Java. Cloud. Leadership.

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

Mark Little, VP

On behalf of the (rest of the) WildFly-Swarm team

Feedback

EAP, AS7, WildFly

- Differentiate the standard from implementations!
  - Bloatware should be a thing of the past
- It is possible to be lightweight and enterprise ready

The Open Source Java application server *reignited*

Designed for flexibility.
Amped with electrifying speed.
Launch your Java EE applications in a flash!

Lightning Fast... *start-up / deployment / configuration*

Standards

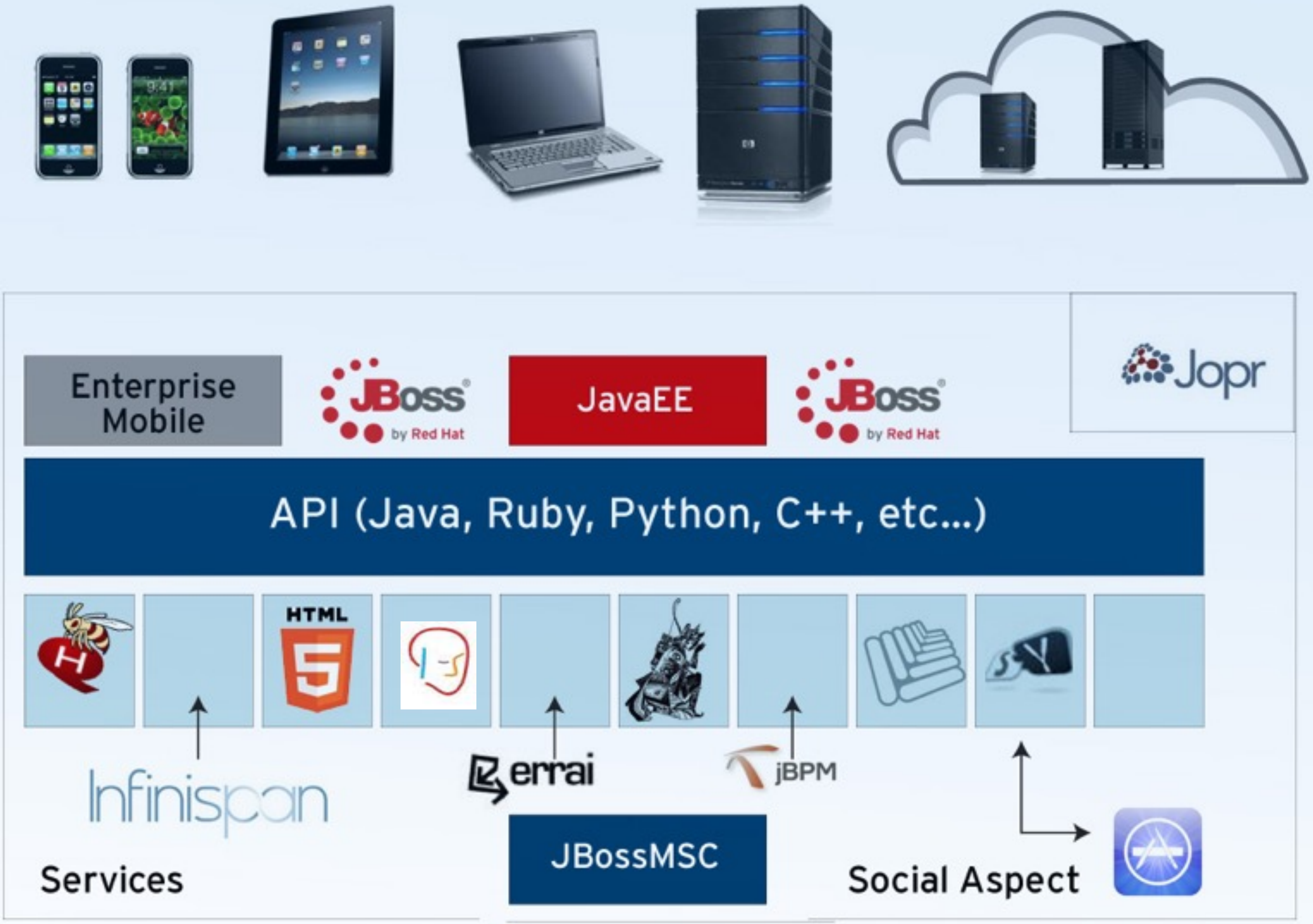
- Transactions (JTA, JTS)
- Persistence (JPA, JDBC)
- Messaging (JMS)
- Security (JAAS, JCE, JSSE, SASL)
- Communication (REST, SOAP, IIOP)
- Cacheing (JSR 107)
- Management (JMX)

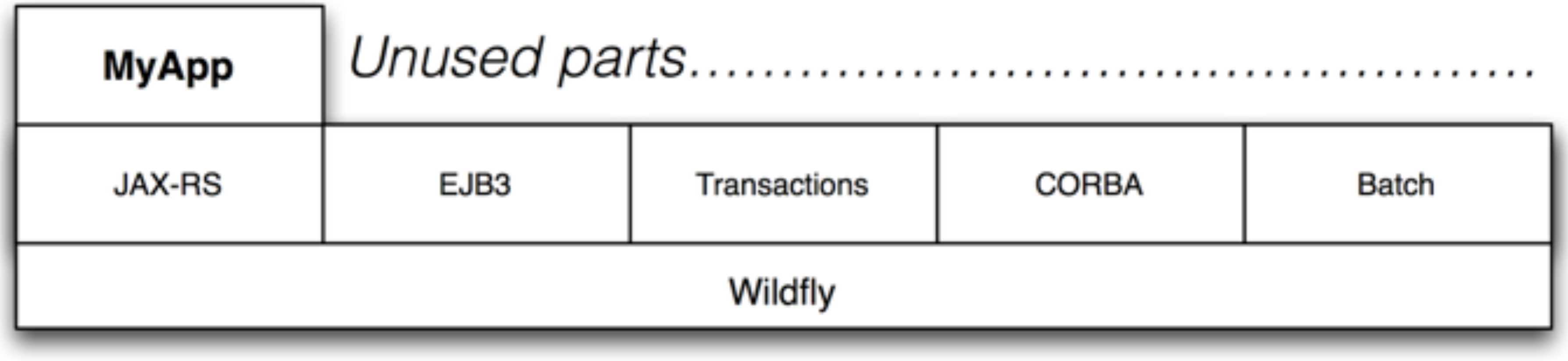
Microservices and Java EE

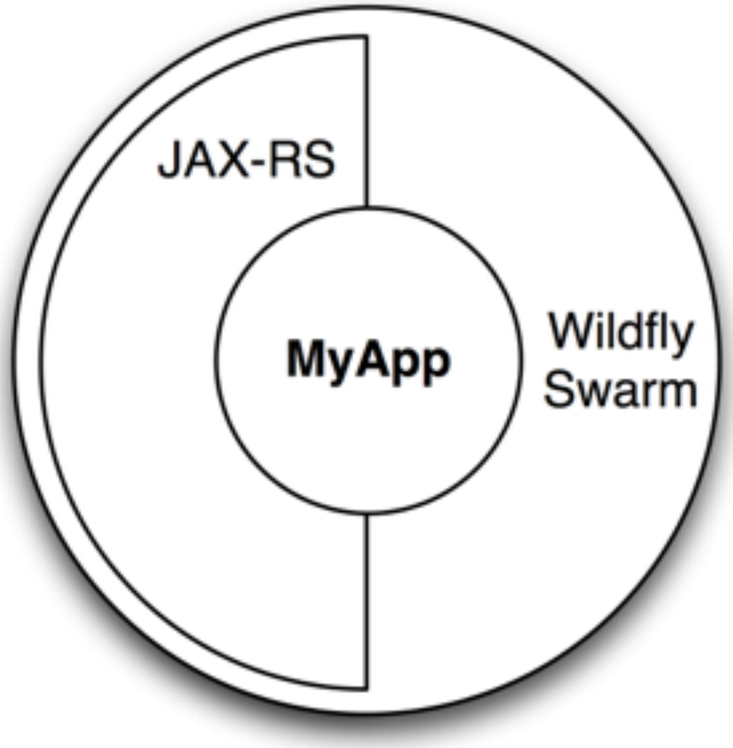
- Not everyone wants to use Docker
- Not everyone wants to use Node.js
- Many developers are happy with Java EE
  - Robust and mature components
  - Scalable, standards compliant, integrates well
- Not everyone wants to use all of Java EE
  - Stripping down EAP/WildFly is common
  - Higher cloud density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

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

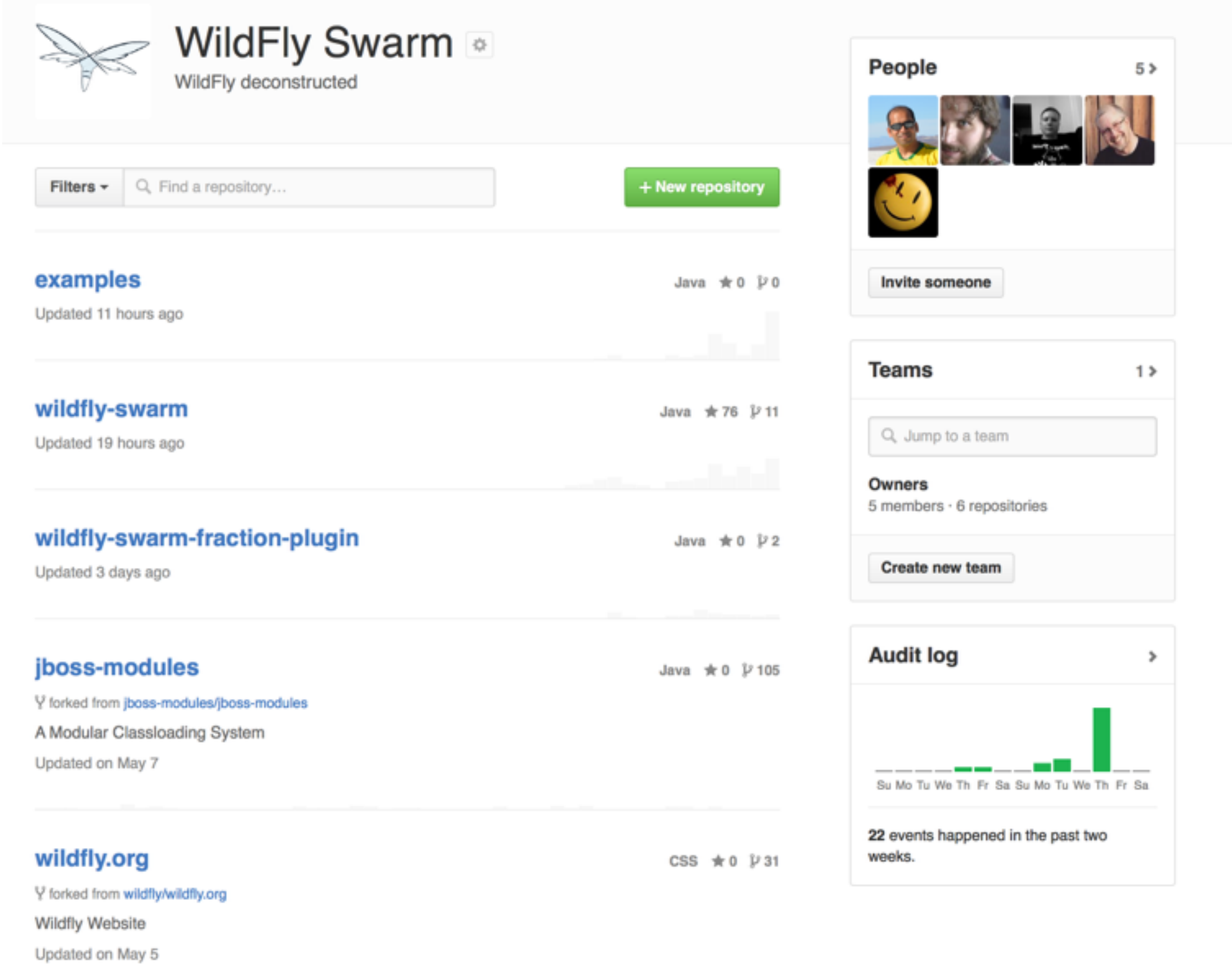

Enterprise Mobile
JBoss by Red Hat
JavaEE
JBoss by Red Hat
Jopr
API (Java, Ruby, Python, C++, etc...)
HTML 5
Infinispan
errai
jBPM
JBossMSC
Services
Social Aspect





myapp-swarm.jar

WORK IN PROGRESS



WildFly Swarm

WildFly deconstructed

examples
Java ★ 0 ⑂ 0
Updated 11 hours ago

wildfly-swarm
Java ★ 76 ⑂ 11
Updated 19 hours ago

wildfly-swarm-fraction-plugin
Java ★ 0 ⑂ 2
Updated 3 days ago

jboss-modules
Java ★ 0 ⑂ 105
forked from jboss-modules/jboss-modules
A Modular Classloading System
Updated on May 7

wildfly.org
CSS ★ 0 ⑂ 31
forked from wildfly/wildfly.org
Wildfly Website
Updated on May 5

People 5

Teams 1

Owners
5 members · 6 repositories

Audit log

22 events happened in the past two weeks.

How does it work?

- Leverages a lot of what is already in WildFly
- Takes a ContentProvider to programmatically deploy content into the container
- New Fraction/Configuration classes for configuration
  - Parallels Subsystems 1-to-1
    - Uses existing ModelNode for passing information
  - Inheritance through maven dependencies
    - The jax-rs fraction implies the undertow fraction, keeping the developer's pom.xml relatively clean
- Tries to minimise developer overhead

Swarm awareness

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

Containers

- container.start()
  - Plugs into WildFly-Core SelfContainedContainer object
  - Passing it the List<ModelNode> that acts like standalone.xml
  - Plus the ContentProvider so we can do programatic content deployments
- container.deploy()
  - Looks for the primary user artefact (a .war, usually) and deploys it via the ContentProvider and a ModelNode to trigger deployment

TransactionsConfiguration

```java
public class TransactionsConfiguration extends AbstractServerConfiguration<TransactionsFraction> {

    public TransactionsConfiguration() {
        super(TransactionsFraction.class);
    }

    @Override
    public TransactionsFraction defaultFraction() {
        return new TransactionsFraction(4712, 4713);
    }

    @Override
    public List<ModelNode> getList(TransactionsFraction fraction) {
        List<ModelNode> list = new ArrayList<>();

        PathAddress address = PathAddress.pathAddress(PathElement.pathElement(SUBSYSTEM, "transactions"));

        ModelNode node = new ModelNode();
        node.get(OP_ADDR).set(EXTENSION, "org.jboss.as.transactions");
        node.get(OP).set(ADD);
        list.add(node);

        node = new ModelNode();
        node.get(OP_ADDR).set(address.toModelNode());
        node.get(OP).set(ADD);
        node.get("socket-binding").set( "txn-recovery-environment");
        node.get("status-socket-binding").set( "txn-status-manager");
```

```java
public class MessagingConfiguration extends AbstractServerConfiguration<MessagingFraction> {

    private PathAddress address = PathAddress.pathAddress(PathElement.pathElement(SUBSYSTEM, "messaging"));

    public MessagingConfiguration() {
        super(MessagingFraction.class);
    }

    @Override
    public MessagingFraction defaultFraction() {
        return new MessagingFraction();
    }

    @Override
    public List<ModelNode> getList(MessagingFraction fraction) {
        List<ModelNode> list = new ArrayList<>();

        ModelNode node = new ModelNode();
        node.get(OP_ADDR).set(EXTENSION, "org.jboss.as.messaging");
        node.get(OP).set(ADD);
        list.add(node);

        node = new ModelNode();
        node.get(OP_ADDR).set(address.toModelNode());
        node.get(OP).set(ADD);
        list.add(node);

        addServers(fraction, list);

        return list;
    }

    protected void addServers(MessagingFraction fraction, List<ModelNode> list) {
        List<MessagingServer> servers = fraction.servers();

        for (MessagingServer each : servers) {
            addServer(each, list);
        }
    }
```

eadership.

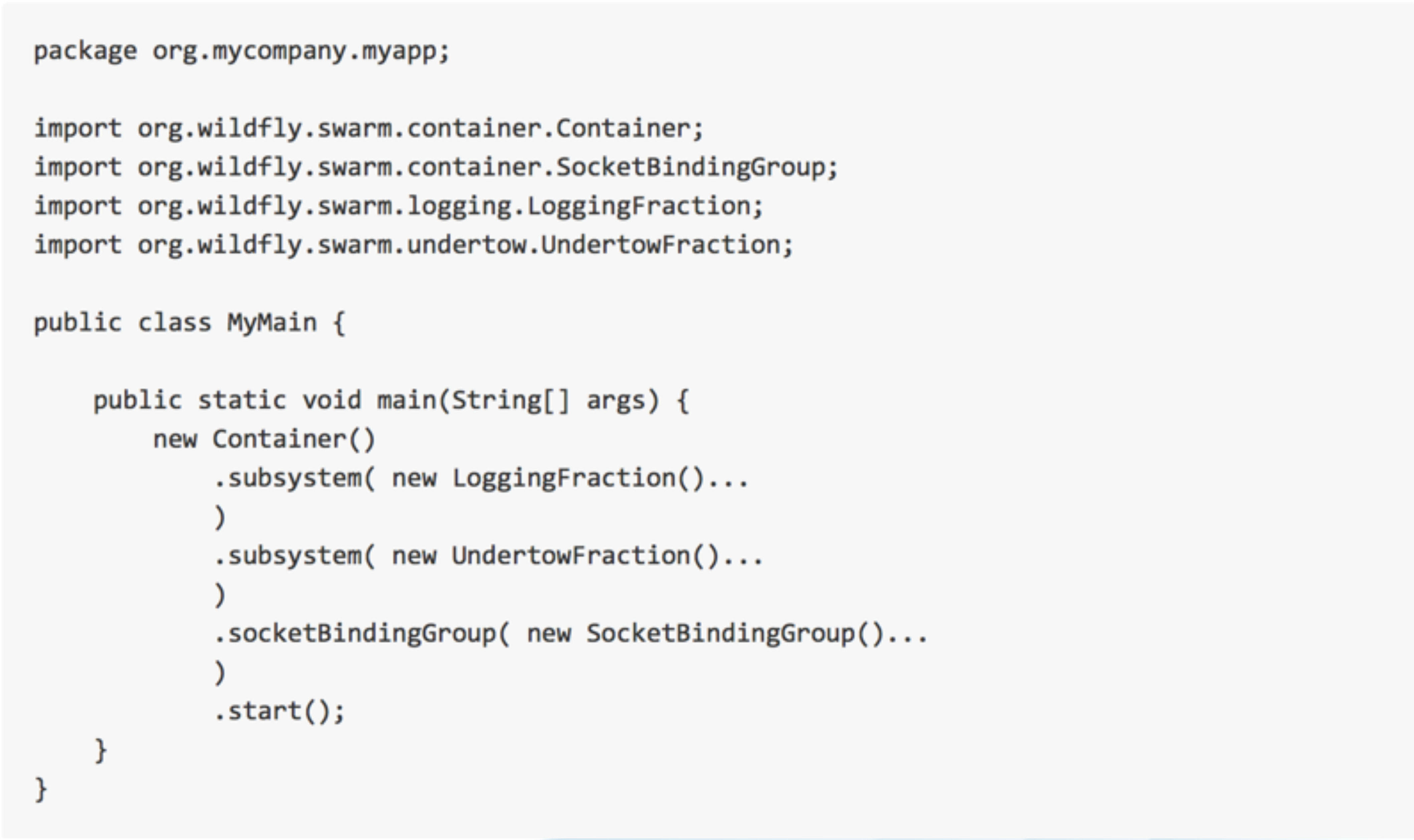
To main or not to main

- main() not a big player in server-side Java EE
- Much is defaulted to ease developers burden
- If you have no main() then a default Container is created and every fraction is defaulted
- If you provide a main() you can configure any fractions
  - Any not explicitly configured will have defaults
- Could also do a lot more in main()
  - Locate other services?
  - Dynamically adapt components?

Some control

```
package org.mycompany.myapp;

import org.wildfly.swarm.container.Container;
import org.wildfly.swarm.logging.LoggingFraction;

public class MyMain {

    public static void main(String[] args) {
        new Container()
            .subsystem( new LoggingFraction()...
            )
            .start();
    }
}
```

Complete control



```java
package org.mycompany.myapp;

import org.wildfly.swarm.container.Container;
import org.wildfly.swarm.container.SocketBindingGroup;
import org.wildfly.swarm.logging.LoggingFraction;
import org.wildfly.swarm.undertow.UndertowFraction;

public class MyMain {

    public static void main(String[] args) {
        new Container()
            .subsystem( new LoggingFraction()...
            )
            .subsystem( new UndertowFraction()...
            )
            .socketBindingGroup( new SocketBindingGroup()...
            )
            .start();
    }
}
```

```java
public class Main {

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

        container.subsystem(new MessagingFraction()
                        .server(
                                new MessagingServer()
                                        .enableInVMConnector()
                                        .topic("my-topic")
                                        .queue("my-queue")
                        )
        );

        // Start the container
        container.start();

        JAXRSDeployment appDeployment = new JAXRSDeployment(container);
        appDeployment.addResource(MyResource.class);

        // Deploy your app
        container.deploy(appDeployment);

        ServiceActivatorDeployment deployment = new ServiceActivatorDeployment(container);
        deployment.addServiceActivator(MyServiceActivator.class);
        deployment.addClass(MyService.class);

        // Deploy the services
        container.deploy(deployment);

    }
}
```

Specifying the main class

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
        <mainClass>com.mycompany.myapp.MyMain</addClasspath>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

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()
- In general run with …
  - java -jar myfatjar-swarm.jar
  - mvn wildfly-swarm:run
  - In your IDE run the main class

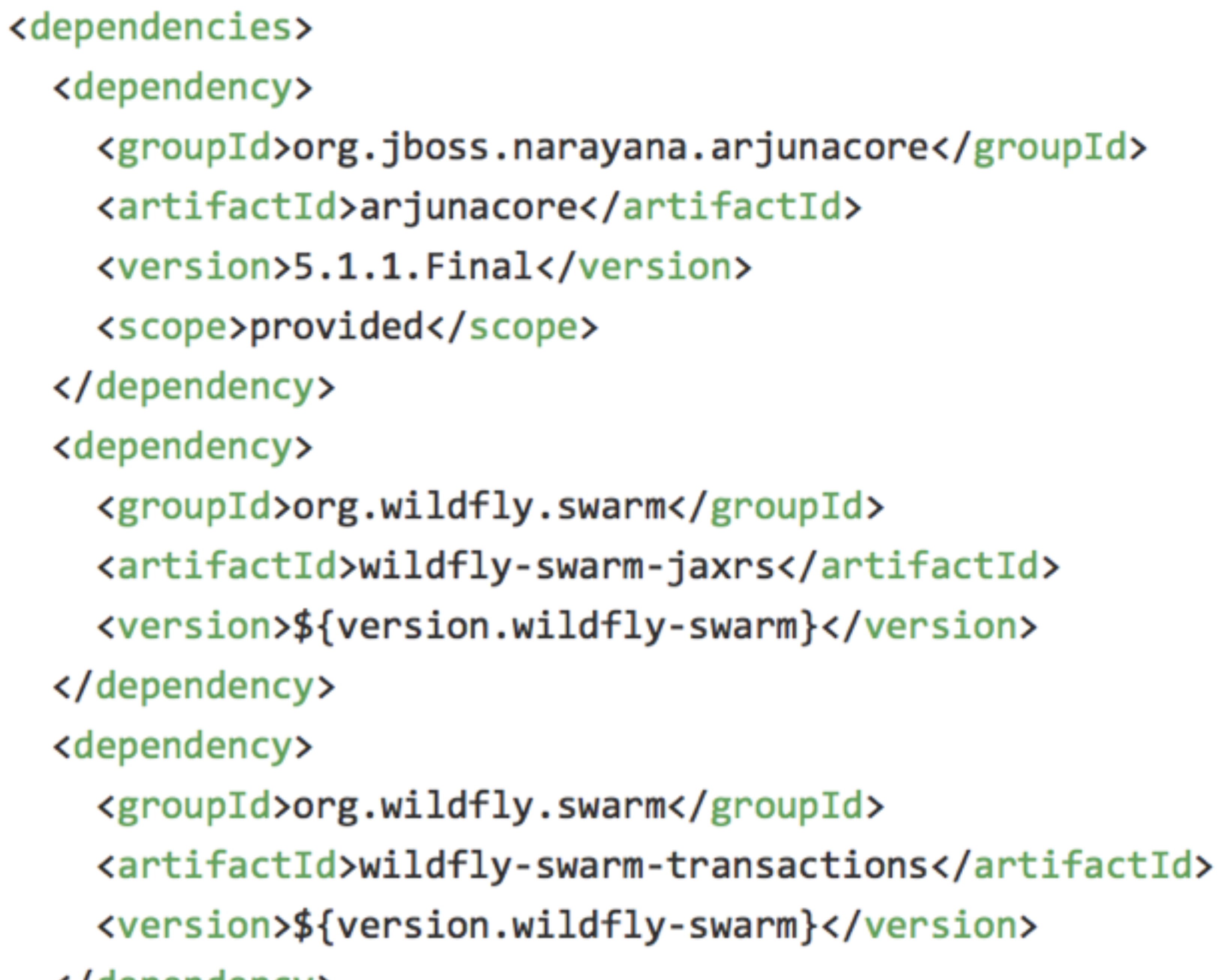
Transactions and JAX-RS

- Basic example showing JTA and JAX-RS
- Defines JAX-RS resource
- Defines its own Main.java
  - Configures transactions explicitly
  - Defaults everything else
- 82 Meg fat jar produced

```
@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";
            }
```

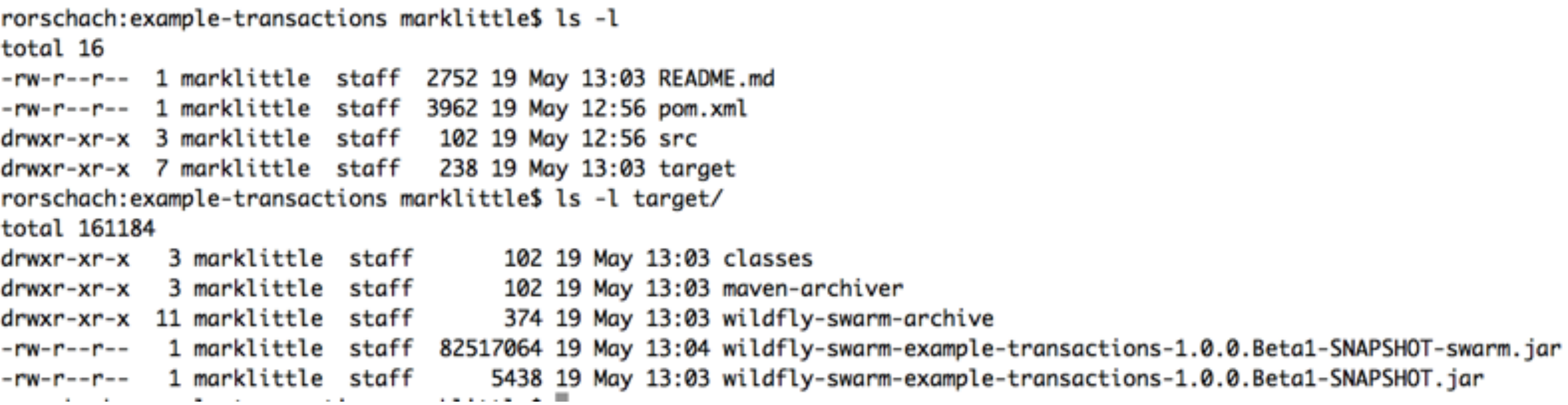
```java
package org.wildfly.swarm.examples.transactions;

import org.wildfly.swarm.container.Container;
import org.wildfly.swarm.jaxrs.JAXRSDeployment;
import org.wildfly.swarm.transactions.TransactionsFraction;

/**
 * @author nmcl
 */

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.
         */

        JAXRSDeployment appDeployment = new JAXRSDeployment(container);
        appDeployment.addResource(MyResource.class);

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

wildfly-swarm dependencies



```
<dependencies>
  <dependency>
    <groupId>org.jboss.narayana.arjunacore</groupId>
    <artifactId>arjunacore</artifactId>
    <version>5.1.1.Final</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-jaxrs</artifactId>
    <version>${version.wildfly-swarm}</version>
  </dependency>
  <dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>wildfly-swarm-transactions</artifactId>
    <version>${version.wildfly-swarm}</version>
```

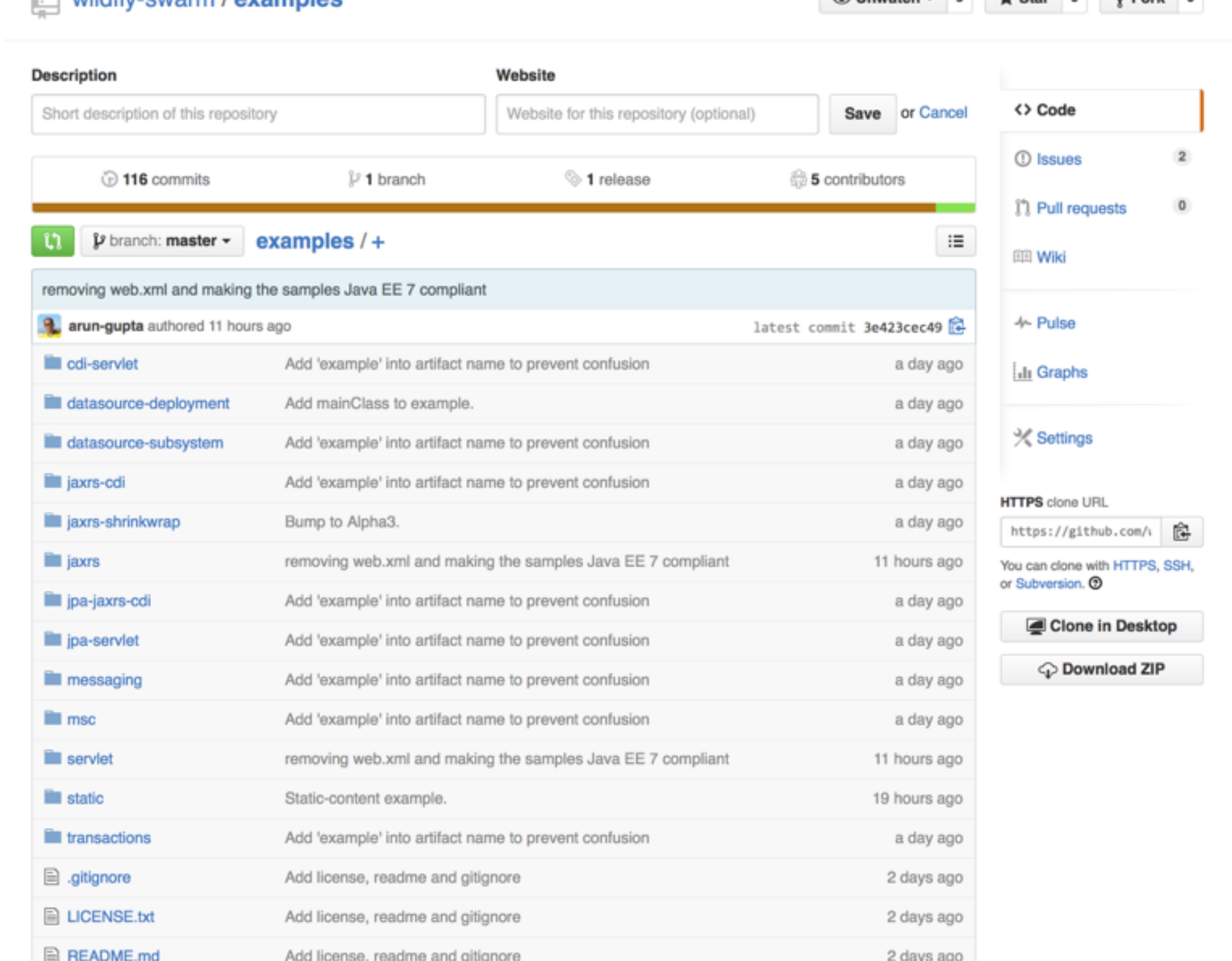
The results



```
rorschach:example-transactions marklittle$ ls -l
total 16
-rw-r--r--  1 marklittle  staff  2752 19 May 13:03 README.md
-rw-r--r--  1 marklittle  staff  3962 19 May 12:56 pom.xml
drwxr-xr-x  3 marklittle  staff   102 19 May 12:56 src
drwxr-xr-x  7 marklittle  staff   238 19 May 13:03 target
rorschach:example-transactions marklittle$ ls -l target/
total 161184
drwxr-xr-x   3 marklittle  staff       102 19 May 13:03 classes
drwxr-xr-x   3 marklittle  staff       102 19 May 13:03 maven-archiver
drwxr-xr-x  11 marklittle  staff       374 19 May 13:03 wildfly-swarm-archive
-rw-r--r--   1 marklittle  staff  82517064 19 May 13:04 wildfly-swarm-example-transactions-1.0.0.Beta1-SNAPSHOT-swarm.jar
-rw-r--r--   1 marklittle  staff      5438 19 May 13:03 wildfly-swarm-example-transactions-1.0.0.Beta1-SNAPSHOT.jar
```

Other examples

- JAX-RS
  - CDI
  - JPA & CDI
  - Shrinkwrap (no war)
- Messaging
- Servlet
  - CDI
  - JPA
- Transactions
  - STM (to come)



wildfly-swarm / examples

Description

Short description of this repository

Website

Website for this repository (optional)

Save or Cancel

116 commits | 1 branch | 1 release | 5 contributors

branch: master | examples / +

removing web.xml and making the samples Java EE 7 compliant

arun-gupta authored 11 hours ago — latest commit 3e423cec49

cdi-servlet	Add 'example' into artifact name to prevent confusion	a day ago
datasource-deployment	Add mainClass to example.	a day ago
datasource-subsystem	Add 'example' into artifact name to prevent confusion	a day ago
jaxrs-cdi	Add 'example' into artifact name to prevent confusion	a day ago
jaxrs-shrinkwrap	Bump to Alpha3.	a day ago
jaxrs	removing web.xml and making the samples Java EE 7 compliant	11 hours ago
jpa-jaxrs-cdi	Add 'example' into artifact name to prevent confusion	a day ago
jpa-servlet	Add 'example' into artifact name to prevent confusion	a day ago
messaging	Add 'example' into artifact name to prevent confusion	a day ago
msc	Add 'example' into artifact name to prevent confusion	a day ago
servlet	removing web.xml and making the samples Java EE 7 compliant	11 hours ago
static	Static-content example.	19 hours ago
transactions	Add 'example' into artifact name to prevent confusion	a day ago
.gitignore	Add license, readme and gitignore	2 days ago
LICENSE.txt	Add license, readme and gitignore	2 days ago
README.md	Add license, readme and gitignore	2 days ago

HTTPS clone URL

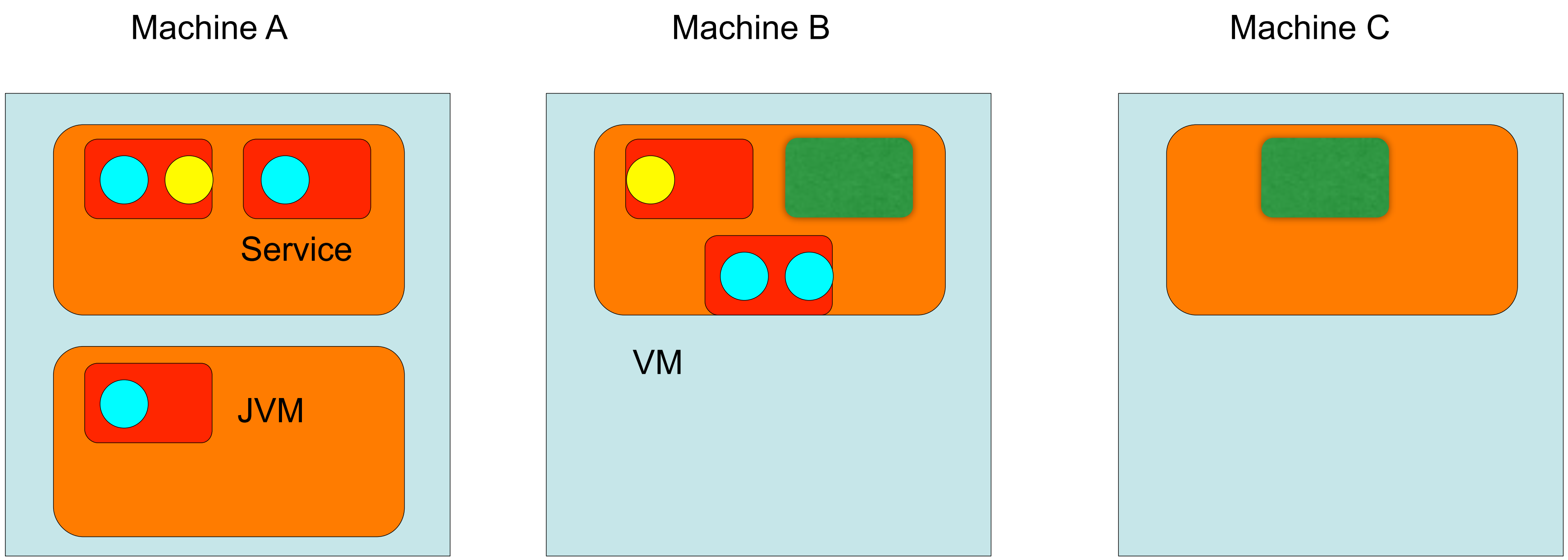
https://github.com/

You can clone with HTTPS, SSH, or Subversion.

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 Linux containers

Services, Linux containers and JVMs



- Java EE services split across machines, containers and JVMs

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)

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