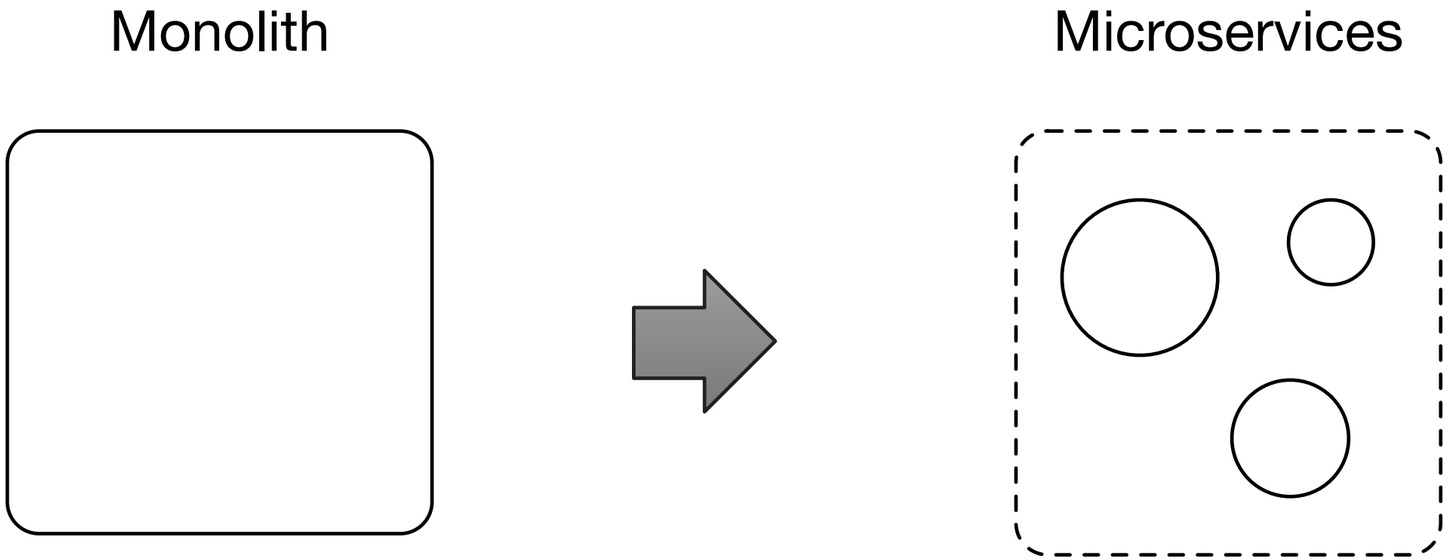
“Right Size” your Services with WildFly Swarm

Heiko Braun
Dimitris Andreadis

Red Hat

Monoliths and microservices

Either/or?



DEVOXX™

"There is always a well-known solution to every human problem - neat, plausible, and *wrong*"

– Mencken, H. L. (1920) *Prejudices: Second Series*

What's in a name?

- Actually it's monolithic architectures and microservices architectures
- Rather than, the "monolith" and a "microservice"

DEVOXX™

"Software Architecture: the fundamental organization of a system embodied in its **components**, their **relationships** to each other and to the environment and the **principles** guiding its design and evolution."

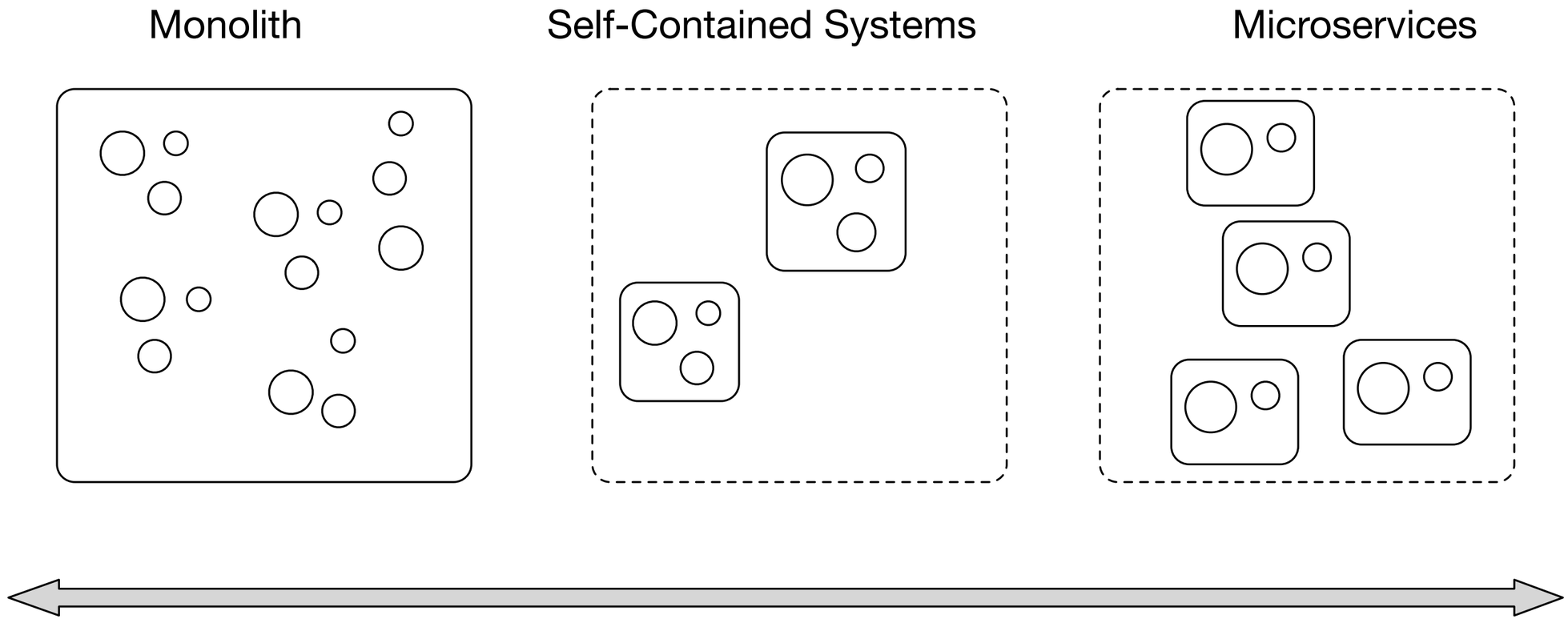
–IEEE in their standard IEEE 1471
(which was later adopted by ISO/IEC 42010).

Architectural properties

	Monolithic architecture	Microservice architecture
Components	Few	Many
Relationships	Space: co-located Time: change together	Space: distributed Time: change independently
Principles	Uniformity	Diversity

(This is a non exhaustive list. Add your own ideas)

Continuum of architectural choices



Coming from Java EE

- Coming from EE, you probably sit more to the left of the continuum
  - The monolithic side more closely resembles the ideas and assumptions in Java EE
  - and the ways application servers have been build and used

DEVOXX™

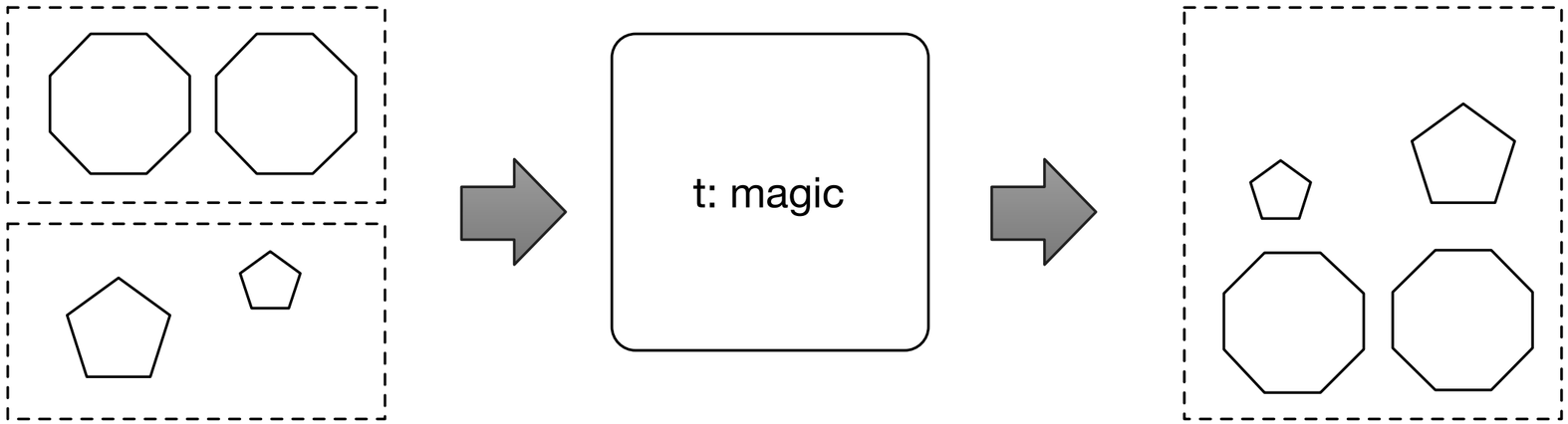
WildFly SWARM

Meet WildFly Swarm

The basic idea

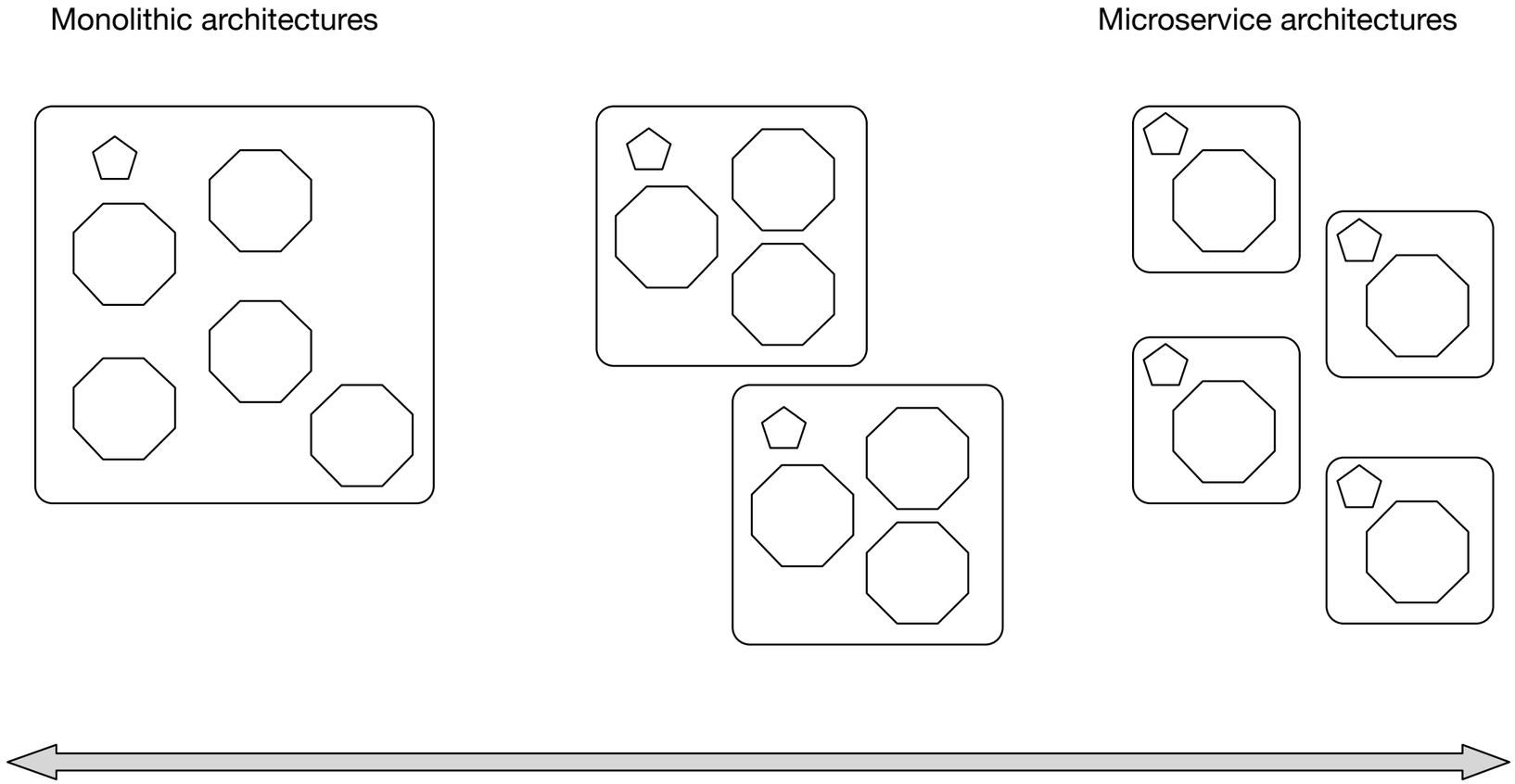
i: runtime capability

i: application code



o: self contained binary

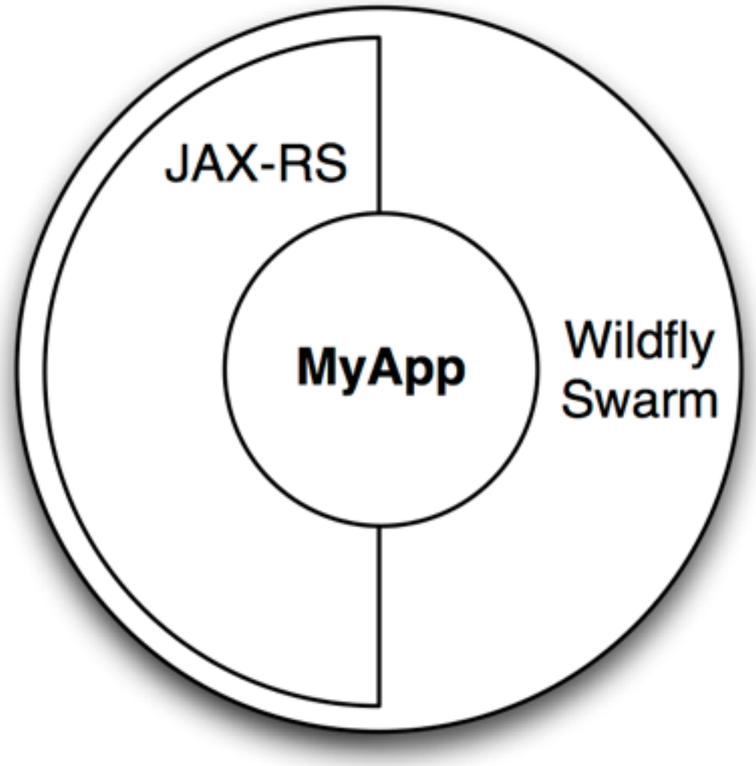
“Right Size” the runtime



Self-contained (Uber) jar

- bundles your application
- the *Fractions* to support it
- an internal maven repo with the dependencies
- bootstrap code

- There is also the notion of a *Hollow* launch-pad type of jar.

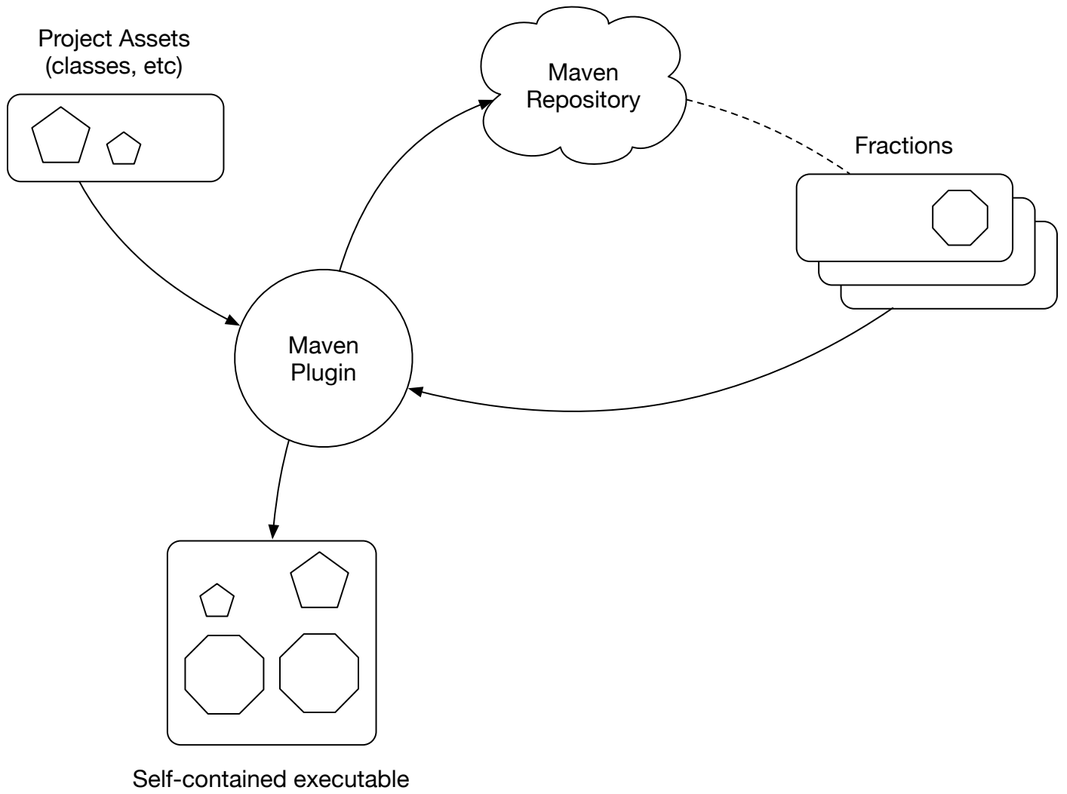


The base WildFly Runtime

- Modular Architecture
- Concurrent Kernel
- Fast & Lightweight
- Cloud friendly

Concept of a Fraction

- A tangible unit, embodied in a maven artefact
  - To support the compositional aspect in Swarm
- Provides the "runtime" capabilities
- Means to configure the system
  - With reasonable defaults



Fraction use cases

- Fractions support explicit and implicit configuration
  - In many cases you won't need to configure anything
- Fractions can be detected or explicitly declared
  - The most simple case is a <war> project, with just the maven plugin
- All of EE is supported in Swarm:
  - JPA, JAX-RS, EJB, JMS, …

Enabling WildFly Swarm

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

```
    <!-- WildFly Swarm Fractions -->
    <dependency>
      <groupId>org.wildfly.swarm</groupId>
      <artifactId>jaxrs</artifactId>
    </dependency>
  </dependencies>
/project>
```

Build / Run

- Build :
  - mvn package

- Run :
  - mvn wildly-swarm:run
  - java -jar <my-app>-swarm.jar
  - IDE > Run … MyMain()
  - IDE > Run … o.w.s.Swarm()

Taking control of main()

```
package com.example.demo;

import org.wildfly.swarm.Swarm;
import org.wildfly.swarm.jaxrs.JAXRSFraction;

public class DemoMain {
    public static void main(String[] args) throws Exception {
        Swarm swarm = new Swarm();

        // create and configure fractions
        swarm.fraction(new JAXRSFraction());

        // start the container and deploy fractions
        swarm.start().deploy();
    }
}
```

Fraction Configuration

```
public static void main(String[] args) throws Exception {

    Swarm swarm = new Swarm();
    swarm.fraction(
            new DatasourcesFraction()
                    .jdbcDriver("h2", (d) -> {
                        d.driverClassName("org.h2.Driver");
                        d.xaDatasourceClass("org.h2.jdbcx.JdbcDataSource");
                        d.driverModuleName("com.h2database.h2");
                    })
                    .dataSource("ExampleDS", (ds) -> {
                        ds.driverName("h2");
                        ds.connectionUrl("...");
                        ds.userName("sa");
                        ds.password("sa");
                    });
    );

    swarm.start().deploy();

}
```

DEVOXX™

Moving further to the right …

Shifting complexities

- You separate out the components the complexity moves elsewhere, i.e.
  - In monolithic architectures you have to coordinate the updates to the software prior to releasing it
  - In microservices architectures you have to manage a multitude of distributed services running in production
- It's not about "right" or "wrong", it's about "better" or "worse" (according to your circumstances)

Organisational Competencies

- M. Fowler [1] identifies a set of baseline competencies:
  - Rapid provisioning
  - Monitoring
  - Rapid application deployment
- "These competencies *should be universally present* across software organisations"

[1] http://martinfowler.com/bliki/MicroservicePrerequisites.html

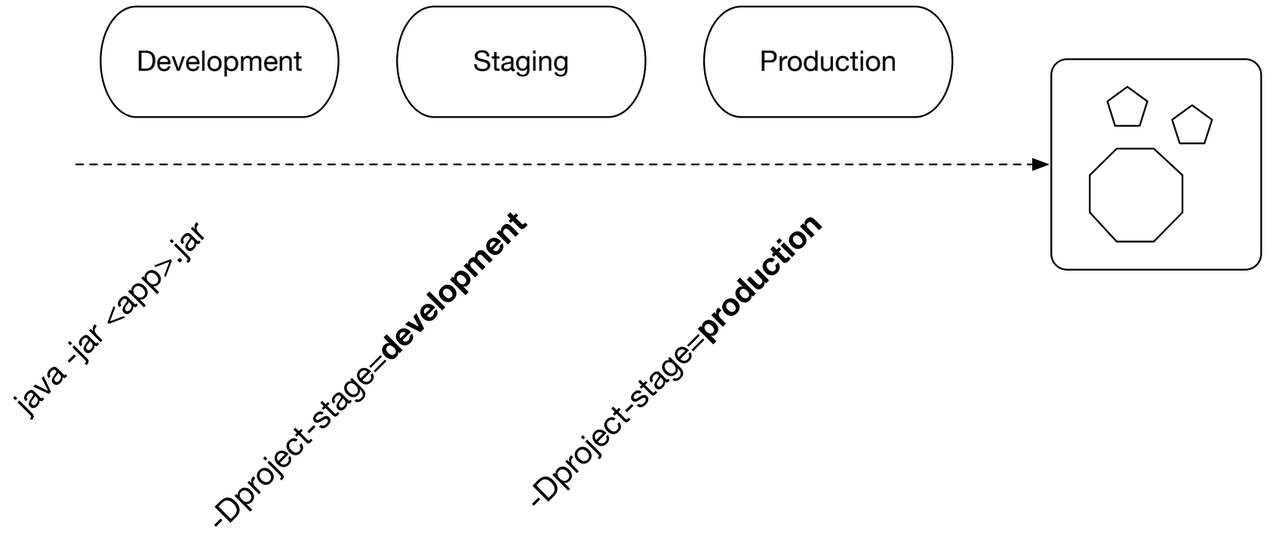
Software requirements

- The actual scope of system components extends beyond Swarm
  - i.e. Cloud infrastructure, CI/CD, etc
- But tools like Swarm can support new architectures:
  - Extending the functional scope
  - Providing new integrations (libraries, 3rd party systems ,etc)
  - Extending the programming models
  - Supporting new operational requirements
- At some point however, this however means going beyond Java EE …

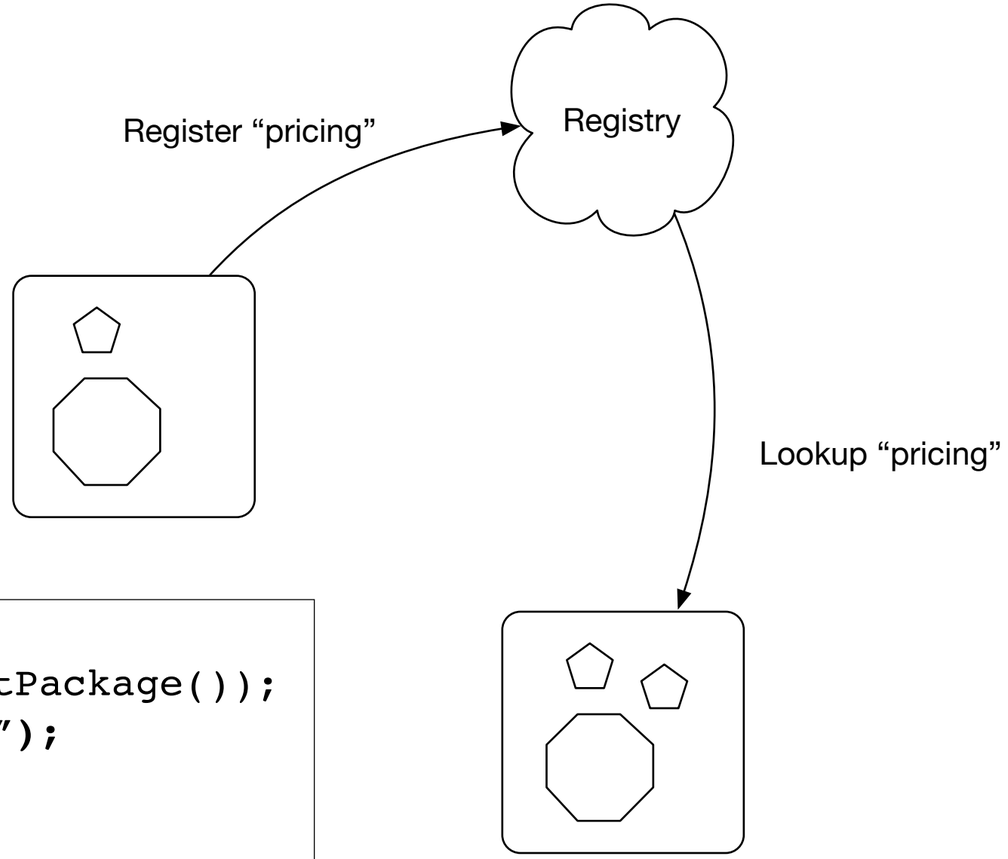
Environment specific configuration

```
logger:
    level: DEBUG
database:
  jdbc:
    url: foo
---
project:
    stage: development
logger:
    level: DEBUG
database:
  jdbc:
    url: bar
---
project:
    stage: production
logger:
    level: INFO
database:
  jdbc:
    url: somethingElse
```

project-stages.yml



Service registration




```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>topology-consul</artifactId>
</dependency>
```

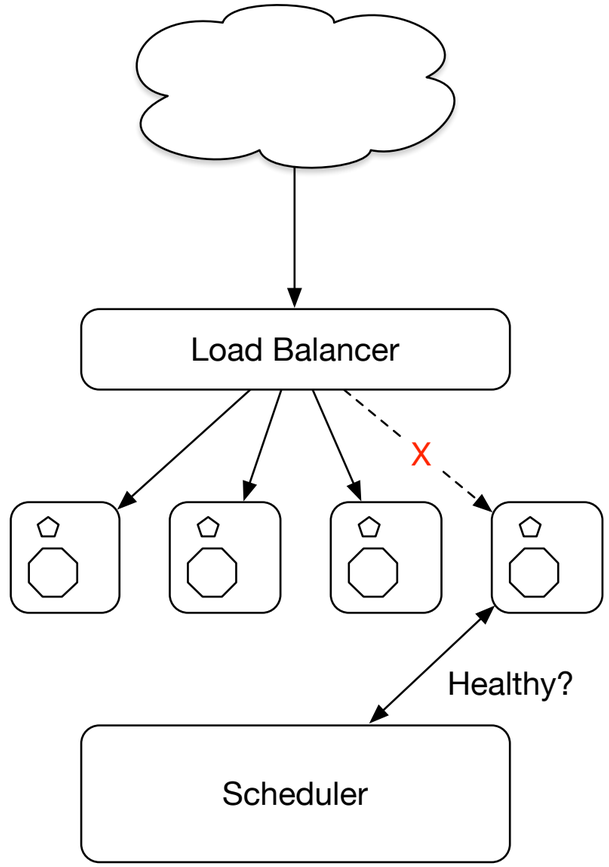
```
JAXRSArchive deployment = …;
deployment.addPackage(Main.class.getPackage());
deployment.as(…).advertise(“pricing”);
deployment.addAllDependencies();

container.deploy(deployment);
```

Health checks

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>monitor</artifactId>
</dependency>
```

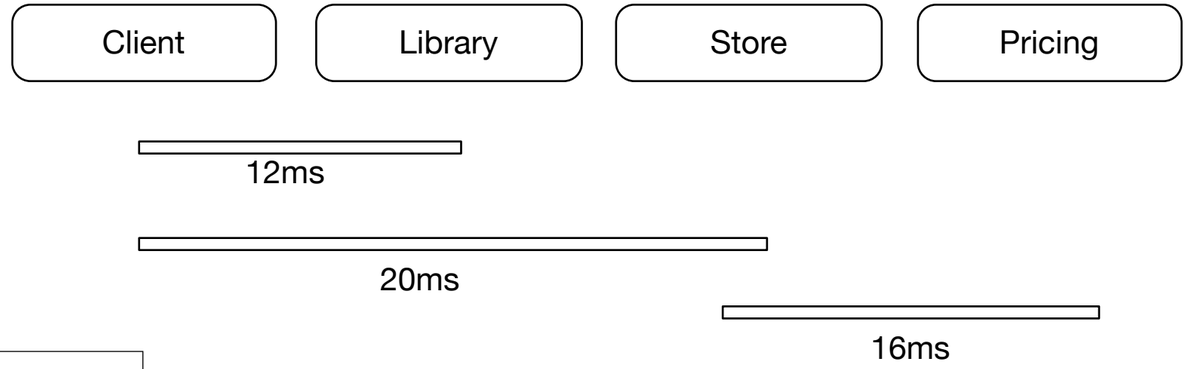
```
@GET
@Path("/other")
@Health
public HealthStatus checkSomethingElse() {
    return HealthStatus.up();
}
```



Distributed Tracing

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>zipkin</artifactId>
</dependency>
```

```
container.fraction(
  new ZipkinFraction(“pricing”)
   .reportAsync("http://localhost:9411/api/v1/spans")
  .sampleRate(0.1f) // keep 10%
);
```



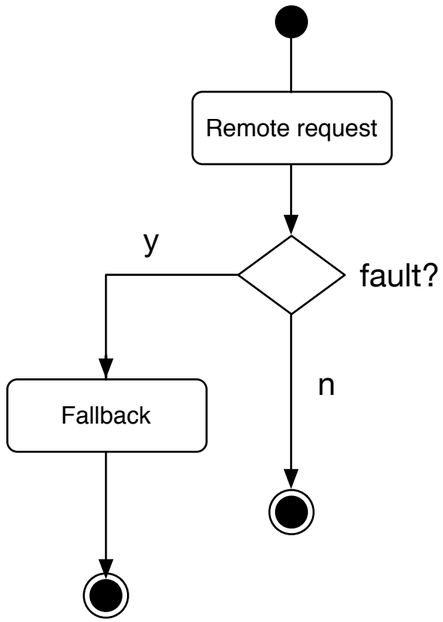
Resilience

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>hystrix</artifactId>
</dependency>
```

```
@CircuitBreaker(fallbackMethod = "myFallback")
public String isolatedCommand() {
  Client client = ClientBuilder.newClient();
  WebTarget target = client.target(“pricing”);

 Response response = target.get();
 return response.readEntity(String.class);
}

public String myFallback() {
  return ..; // cached values
}
```



There is much more it

- Logstash/Fluentd
- Netflix Ribbon
- Various service registries
- Openshift / Kubernetes Integration
- Swagger API Docs
- Vert.x Integration
- Jolokia
- Infinispan
- Remote Management
- ActiveMQ Integration
- SSO
- Contract-Based Testing
- …

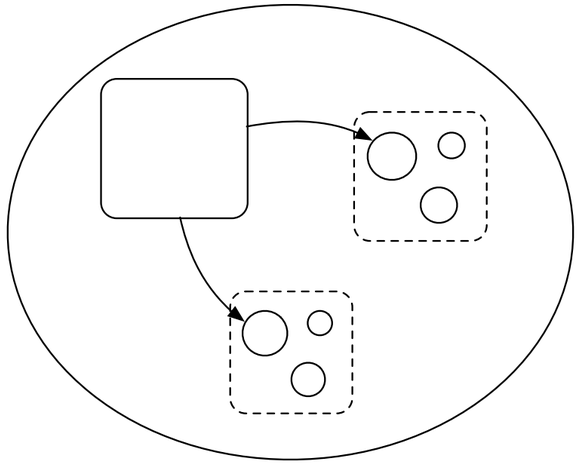
Relation to the MicroProfile

- Feedback into the MicroProfile
- Look for consensus
- Aim for standardisation

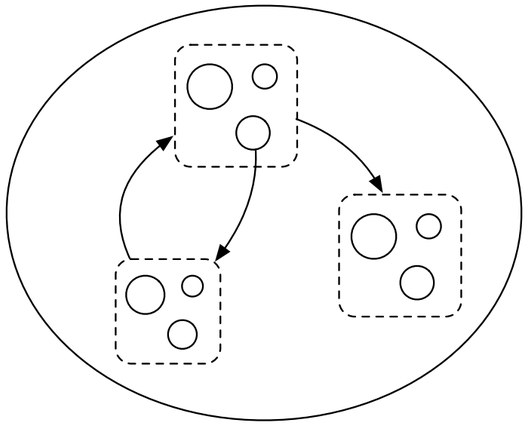
MicroProfile

Spectrum of possibilities

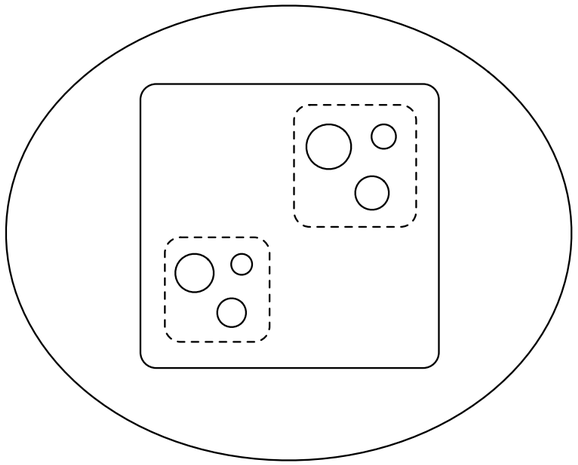
Monolith & microservices



All microservices



Self-Contained systems



DEVOXX™

Thanks!

Visit **http://wildfly-swarm.io** for more information

Join us on IRC: @wildfly-swarm at freenode.net