# Java EE Microservices with WildFly Swarm

Heiko Braun <hbraun@redhat.com>
Feb 2016

Organised by the Lightweight Java User Group Munich, hosted at ComSysto. Thanks to both.

# This evening

- The Context: Microservices and Java EE
- WildFly Swarm: Concepts, Ideas & Mechanics
- Code and Demo
- Outlook, Discussions and (hopefully) beer

# What are Microservices anyway?

# Like SOA, but different …

- Microservices are different primarily due to innovations like:
  - Linux containers,
  - automated, elastic infrastructure, you know, the cloud
  - plus wide adoption of CI, continuous integration
  - and the growing adoption of DevOps principles & practices

"In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies."

– **Martin Fowler**, ThoughtWorks

# What is Java EE anyway?

# Perspectives on Java EE

- It's different things to different people:
  - A collection of (useful) API's
  - Technical capabilities of a system
  - A love/hate relationship (of the past)
  - (Existing) knowledge and expertise

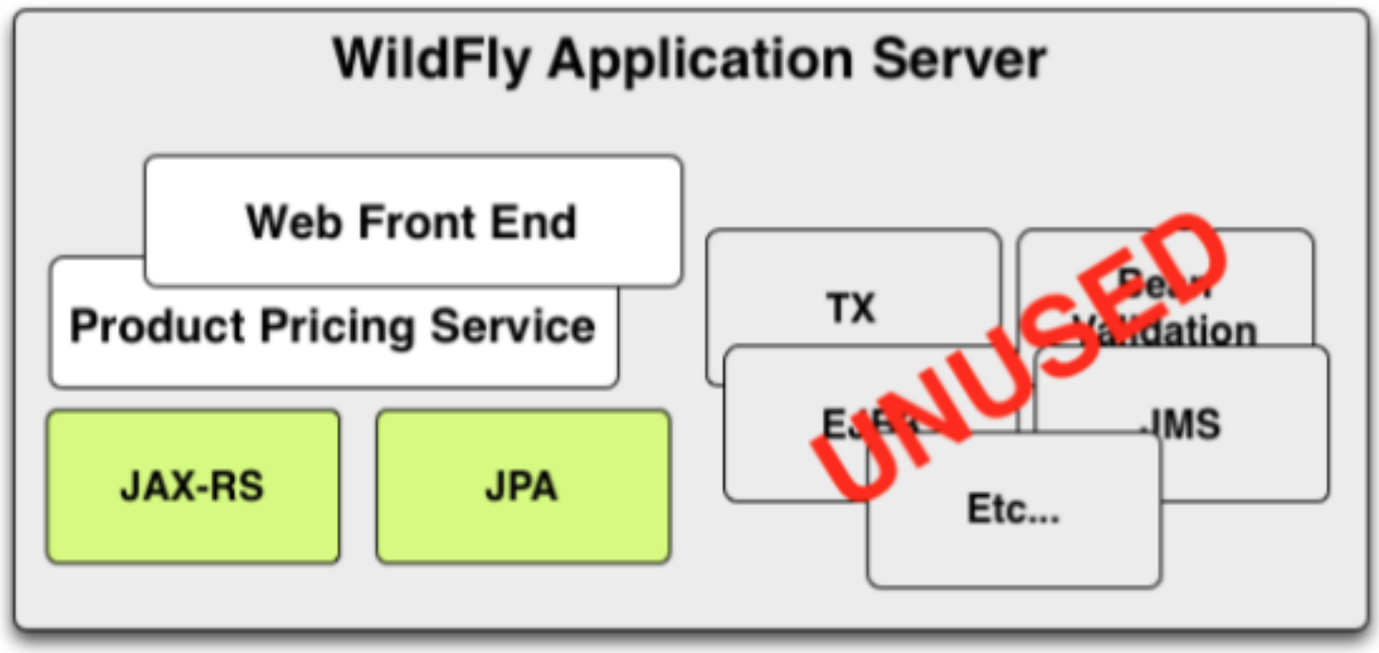
# Hello WildFly Swarm

# WildFly Swarm

- OSS Project sponsored by Red Hat
- Sidekick of Wildfly Application Server
- Small, but ambitious and friendly community
- Part of a bigger system of interrelated projects under the JBoss / Red Hat umbrella

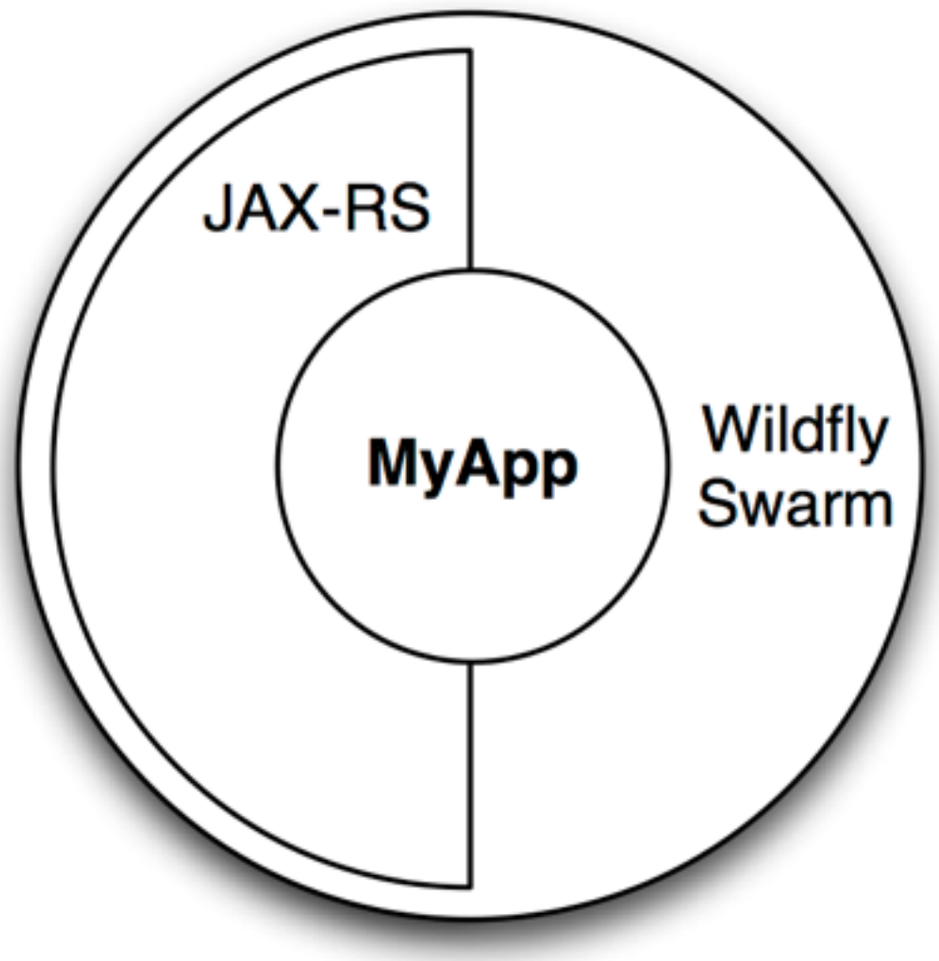
# Just Enough App Server

- Use the API's you want
- Include the capabilities you need
- Wrap it up for deployment



# Uberjar

- A single .jar file containing your application,
- the portions of WildFly required to support it,
- an internal Maven repository of dependencies,
- plus a shim to bootstrap it all



# Fractions

- A well-defined collection of application capabilities.
  - May map directly to a WildFly subsystem,
  - or bring in external capabilities such as Netflix Ribbon.

# What Fractions can do

- Enable WildFly subsystems (JAX-RS, Infinispan)
- Integrate additional system capabilities (Topology)
- Provide deployments (ribbon-webapp, jolokia)
- Alter deployments (keycloak)

# Example Fractions

| | | |
|---|---|---|
| Datasources | Keycloak (SSO) | Undertow (Http/Web) |
| EJB | Messaging | Clustering |
| JAX-RS | JPA | Infinispan |
| Transactions | CDI | Management |

# Convert Java EE App to use WildFly Swarm

javaee-samples / **javaee7-simple-sample**

Watch 4 | Star 13 | Fork 15

A simple Java EE 7 Sample

**61** commits | **1** branch | **8** releases | **2** contributors

Branch: **master** **javaee7-simple-sample** / +

**arun-gupta** cleaning up whitespace — Latest commit `9ef152e` on Sep 19

| | | |
|---|---|---|
| src/main | adding 'all' beans.xml | 8 months ago |
| .gitignore | adding gitignore | a year ago |
| README.asciidoc | reorganizing | 8 months ago |
| pom.xml | cleaning up whitespace | a month ago |

**README.asciidoc**

# A simple Java EE 7 Sample

This is a trivial Java EE 7 sample.


Code
Issues 0
Pull requests 0
Wiki
Pulse
Graphs


**SSH** clone URL

`git@github.com:j`

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

# Adding Fractions to your build

**pom.xml**

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

# Adding the WildFly Swarm Plugin to your build

**pom.xml**

```
<plugin>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-plugin</artifactId>
    <version>${version.wildfly-swarm}</version>
    <configuration>
        <properties>
            <swarm.bind.address>127.0.0.1</swarm.bind.address>
            <java.net.preferIPv4Stack>true</java.net.preferIPv4Stack>
            <jboss.node.name>${project.artifactId}</jboss.node.name>
        </properties>
        <useUberJar>true</useUberJar>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>package</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

# Building a WildFly Swarm App

```
$ mvn package

$ ls -l target/javaee7-simple-sample-swarm.jar
```

# Running a WildFly Swarm App

```
$ java -jar target/javaee7-simple-sample-swarm.jar

$ mvn wildfly-swarm:run
```

# Going beyond simple (and Java EE)

# Custom Configuration

```
public class Main {

    public static void main(String... args) throws Exception {

        Container container = new Container();

        container.fraction(new DatasourcesFraction()
                .jdbcDriver(new JDBCDriver("h2")
                        .driverName("h2")
                        .driverDatasourceClassName("org.h2.Driver")
                        .xaDatasourceClass("org.h2.jdbcx.JdbcDataSource")
                        .driverModuleName("com.h2database.h2"))
                .dataSource(new DataSource("LibraryDS")
                        .driverName("h2")
                        .jndiName("java:/LibraryDS")
                        .connectionUrl("jdbc:h2:./library;DB_CLOSE_ON_EXIT=TRUE")
                        .userName("sa")
                        .password( "sa" )));

        container.start();

    }
}
```

(alternatively use standalone.xml)

# Advertising Services

```
public class Main {

    public static void main(String... args) throws Exception {

        Container container = new Container();

        JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class);
        deployment.addPackage(Main.class.getPackage());
        deployment.as(RibbonArchive.class).advertise("pricing");

        container.start();
        container.deploy(deployment);

    }
}
```

(supports different service registries)

# Load Balancing & Circuit Breaking

```
@ResourceGroup( name="time" )
public interface TimeService {

    TimeService INSTANCE = SecuredRibbon.from(TimeService.class);

    @TemplateName("currentTime")
    @Http(
            method = Http.HttpMethod.GET,
            uri = "/"
    )
    @Hystrix(
            fallbackHandler = TimeFallbackHandler.class
    )
    RibbonRequest<ByteBuf> currentTime();

}
```

(Integration of Ribbon with Topology. Supports Hystrix)

# Securing Access to Services

```
public class Main {

    public static void main(String... args) throws Exception {

        Container container = new Container();

        JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class);
        deployment.addPackage(Main.class.getPackage());
        deployment.as(Secured.class)
                .protect("/items")
                .withMethod("GET")
                .withRole("*");

        container.start();
        container.deploy(deployment);

    }
}
```

(provided by Keycloak: OpenID, SAML, Social Login, OAuth, LDAP, Active Directory)

# Publishing Service Interface Descriptions

```
@Path("/time")
@Api(value = "/time", description = "Get the time", tags = "time")
@Produces(MediaType.APPLICATION_JSON)
public class TimeResource {

    @GET
    @Path("/now")
    @ApiOperation(value = "Get the current time",
            notes = "Returns the time as a string",
            response = String.class
    )
    @Produces(MediaType.APPLICATION_JSON)
    public String get() {

        return String.format("{\"value\" : \"The time is %s\"}",
                new DateTime()
        );
    }
}
```

(provided by Swagger)

```
$ curl http://localhost:8080/swagger.json
```


```
{
  "basePath": "/",
  "paths": {
    "/time/now": {
      "get": {
        "description": "Returns the time as a string",
        "operationId": "get",
        "parameters": [],
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "successful operation",
            "schema": {
              "type": "string"
            }
          }
        },
        "summary": "Get the current time"
      }
    }
  }
}
```

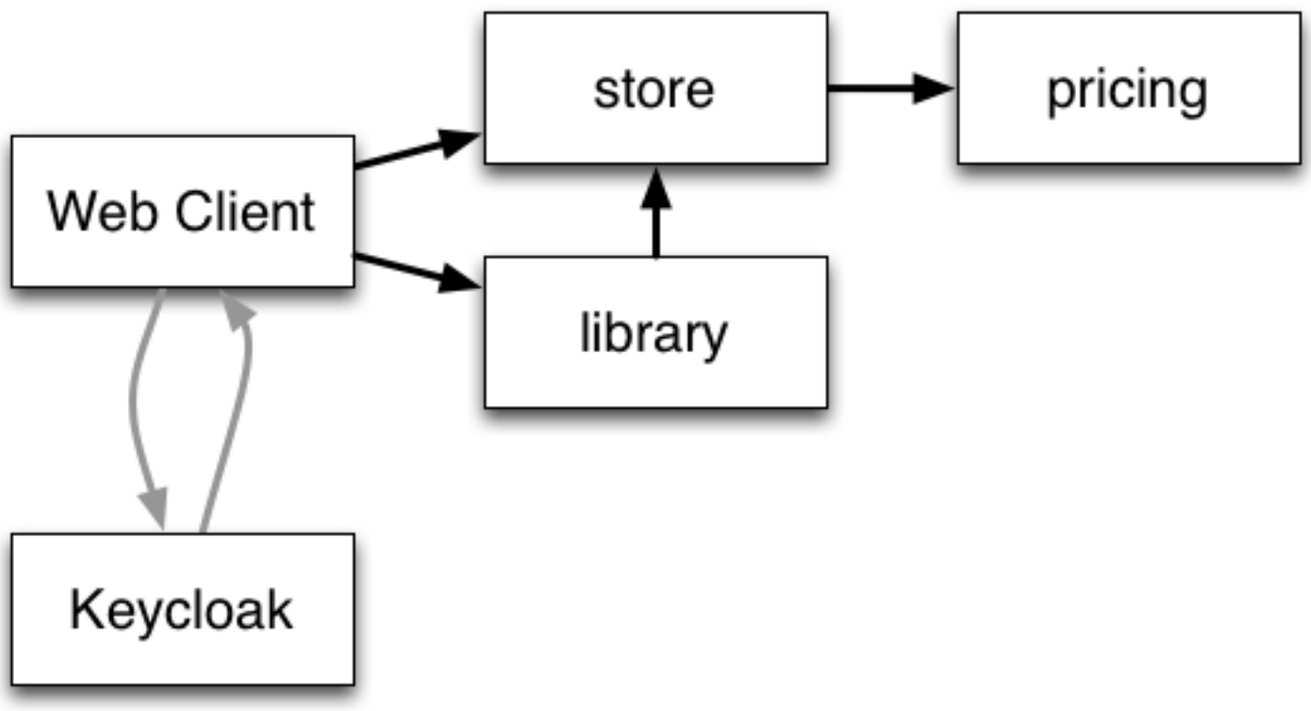
# Other Noteworthy Features

- Testing:
  - Arquillian (in container, web driver)
  - Consumer-Driven Contracts
    (expressing and asserting expectations of a provider contract)
- Logging & Monitoring
  - Simple REST interface on each node
  - Centralised Logging with Logstash
  - Push Runtime Data to Hawkular,Influx,etc
- Remote Management
  - CLI (full access to the server config and runtime state)

# Booker Demo

- Booker! is an electronic bookstore that demonstrates how many WildFly Swarm-based microservices can play together.

- https://github.com/wildfly-swarm/booker



# The Road Ahead

- API Gateway
  (integration with APIMan)

- Integration with Kubernetes / OpenShift V3
  (i.e. Service Discovery)

- Environment Abstractions (Local, CI, Cloud)

- Tooling (Forge, Eclipse, IntelliJ)

- Spring Support

# Get Involved

- Project Home: http://wildfly-swarm.io
- GitHub: https://github.com/wildfly-swarm
- Twitter: @wildflyswarm
- Freenode: @wildfly-swarm
- Issues: https://issues.jboss.org/projects/SWARM
  (see ‘getting-started’ labels)

# Resources

- Keycloak: http://keycloak.jboss.org/
- Hawkular: http://www.hawkular.org/
- OpenShift V3: https://blog.openshift.com/openshift-v3-deep-dive-docker-kubernetes/
- APIMan: http://www.apiman.io/latest/
- Arquillian: http://arquillian.org/
- WildFly: http://wildfly.org
- Consul: https://www.consul.io/