JAVA EE 7 APPLICATIONS AS A MICROSERVICE WITH

Ken Finnigan

Mark Little

HELLOWORLD JAX-RS WILDFLY SWARM

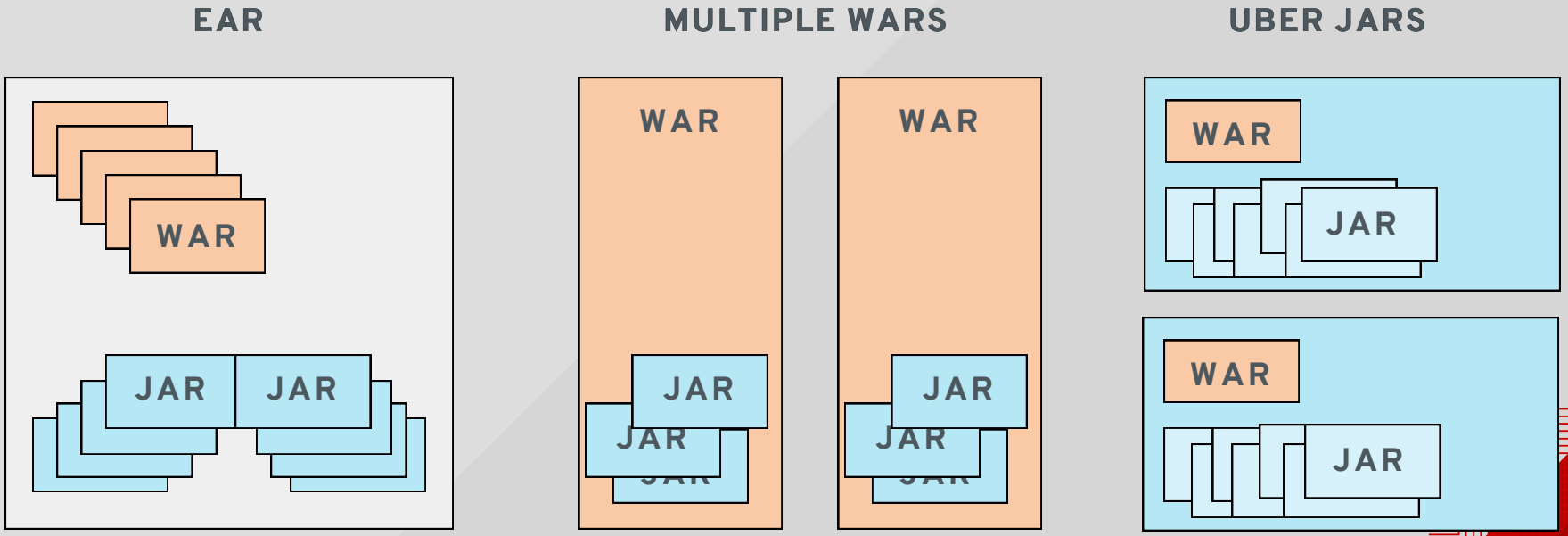
MICROSERVICES

- Decoupled
- Independent release cycles
- Micro functionality, not lines-of-code
- Ideally self-contained
- Scales independently
- All aspects owned by a 2 pizza team

MICROSERVICES + JAVA EE

- Unexpected bedfellows
- Millions of developers know Java EE
  - Robust and mature components
  - Scalable, standards compliant, integrates well
  - Why create new APIs?
- Not everyone wants to use all of Java EE
  - Stripping down EAP/WildFly is common

EVOLUTION


EAR
WAR
JAR
JAR
MULTIPLE WARS
WAR
JAR
JAR
WAR
JAR
JAR
UBER JARS
WAR
JAR
WAR
JAR

WILDFLY SWARM

- Allows Java EE components to become independently deployable services
  - Applications deploy with only the components needed
- Re-uses existing WildFly
  - Self-contained services
- Build applications as uber jars
- Not limited to WildFly subsystems
  - Netflix OSS - Ribbon, Hystrix

JAX-RS Resource

```
@Path("persons")
public class PersonResource {

  @Inject
  PersonDatabase database;

  @GET
  @Produces("application/xml")
  public Person[] get() {
    return database.currentList();
  }

  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Person get(@PathParam("id") int id) {
    return database.getPerson(id);
  }
}
```

JAX-RS Resource in WildFly Swarm

```
@Path("persons")
public class PersonResource {

  @Inject
  PersonDatabase database;

  @GET
  @Produces("application/xml")
  public Person[] get() {
    return database.currentList();
  }

  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Person get(@PathParam("id") int id) {
    return database.getPerson(id);
  }
}
```

SPOT THE DIFFERENCE?

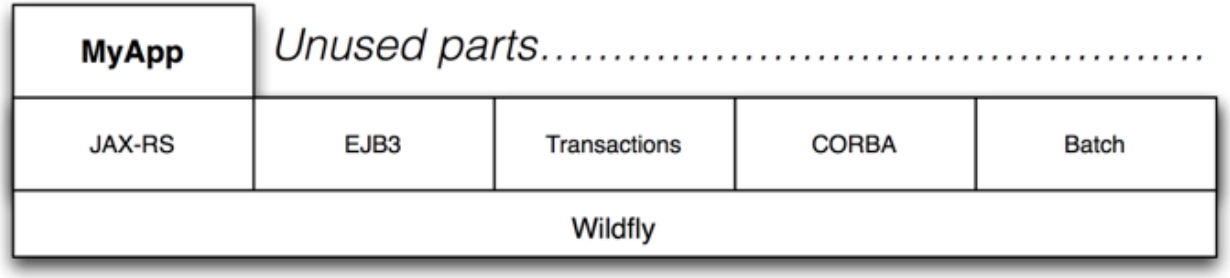
JAX-RS Resource

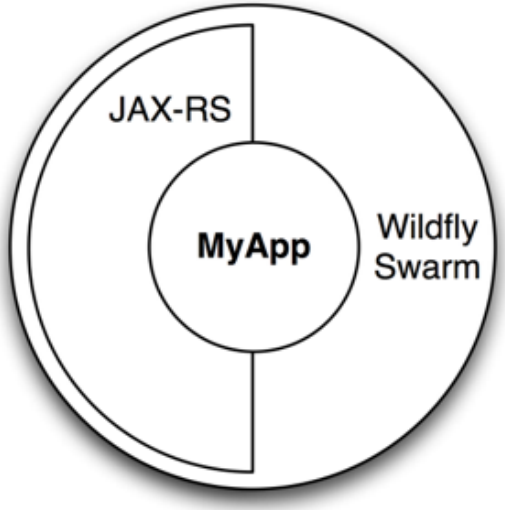
```
@Path("persons")
public class PersonResource {

  @Inject
  PersonDatabase database;

  @GET
  @Produces("application/xml")
  public Person[] get() {
    return database.currentList();
  }

  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Person get(
      @PathParam("id") int id) {
    return database.getPerson(id);
  }
}
```

JAX-RS Resource in WildFly Swarm

```
@Path("persons")
public class PersonResource {

  @Inject
  PersonDatabase database;

  @GET
  @Produces("application/xml")
  public Person[] get() {
    return database.currentList();
  }

  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Person get(
      @PathParam("id") int id) {
    return database.getPerson(id);
  }
}
```

SIMPLIFY





myapp-swarm.jar

EASE OF USE

- WildFly subsystems via Fractions
- Maven plugin
- Application code unchanged
  - Sensible defaults

WHAT IS A FRACTION?

Minimally

- Defines a module.xml for a jar(s), with any required module dependencies
- Typical usages:
  - Non WildFly subsystems, ie. RxJava, RxNetty
  - Activating WildFly modules that are excluded by default

WHAT IS A FRACTION?

Additionally

- An API for defining how an application may configure a Fraction
  - Utilizes generated Java API of WildFly subsystems (WIP)
  - Provide custom archive types
- A Runtime for defining a default Fraction configuration and setting the required Fraction configuration in the Container
- module.xml for all the above pieces with appropriate dependencies

WHAT IS A FRACTION?

Inheritance

- Fraction dependencies in Maven
  - e.g. JAX-RS Fraction implies Undertow Fraction
- Developer's pom.xml relatively clean, only highest order Fraction required

WHAT IS A FRACTION?

Aggregation

- Define new Fraction with dependencies on any combination of Fractions
- Custom configuration for particular combination of Fractions
- Useful as "common" framework for developers

WILDFLY SWARM FRACTIONS

WILDFLY SUBSYSTEMS

Datasources
EJB
JAX-RS
Transactions
JMX
Remoting
CDI/Weld
Keycloak
Messaging
JSF
Undertow
JPA
Bean Validation
JCA
Clustering
Mail
Infinispan
Hawkular

NON WILDFLY

Ribbon
Hystrix
RxJava
Ribbon Secured
RxNetty

CUSTOM

Logstash

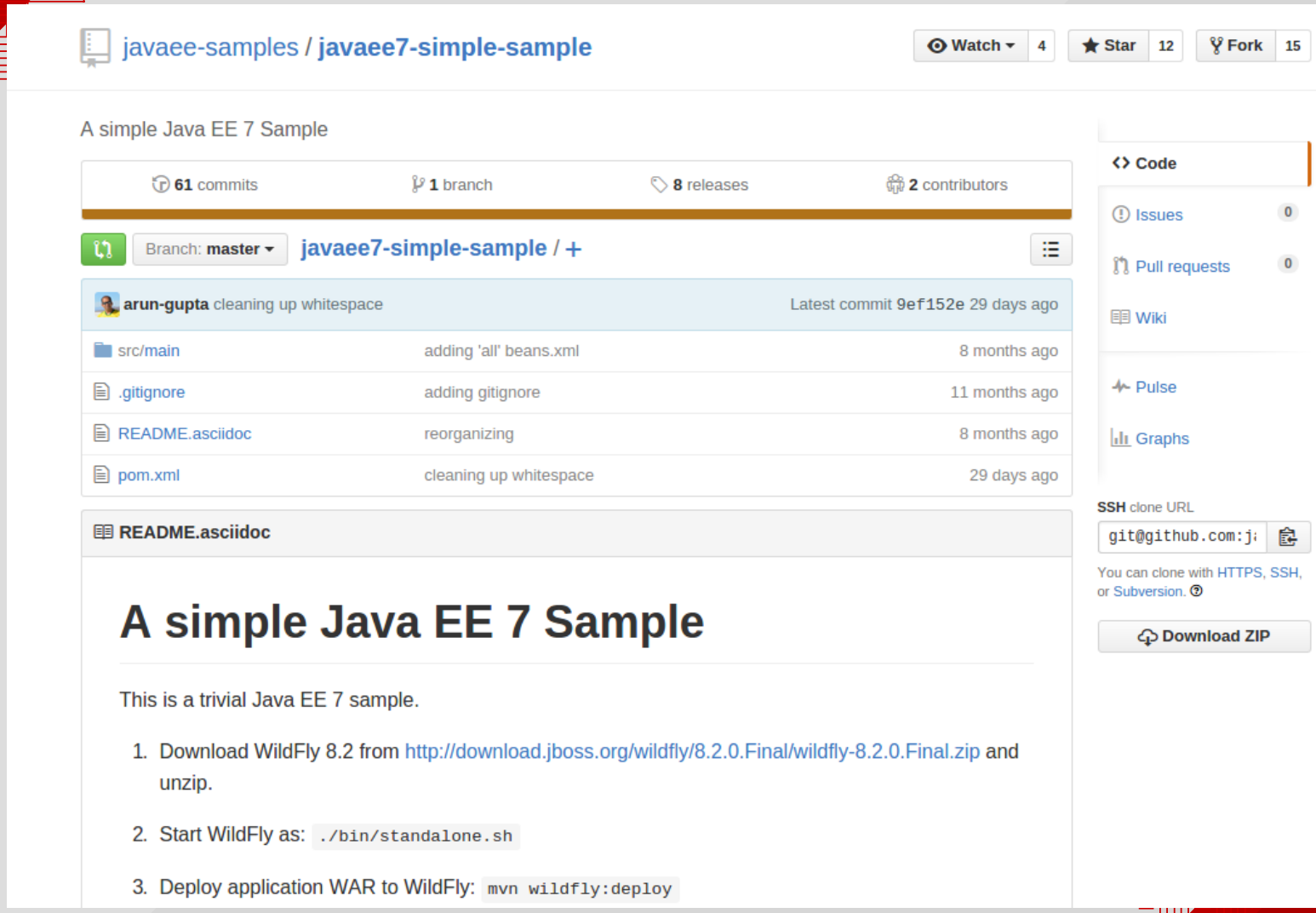
AND GROWING!

DO I CARE?

- Building blocks for using WildFly Swarm
- 99% - Use existing Fractions
- 1% - Develop your own Fractions
- Replacement for existing Maven dependencies

CONVERT JAVA EE 7 APPLICATION TO USE WILDFLY SWARM

GITHUB.COM/JAVAEE-SAMPLES/JAVAEE7-SIMPLE-SAMPLE



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

Watch 4 | Star 12 | 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` 29 days ago

src/main	adding 'all' beans.xml	8 months ago
.gitignore	adding gitignore	11 months ago
README.asciidoc	reorganizing	8 months ago
pom.xml	cleaning up whitespace	29 days ago

README.asciidoc

A simple Java EE 7 Sample

This is a trivial Java EE 7 sample.

1. Download WildFly 8.2 from http://download.jboss.org/wildfly/8.2.0.Final/wildfly-8.2.0.Final.zip and unzip.
2. Start WildFly as: `./bin/standalone.sh`
3. Deploy application WAR to WildFly: `mvn wildfly:deploy`

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs

SSH clone URL

`git@github.com:j`

You can clone with HTTPS, SSH, or Subversion.

Download ZIP

WHAT DID WE CHANGE?

MAVEN PLUGIN

```
<plugin>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-plugin</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>package</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

JAX-RS WITH CDI FRACTION

```
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-jaxrs-weld</artifactId>
</dependency>
```

JAXB FOR JAX-RS FRACTION

```
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-jaxrs-jaxb</artifactId>
</dependency>
```

THAT WAS IT!

TO MAIN OR NOT TO MAIN

No main()

- Default Container created
- Fractions on classpath created with their default config
- Primary user artifact deployed, usually WAR

TO MAIN OR NOT TO MAIN

main()

- Create a Container
- Custom configuration of Fractions
  - Fractions without custom configuration will be set with defaults
- Custom Archive content

CUSTOM MAIN

```
<plugin>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>wildfly-swarm-plugin</artifactId>
  <configuration>
    <mainClass>com.mycompany.myapp.MyMain</mainClass>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```
public static void main(String[] args) throws Exception {
  Container container = new Container();

  container.fraction(new DatasourcesFraction()
              .jdbcDriver("h2", (d) -> {
                  d.driverDatasourceClassName("org.h2.Driver");
                  d.xaDatasourceClass("org.h2.jdbcx.JdbcDataSource");
                  d.driverModuleName("com.h2database.h2");
              })
              .dataSource("MyDS", (ds) -> {
                  ds.driverName("h2");
                  ds.connectionUrl("jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_
                  ds.userName("sa");
                  ds.password("sa");
              })
  );

  // Prevent JPA Fraction from installing it's default datasource fraction
  container.fraction(new JPAFraction()
               .inhibitDefaultDatasource()
               .defaultDatasource("jboss/datasources/MyDS")
  );

  container.start();

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class);
  deployment.addClasses(Employee.class);
  deployment.addAsWebInfResource(
    new ClassLoaderAsset("META-INF/persistence.xml", Main.class.getClassLoader()
      "classes/META-INF/persistence.xml");
  deployment.addAsWebInfResource(
    new ClassLoaderAsset("META-INF/load.sql", Main.class.getClassLoader()), "cla
  deployment.addResource(EmployeeResource.class);
  deployment.addAllDependencies();

  container.deploy(deployment);
```

BUILD

```
> mvn package
```

RUN

```
java -jar myApp-swarm.jar
```

```
mvn wildfly-swarm:run
```

In your IDE

```
.org.wildfly.swarm.Swarm
.com.mycompany.myapp.MyMain
```

TRANSACTIONS AND MICROSERVICES?

- "Transactions should be contained within a single service"
  - "A microservice should be tied to a single database"
- "Atomicity is overrated"
- "Transactions limited scalability"

WHEN TO USE TRANSACTIONS

- When you need ACID semantics!
- Or ...
  - When you have a need to guarantee consensus in the presence of failures
  - When you need isolation and consistency across failures
- Relaxing ACID semantics is possible
- Recoverable transactions may be sufficient

```
@Path("/")
public class MyResource {
  @GET
  @Produces("text/plain")
  public String init() throws Exception {
    return "Active";
  }

  @Path("begincommit")
  @GET
  @Produces("text/plain")
  public String beginCommit() throws Exception {

    UserTransaction txn = (UserTransaction) new InitialContext()
        .lookup("java:comp/UserTransaction");
    String value = "Transaction ";

    try {
      txn.begin();

      value += "begun ok";

      try {
        txn.commit();

        value += " and committed ok";
      } catch (final Throwable ex) {
        value += " but failed to commit";
      }
```

```
public class Main {
    public static void main(String[] args) throws Exception {
        Container container = new Container();

        /*
     * Use specific TransactionFraction even though it doesn't do
         * any more than the default one - for now.
         */

        container.subsystem(new TransactionsFraction(4712, 4713));

        // Start the container

        container.start();

        /*
     * Now register JAX-RS resource class.
         */

        JAXRSArchive appDeployment =
            ShrinkWrap.create(JAXRSArchive.class);

        appDeployment.addResource(MyResource.class);

        container.deploy(appDeployment);
    }
}
```

SOFTWARE TRANSACTIONAL MEMORY

- ACI … no D
- Framework for building transactions
- Using JTA where wanted
- Volatile updates, even shared between multiple services, more appropriate
- Compensations

```
@Optimistic
public class SampleLockable implements Sample
{
  public SampleLockable (int init) {
    _isState = init;
  }

  @ReadLock
  public int value() {
    return _isState;
  }

  @WriteLock
  public void increment() {
    _isState++;
  }

  @WriteLock
  public void decrement() {
    _isState--;
  }

  @State
  private int _isState;
}
```

```
MyExample ex = new MyExample(10);
Container<Sample> theContainer =
    new Container<Sample>();
AtomicAction act = new AtomicAction();

act.begin();

obj1.increment();

act.commit();
```

ROADMAP

- More WildFly subsystems as Fractions
  - Infinispan
  - Camel
- Additional frameworks
  - Spring
- Uber jar diet
- Improve testing of WildFly Swarm based applications
- Additional use of Java Config API in Fractions
- Improved APIs for main()

KEEP IN TOUCH

GitHub

https://github.com/wildfly-swarm

Twitter

@wildflyswarm

IRC

#wildfly-swarm

Website

http://wildfly.org/swarm

User Guide

http://wildfly-swarm.gitbooks.io/wildfly-swarm-users-guide/content/

Interested in feedback and input on direction