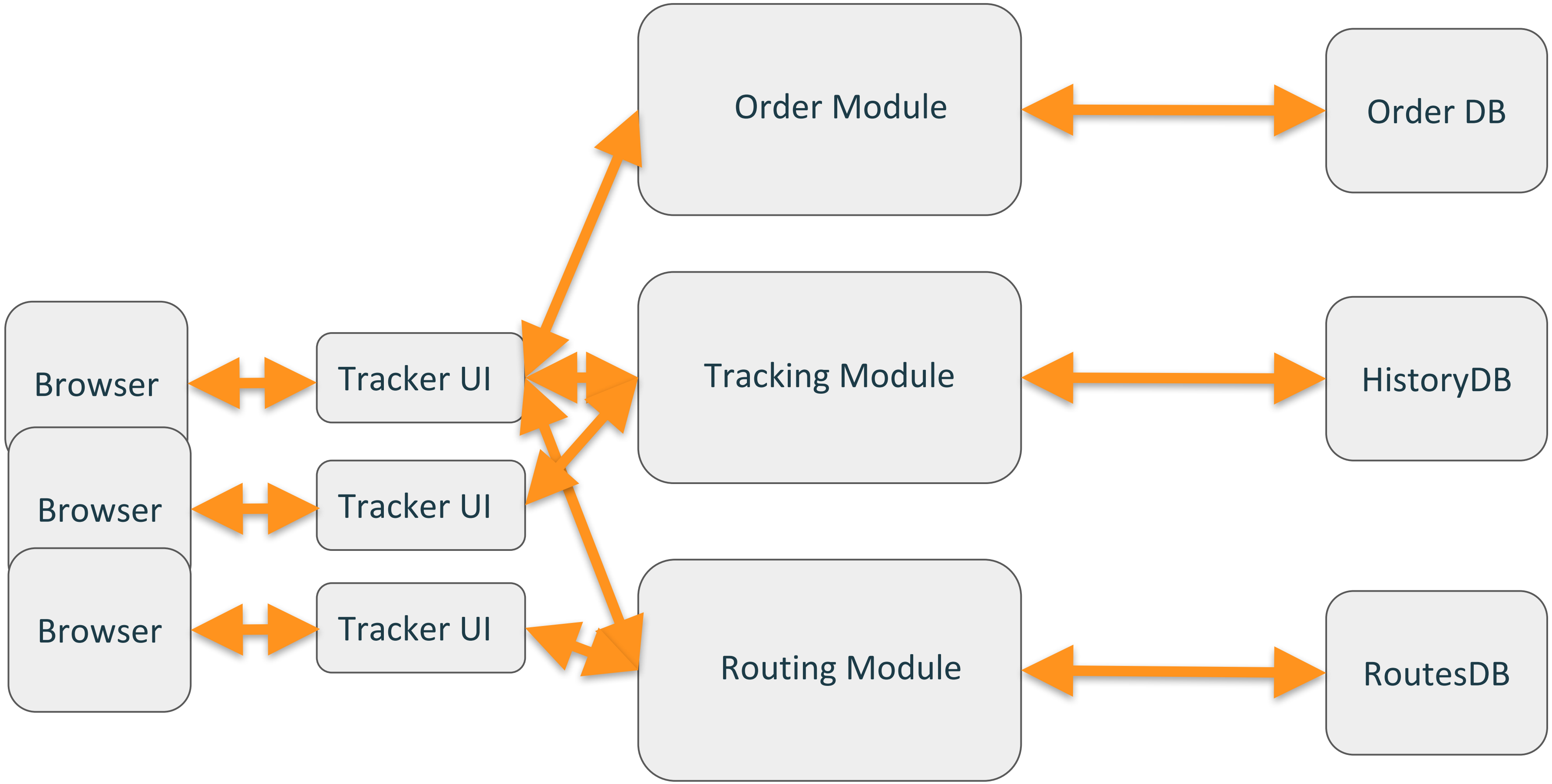


- Rather than acting on data *at rest*, modern software increasingly **operates** on data in ***near real-time***.
- Shortened **time-frames** for putting changes into **production**
- **New business models** evolve from existing ones
- **New questions** need to be answered by existing applications
- Datacenter **costs** need to go down constantly

> Traditional application architectures and platforms are obsolete.
-- Gartner

Modernization!


Browser
WebUI
Module
.JAR
Module
.JAR
Module
.JAR
RDBMS
RDBMS
RDBMS


Order Module
Order DB
Tracking Module
HistoryDB
Routing Module
RoutesDB
Browser
Browser
Browser
Tracker UI
Tracker UI
Tracker UI

REQ: Building and Scaling Microservices

- Lightweight runtime
- Cross – Service Security
- Transaction Management
- Service Scaling
- Load Balancing
- SLA's
- Flexible Deployment
- Configuration
- Service Discovery
- Service Versions
- Monitoring
- Governance
- Asynchronous communication
- Non-blocking I/O
- Streaming Data
- Polyglot Services
- Modularity (Service definition)
- High performance persistence (CQRS)
- Event handling / messaging (ES)
- Eventual consistency
- API Management
- Health check and recovery

"Microservices" is a lousy term

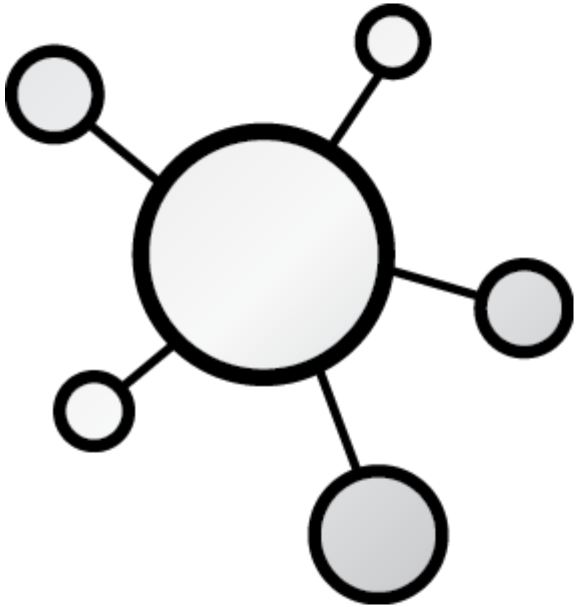
- Size is irrelevant

We want flexible systems and organizations that can adapt to their complex environments, make changes without rigid dependencies and coordination, can learn, experiment, and exhibit emergent behavior.

We need to build systems for **flexibility** and **resiliency**, not just **efficiency** and **robustness.**

Outer Architecture

Software Design



Methodology and Organization

Distributed Systems

Datacenter Operating System

Software Design

Architecture Principles

- Single Responsible Principle
- Service Oriented Architecture
  - Encapsulation
  - Separation of Concern
  - Loose Coupling
- Hexagonal Architecture

Design Patterns

- Domain-driven Design
- Bounded Contexts
- Event Sourcing
- CQRS
- Eventual Consistency
- Context Maps

Design Best Practices

- Design for Automation
- Designed for failure
- Service load balancing and automatic scaling
- Design for Data Separation
- Design for Integrity
- Design for Performance

Strategies For Decomposing

Verb or Use Case
e.g. Checkout UI

Noun
e.g. Catalog product service

Single Responsible Principle
e.g. Unix utilities

What is Lagom?

- Reactive Microservices Framework for the JVM
- Focused on right sized services
- Asynchronous I/O and communication as first class priorities
- Highly **productive development** environment
- Takes you all the way to **production**

Highly opinionated!

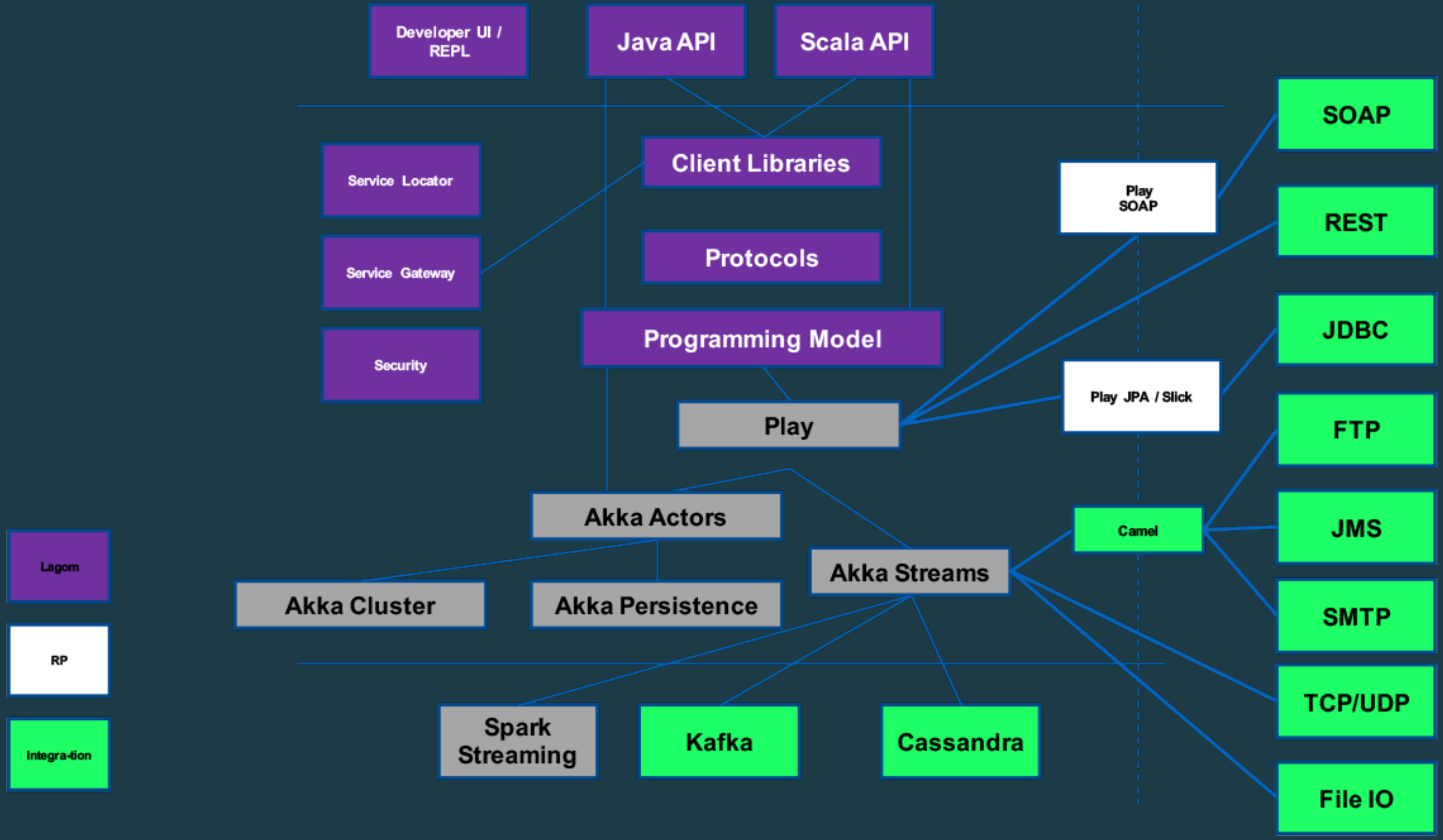
- Use bounded contexts as boundaries for services! **(Domain Driven Design)**
- The event log is the book of record! **(Event Sourcing)**
- Separate the read and write sides! **(CQRS)**
- Microservices, too, need to be elastic and resilient! **(Reactive)**
- **Developer experience** matters! (The Lagom development setup)

The parts

- Service API
- Persistence API
- Development environment
- Production environment

Lagom Persistence API

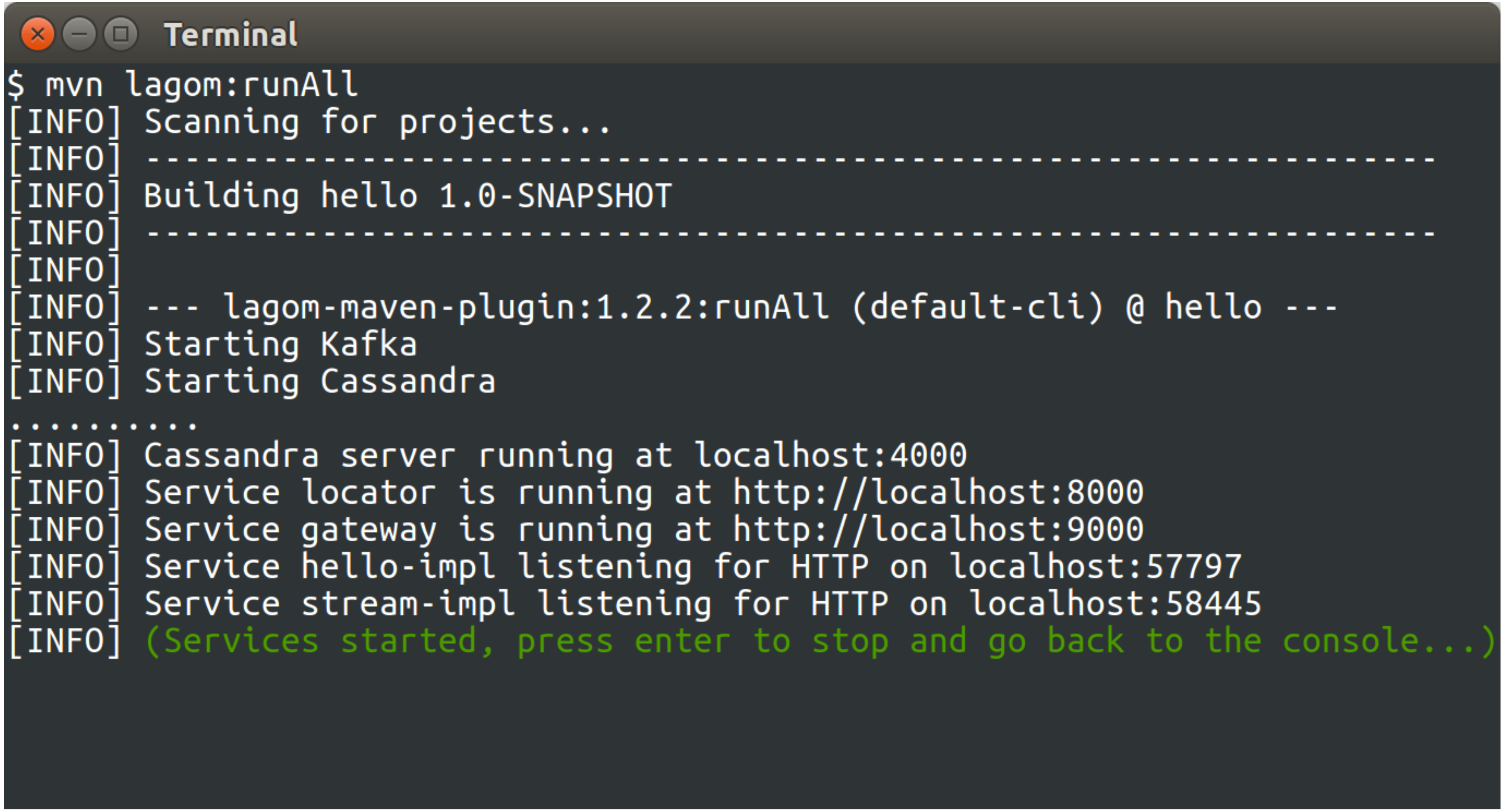
- Event sourced (deltas) with Cassandra backend by default
- No object/relational impedance mismatch
- Can always replay to determine current state
- Allows you to learn more from your data later
- Persistent entity is an Aggregate Root in DDD
- Can be overridden for CRUD if you want


Developer UI / REPL
Java API
Scala API
Client Libraries
Service Locator
Service Gateway
Security
Protocols
Programming Model
Play
Play SOAP
Play JPA / Slick
SOAP
REST
JDBC
FTP
JMS
SMTP
TCP/UDP
File IO
Camel
Akka Actors
Akka Streams
Akka Cluster
Akka Persistence
Spark Streaming
Kafka
Cassandra
Lagom
RP
Integra-tion

Getting started.

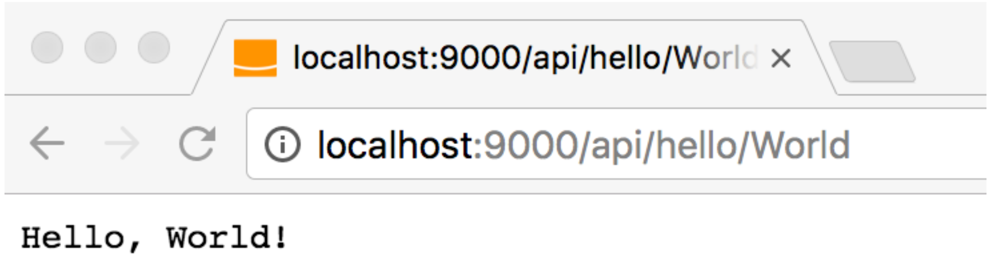
Creating a new Lagom project

```
mvn archetype:generate
   -DarchetypeGroupId=com.lightbend.lagom
   -DarchetypeArtifactId=maven-archetype-lagom-java
   -DarchetypeVersion=1.2.2
```



```
$ mvn lagom:runAll
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------
[INFO] Building hello 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------
[INFO]
[INFO] --- lagom-maven-plugin:1.2.2:runAll (default-cli) @ hello ---
[INFO] Starting Kafka
[INFO] Starting Cassandra
..........
[INFO] Cassandra server running at localhost:4000
[INFO] Service locator is running at http://localhost:8000
[INFO] Service gateway is running at http://localhost:9000
[INFO] Service hello-impl listening for HTTP on localhost:57797
[INFO] Service stream-impl listening for HTTP on localhost:58445
[INFO] (Services started, press enter to stop and go back to the console...)
```

```
$ cd my-first-system
$ mvn lagom:runAll ...
[info] Starting embedded Cassandra server
..........
[info] Cassandra server running at 127.0.0.1:4000
[info] Service locator is running at
http://localhost:8000
[info] Service gateway is running at
http://localhost:9000
..........
[info] Service helloworld-impl listening for HTTP on
0:0:0:0:0:0:0:0:24266
[info] Service hellostream-impl listening for HTTP on
0:0:0:0:0:0:0:0:26230 (Services started, press enter
to stop and go back to the console...)
```

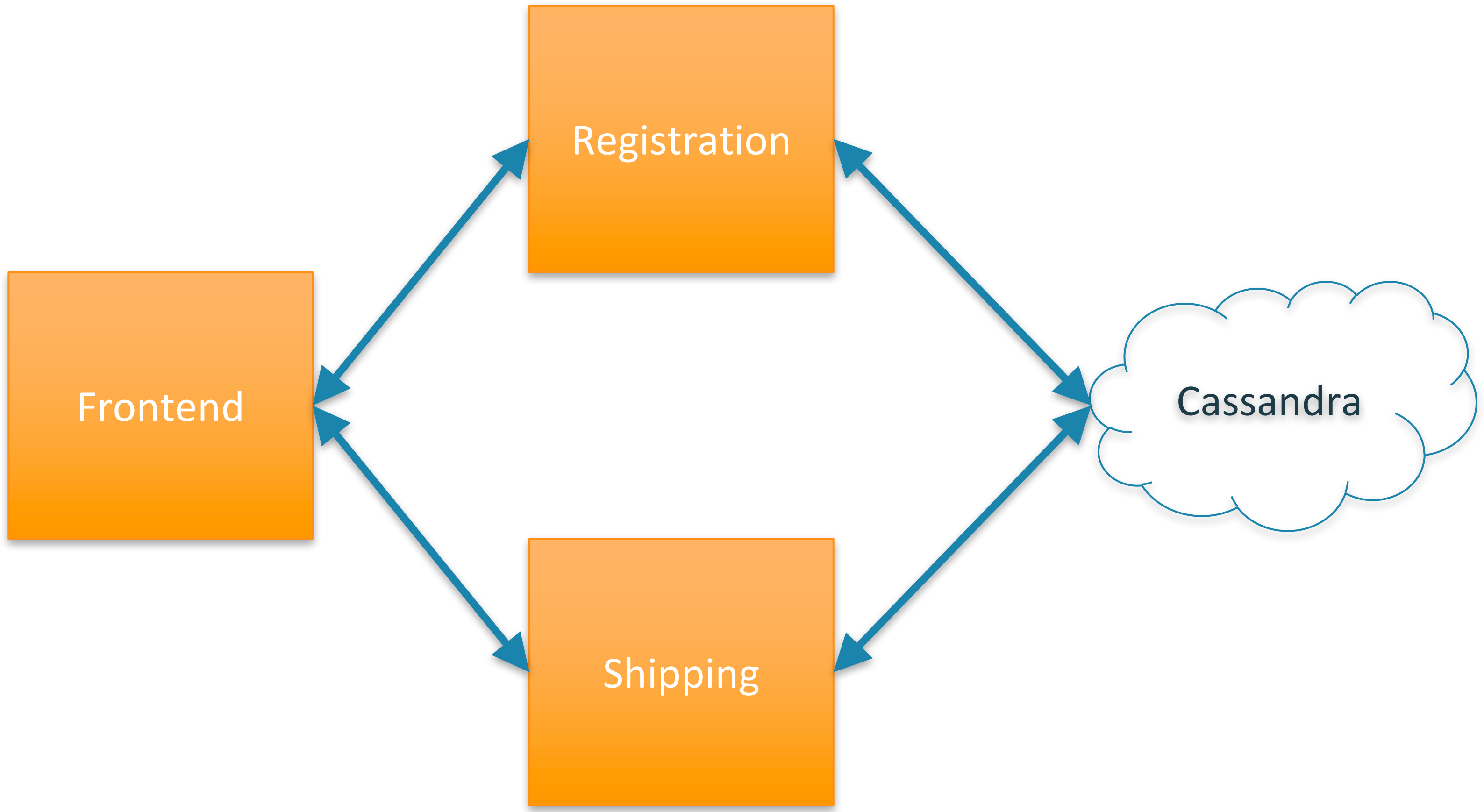

localhost:9000/api/hello/World
localhost:9000/api/hello/World

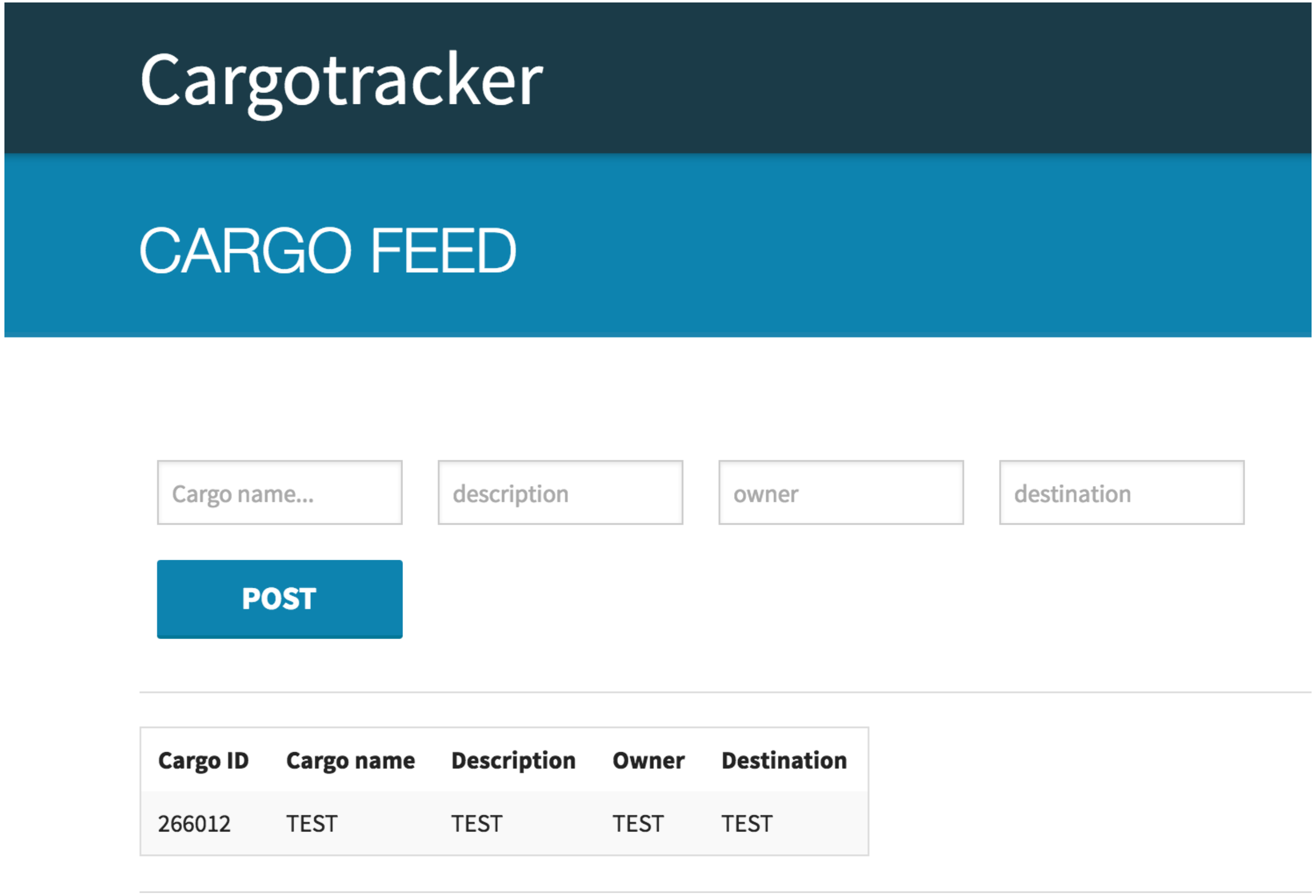
```
Hello, World!
```

The somewhat bigger example!

Cargo Tracker

https://github.com/lagom/activator-lagom-cargotracker


Registration
Frontend
Cassandra
Shipping



Cargo ID	Cargo name	Description	Owner	Destination
266012	TEST	TEST	TEST	TEST

Now that we have our bundles, how do we get into production?

Out of the box support for ConductR but..

- **Lagom doesn't prescribe any particular production environment**, however out of the box support is provided for Lightbend ConductR.
- Zookeper based version: https://github.com/jboner/lagom-service-locator-zookeeper
- Consul based version: https://github.com/jboner/lagom-service-locator-consul

Create Service bundles via sbt

```
>sbt bundle:dist
...
[info] Your package is ready in
/Users/myfear/lagom-cargotracker/front-
end/target/universal/front-end-1.0-
SNAPSHOT.zip
```

Create Service Bundles with Maven

- Creating a bundle configuration file, bundle.conf
- Creating a start script
- Creating a Maven assembly plugin descriptor to create the bundle zip
- Binding the Maven assembly plugin and Lagom renameConductRBundle goals to your projects lifecycle

http://www.lagomframework.com/documentation/1.3.x/java/ConductR.html lagom

Get started with Lagom 1.3.0 today

Java

Scala

Maven

sbt

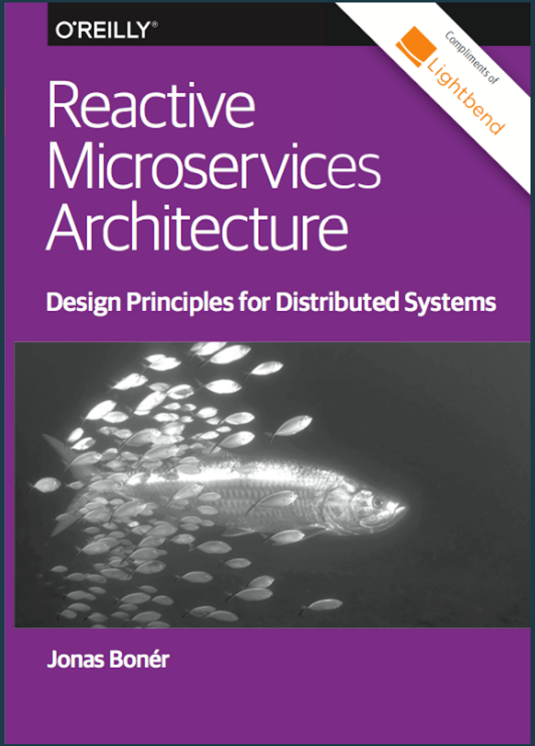
Next Steps! Download and try Lagom!

Project Site:
http://www.lightbend.com/lagom

GitHub Repo:
https://github.com/lagom

Documentation:
http://www.lagomframework.com/documentation/1.3.x/java/Home.html

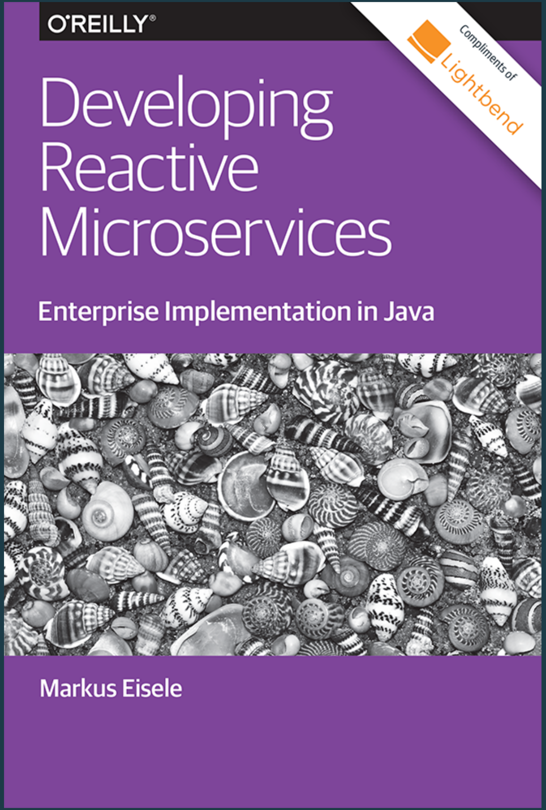
Example:
https://github.com/typesafehub/activator-lagom-java


O'REILLY®
Compliments of Lightbend
Reactive Microservices Architecture
Design Principles for Distributed Systems
Jonas Bonér

Written for architects and developers that must quickly gain a fundamental understanding of microservice-based architectures, this free O'Reilly report explores the journey from SOA to microservices, discusses approaches to dismantling your monolith, and reviews the key tenets of a Reactive microservice:

- Isolate all the Things
- Act Autonomously
- Do One Thing, and Do It Well
- Own Your State, Exclusively
- Embrace Asynchronous Message-Passing
- Stay Mobile, but Addressable
- Collaborate as Systems to Solve Problems

http://bit.ly/ReactiveMicroservice


O'REILLY®
Compliments of
Lightbend
Developing Reactive Microservices
Enterprise Implementation in Java
Markus Eisele

The detailed example in this report is based on Lagom, a new framework that helps you follow the requirements for building distributed, reactive systems.

- Get an overview of the Reactive Programming model and basic requirements for developing reactive microservices
- Learn how to create base services, expose endpoints, and then connect them with a simple, web-based user interface
- Understand how to deal with persistence, state, and clients
- Use integration technologies to start a successful migration away from legacy systems

http://bit.ly/DevelopReactiveMicroservice

Lightbend