Java. Cloud. Leadership.

WildFly-Swarm - Does my fatjar look big in this?

Mark Little, VP
Bruno Georges, Director

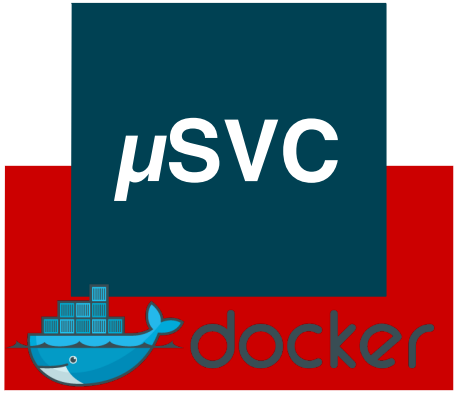
Microservices

TRADITIONAL **SOA** **MICROSERVICES**

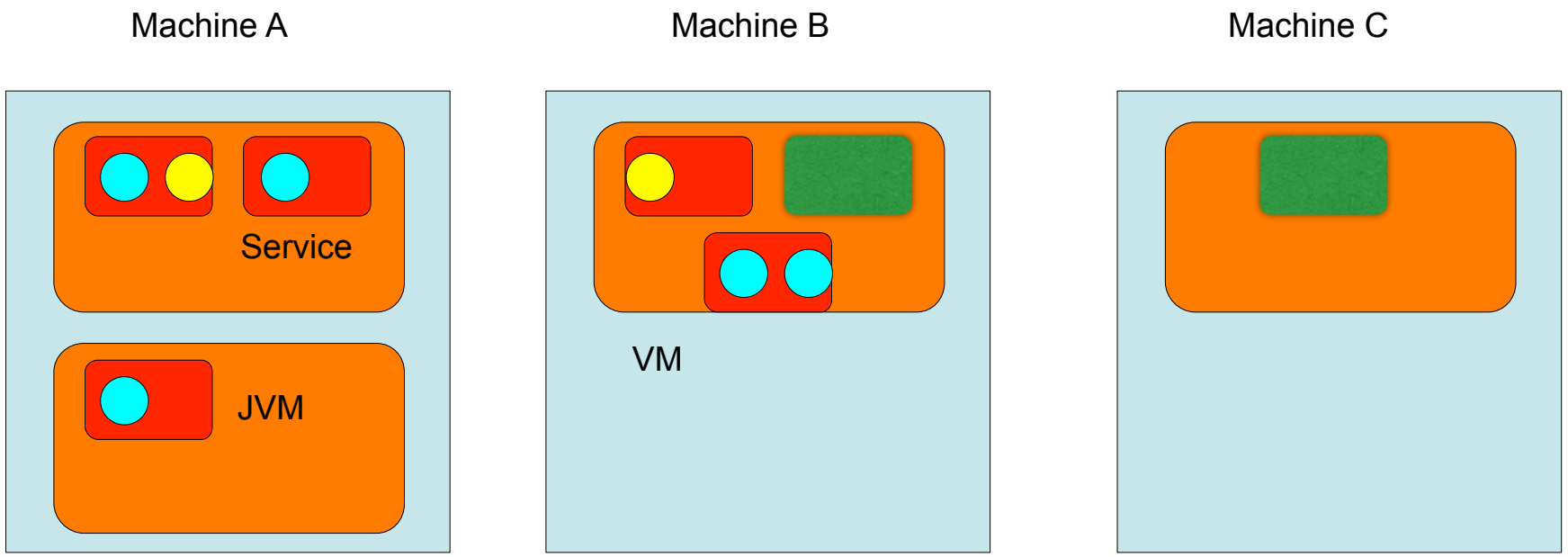
One size != all

Microservices + containers

Docker …



Services, Linux containers and JVMs



- Java EE services split across machines, containers and JVMs

Microservices and Java EE

- Not everyone wants to use Docker
- Not everyone wants to use Node.js
- Many developers are happy with Java EE
  - Robust and mature components
  - Scalable, standards compliant, integrates well
- Not everyone wants to use all of Java EE
  - Stripping down EAP/WildFly is common
  - Higher cloud density and multi-tenancy
- JSR 111 (Java Services Framework)

Enterprise Mobile

JBoss by Red Hat

JavaEE

JBoss by Red Hat

Jopr

API (Java, Ruby, Python, C++, etc...)

HTML

Infinispan

errai

jBPM

JBossMSC

Services

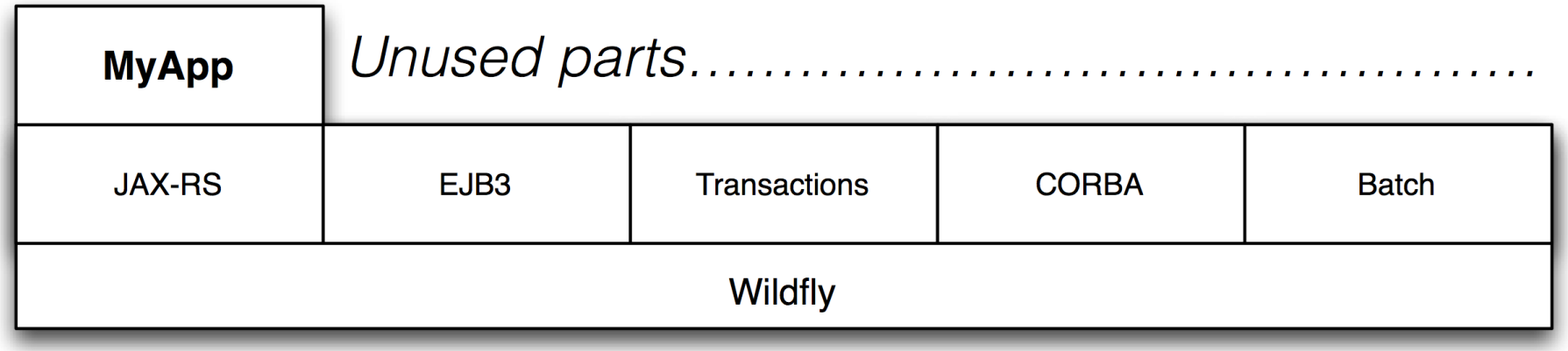
Social Aspect

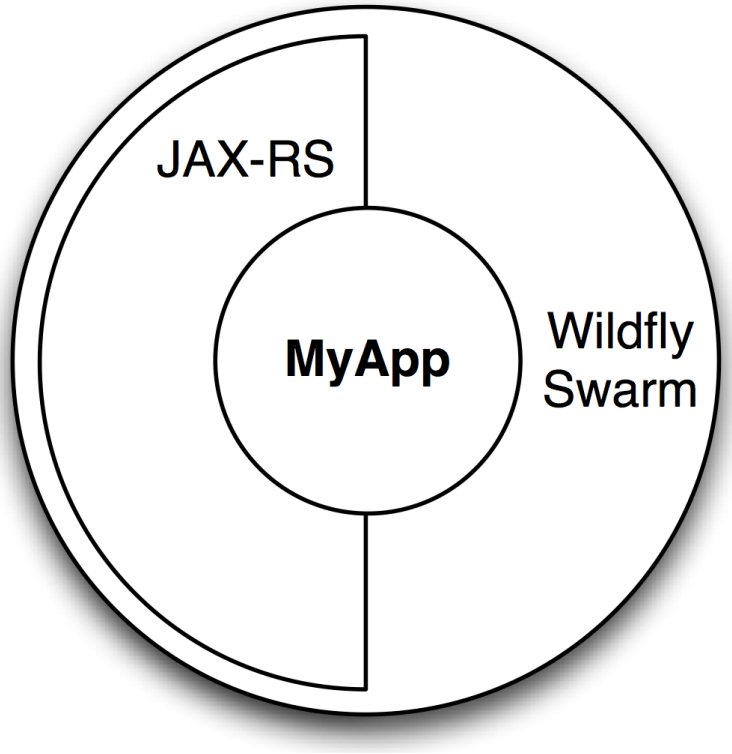
WildFly-Swarm

- Allows Java EE components to become independently deployable (micro) services
  - Applications deploy with only the components needed
  - Just enough Application Server (JeAS)
- Re-uses existing WildFly and EAP
  - Self-contained services without wrapping it all in Docker
- Build applications as fat jars (Java circa 1996)
  - Avian?
- The 2009 JBossAS 7 re-architecture makes it possible

Where might it be useful?

- Building EE applications with limited capabilities
  - Comfortable with the Java EE model
- Need multiple components/services for business logic
  - WildFly-core handles class loading and lifecycle issues
- More streamlined “virtual” application server
  - Shared services
  - Multi-tenancy/higher densities
- Microservices (aka SOA)





myapp-swarm.jar

Putting it into practice

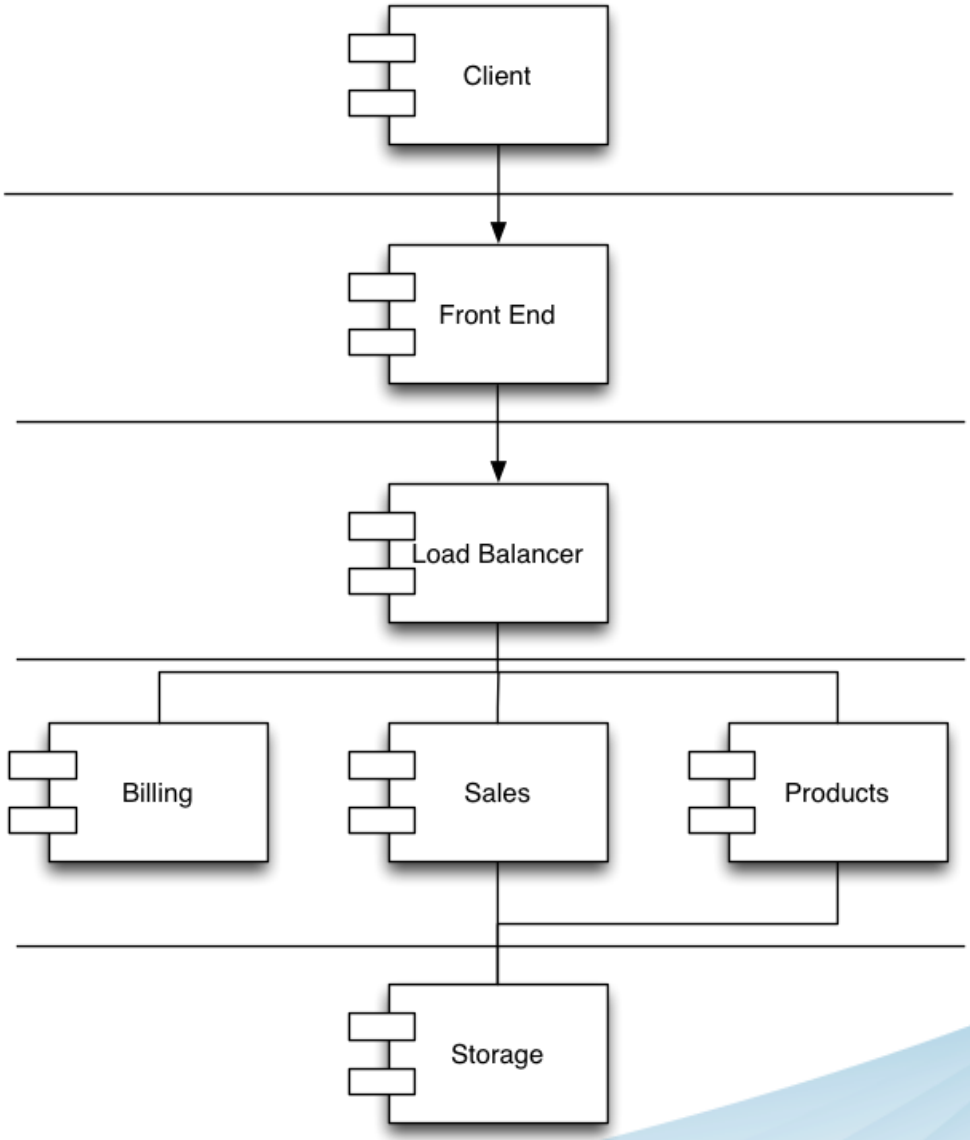
Red Hat Reference Architecture

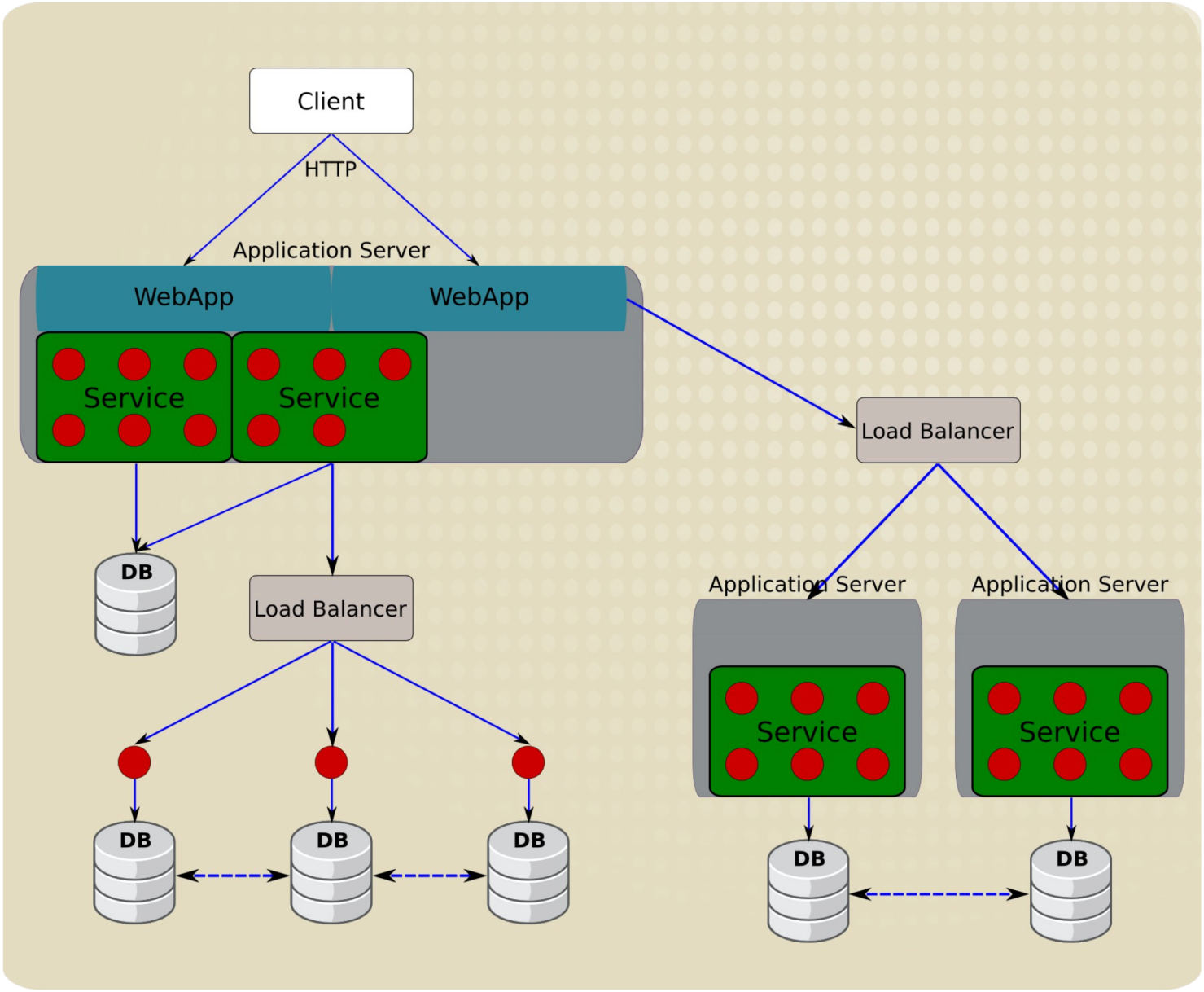
- Microservice Architecture - Building microservices with JBoss EAP 6
  - By Babak Mozaffari
  - Define MSA
  - Analyze advantages/disadvantages
  - Explore different deployment scenarios
- Application / demo available on Github
  - EAP6 implementation
  - Swarm implementation

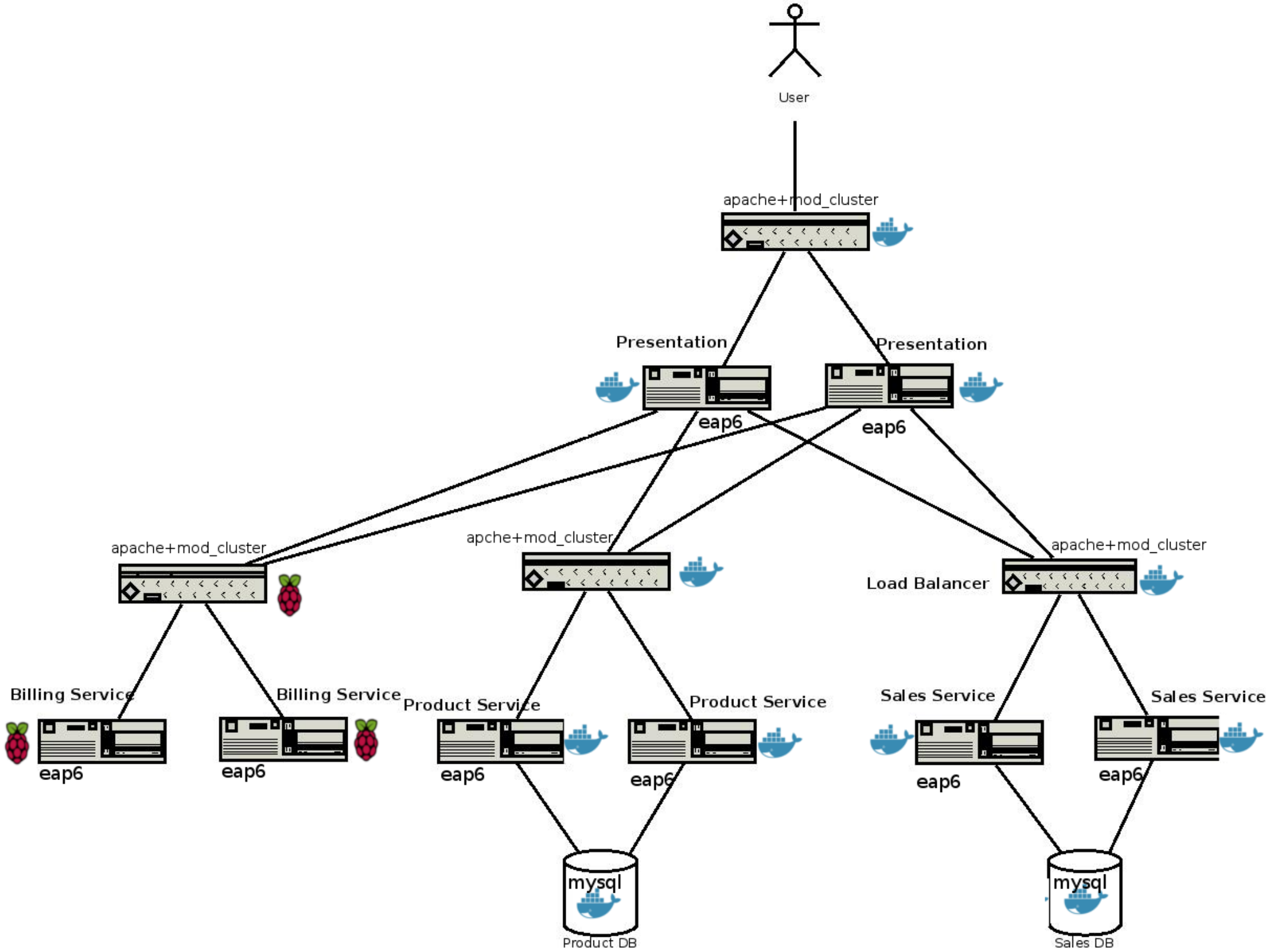
Sample Application

Online shopping - Architecture

- Presentation Layer: frontend ui to call the backend service (`code/Presentation/target/presentation.war`)
- Product service: provide all restful api about stock product:for example list, search detail info (`code/Product/target/product.war`)
- Sales service: provide user register, login, cart, order restful api (`code/Sales/target/sales.war`)
- Billing Service： provide checkout and payment restful api for customer's order (`code/Billing/target/billing.war`)

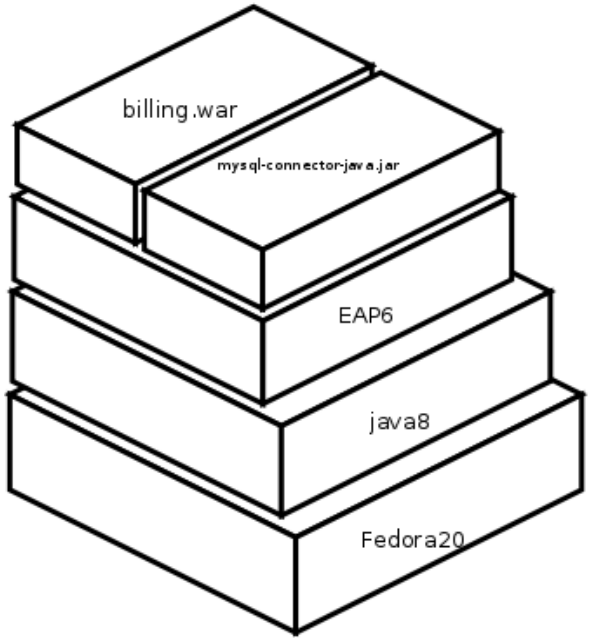



Client
HTTP
Application Server
WebApp
WebApp
Service
Service
Load Balancer
DB
Load Balancer
Application Server
Application Server
Service
Service
DB
DB
DB
DB
DB

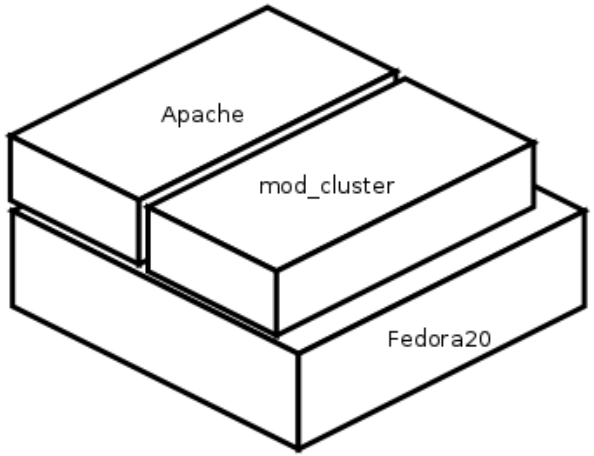

User
apache+mod_cluster
Presentation
Presentation
eap6
eap6
apache+mod_cluster
apche+mod_cluster
apache+mod_cluster
Load Balancer
Billing Service
Billing Service
Product Service
Product Service
Sales Service
Sales Service
eap6
eap6
eap6
eap6
eap6
eap6
mysql
mysql
Product DB
Sales DB

EAP6 based demo Architecture - Docker layers

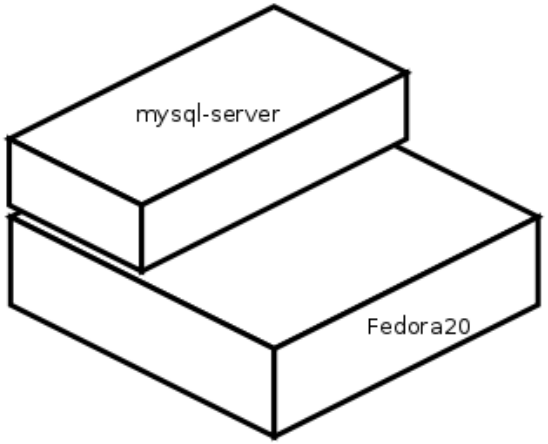
msdemo-billing



msdemo-httpd



msdemo-db



Now using Swarm

Reuse or rewrite?

- Changes from EAP6 to Swarm revolves mostly around maven dependencies. For example JAX-RS:

```
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>jaxrs-api</artifactId>
  <scope>provided</scope>
</dependency>
```

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>wildfly-swarm-jaxrs</artifactId>
  <version>${version.wildfly-swarm}</version>
</dependency>
```

Bootstrapping

- Some specifics
  - org.wildfly.swarm.bootstrap.Main(.main)
    - Bootstraps the jboss-modules system
  - user's main() or org.wildfly.swarm.Swarm
    - Construct a Container, apply Fractions (explicitly or via defaults) and then start()
- Main or not Main ?
  - main() not a big player in server-side Java EE
  - In Swarm much is defaulted to ease developers burden
  - If you have no main() then a default Container is created and every fraction is defaulted

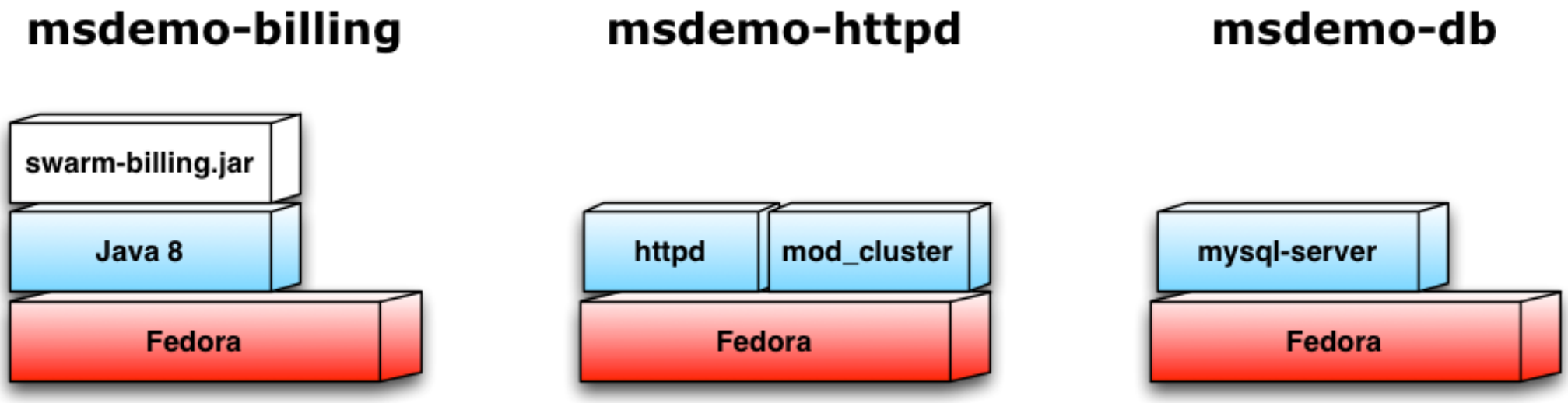
Run

- Run - different options:
  - single executable jars `(i.e: billing-swarm.jar)`
    - `java -jar billing-swarm.jar`
    - `mvn wildfly-swarm:run`
  - `In your IDE run the main class`
  - In your IDE
- Let's have a look at some examples

Choice of deployments

- As a Single executable
  - fat-jar
- In a container
  - Docker
  - Other containers ?
- Let's look at a Docker example
  - and compare with EAP6 based MSA

Swarm based Microservices demo - Docker layers



Running the demos

- Git clone:
  - https://github.com/bgeorges/microservices
- Choose an implementation: EAP6, Swarm, Vert.X, …
- Install and start docker service on your machine
- Go to docker directory
- Run build.sh build all docker images
- Run all docker containers with start_all_docker.sh

Next steps

- Discovery/Load-balancing
  - Fabric8 can help but not exclusively
  - Vert.x event bus
  - Generic libraries in the spirit of NetflixOSS-Ribbon etc.
- Testing and Contracts
  - Need to be able to mock other services effectively
    - Don’t make me install everything to test one bit
  - Need to be able to mock other services correctly
- Expose services to other languages
- Deployable into other containers

Conclusions

- "And miles to go before I sleep", Robert Frost
- https://github.com/wildfly-swarm
- @wildflyswarm
- http://wildfly.org/swarm/
- Interested in feedback and input on direction