Microservices
and how
WildFly Swarm Can Play a Part

Ken Finnigan & Bob McWhirter
Red Hat

Who are you?

- **Ken Finnigan**
  - Project lead for WildFly Swarm
  - Contributor to MicroProfile
  - Prolific Author
  - Australian
- **Bob McWhirter**
  - Project lead for WildFly Swarm
  - Tall
  - Not Australian

redhat.

Microservices

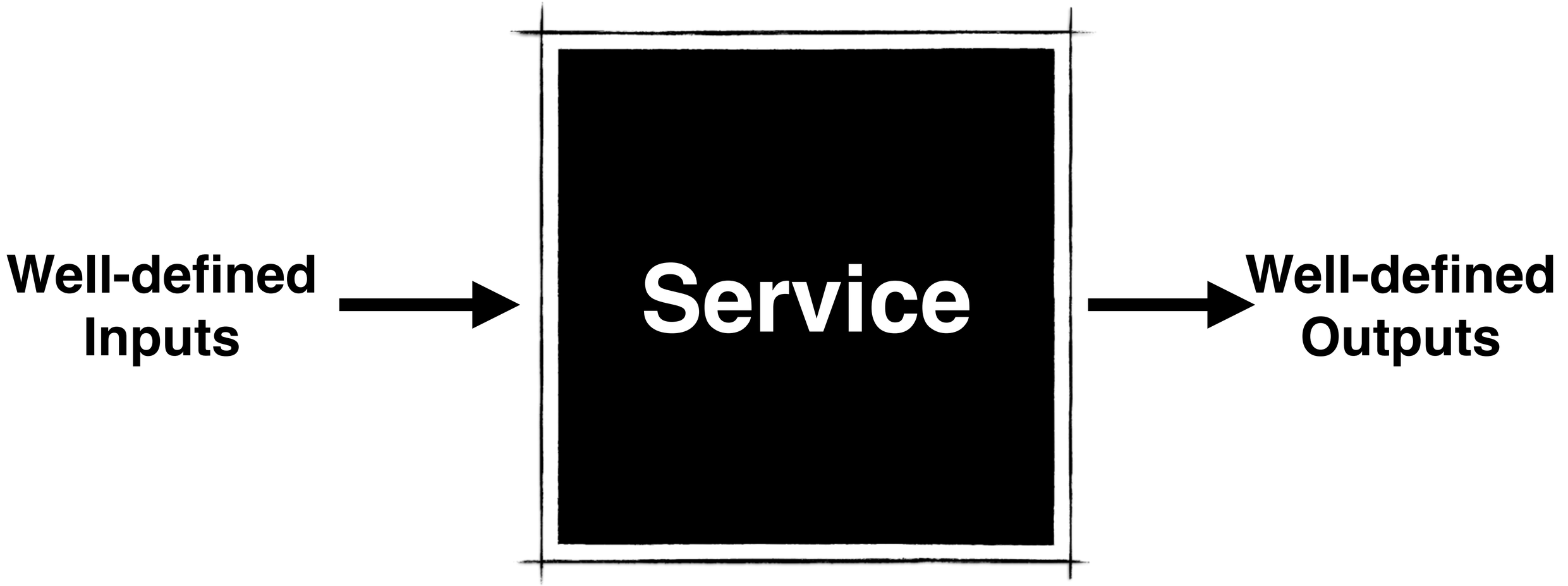
The Promise

- Helps define architecture
- Independent release cycles
- **Accelerate business velocity**

All of this requires discipline

Bounded Context

- Each service does **one thing** well,with well-defined bounds
- If it used to be a library, it’s probably a service now


Well-defined
Inputs
Service
Well-defined
Outputs

Loose Coupling

- Loosely coupled implementations
- Loosely coupled locations

Which leads us to …

Conway's Law

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"

–Melvin Conway

A

monolithic **organization**

produces

monolithic **software**

Which leads us to …

The Two Pizza Rule

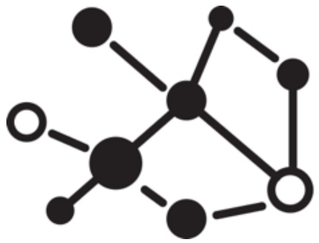
A meeting should never have so many attendees that they could not all be fed with **two pizzas**.

Generally, this limits the number of attendees at a meeting to **less than eight**.

Therefore, your bounded-context should be something solvable by less than 8 people…

including the Product Manager.

For instance…

From the  MicroProfile example

Sessions

Speakers

Schedules

Votes

but also…

Security
Logging
Discovery
Monitoring

Caveats…

If you failed at **SOA**,
you're probably going to fail at
microservices.

If you failed at **SOA**,
you're probably going to fail at
microservices.

If you failed at **SOA**,
you're probably going to fail at
microservices.

Monthly/Weekly/Hourly
releases
of 200 microservices
is not easy

Things that can help

"Containers"

(Linux Containers, Uberjars)

Because what’s tested is exactly what should be deployed

CI/CD pipelines

If you’re deploying continuously, you should be building continuously

"Cloud"

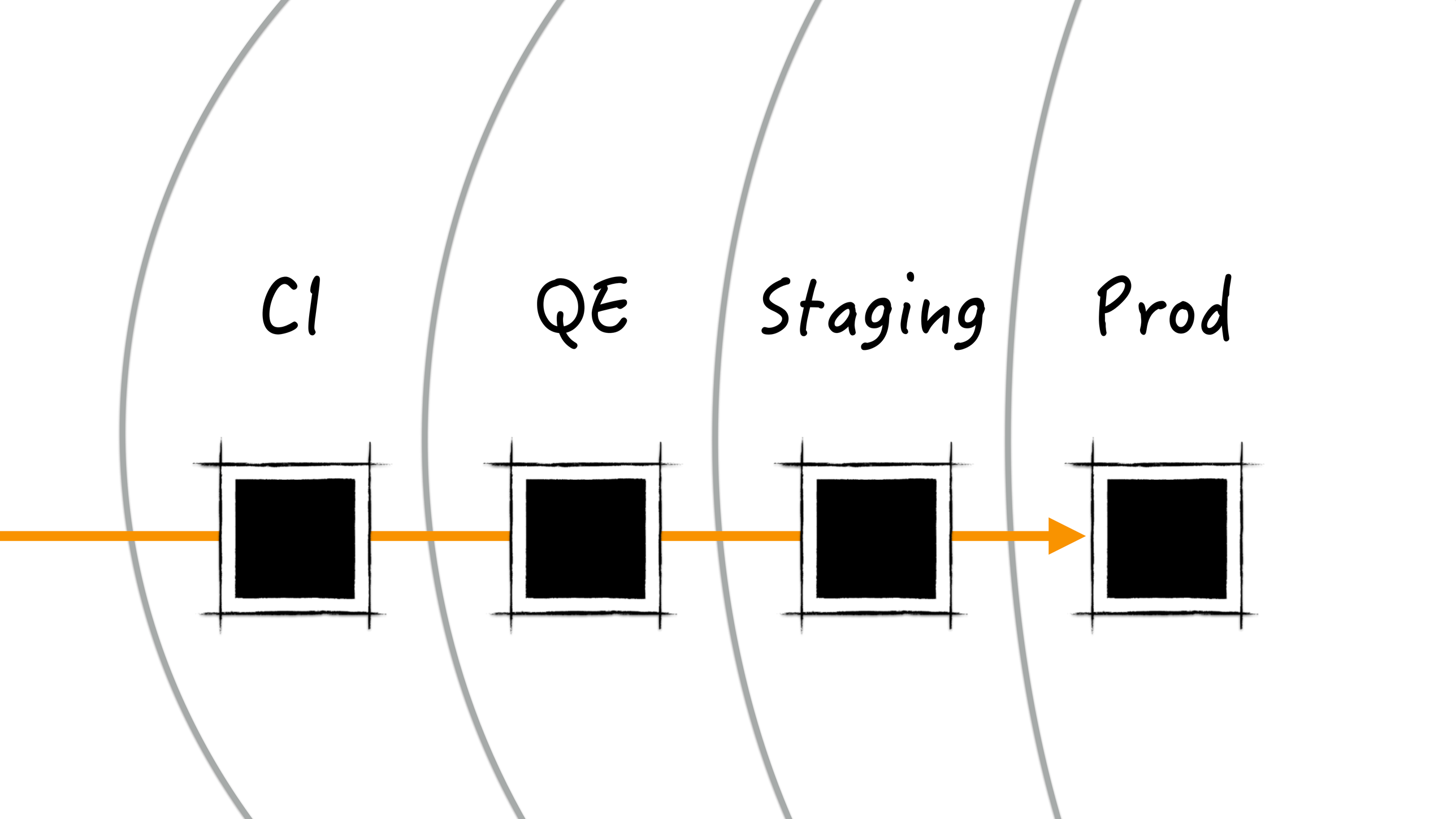
(public or private)

If you’re deploying continuously, automated provisioning is useful

And things like OpenShift can provide cross-cutting functionality

Discovery, Failover, Logging

Monitoring, Provisioning


CI
QE
Staging
Prod

So you want to write a microservice…

- You don’t have to move to Node.js
- Or something “reactive” (but it might be useful)
- You’re a **JavaEE** developer as old as Bob (forty some-odd years old)


WildFly SWARM

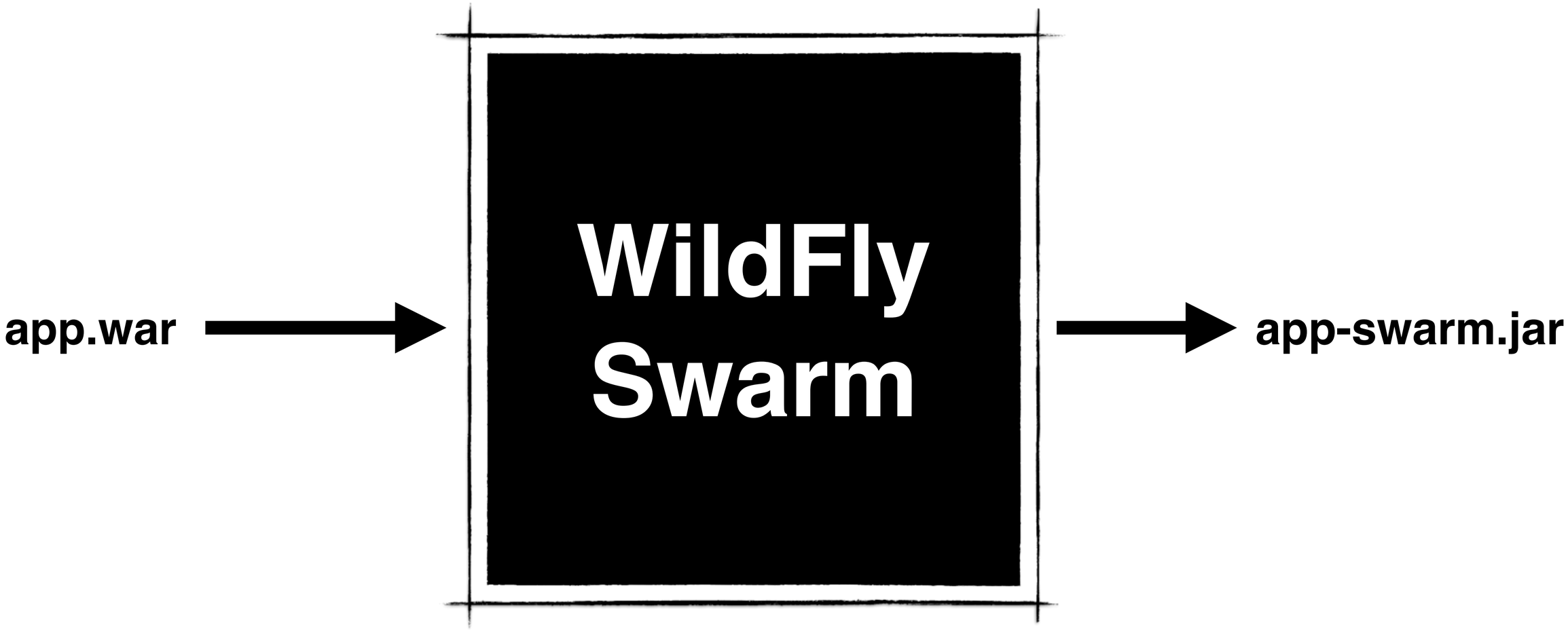
It’s like WildFly

just swarmier

Something with a JSON API?

Why include EJB, JSF?

WildFly Swarm allows you to wrap just the bits of a normal JavaEE app-server you need, ***around your app***.


app.war
WildFly
Swarm
app-swarm.jar

Run it!

```
$ java -jar myapp-swarm.jar
```

warning: maven

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

WildFly Swarm figures out what bits you *need*

- Servlet
- JAX-RS
- JMS
- CDI
- Batch
- JPA
- JAAS
- Transactions
- Naming
- Bean Validation
- Resource Adapters
- Java FX
- JMX
- More…

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

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>bom</artifactId>
  <version>${version.wildfly-swarm}</version>
  <type>pom<type>
  <scope>import</scope>
</plugin>
```

bom

bom-unstable

bom-deprecated

bom-experimental

bom-frozen

Configuration?

standalone.xml

Java API

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

  Swarm swarm = new Swarm();
  swarm.fraction(
    LoggingFraction.createDebugLoggingFraction()
  );

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class, "myapp.war");
  deployment.addClass(MyResource.class);
  deployment.addClass(NotFoundExceptionMapper.class);
  deployment.addAllDependencies();
  swarm.start().deploy(deployment);
}
```

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

  Swarm swarm = new Swarm();
  swarm.fraction(
    LoggingFraction.createDebugLoggingFraction()
  );

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class, "myapp.war");
  deployment.addClass(MyResource.class);
  deployment.addClass(NotFoundExceptionMapper.class);
  deployment.addAllDependencies();
  swarm.start().deploy(deployment);
}
```

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

  Swarm swarm = new Swarm();
  swarm.fraction(
    LoggingFraction.createDebugLoggingFraction()
  );

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class, "myapp.war");
  deployment.addClass(MyResource.class);
  deployment.addClass(NotFoundExceptionMapper.class);
  deployment.addAllDependencies();
  swarm.start().deploy(deployment);
}
```

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

  Swarm swarm = new Swarm();
  swarm.fraction(
    LoggingFraction.createDebugLoggingFraction()
  );

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class, "myapp.war");
  deployment.addClass(MyResource.class);
  deployment.addClass(NotFoundExceptionMapper.class);
  deployment.addAllDependencies();
  swarm.start().deploy(deployment);
}
```

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

  Swarm swarm = new Swarm();
  swarm.fraction(
    LoggingFraction.createDebugLoggingFraction()
  );

  JAXRSArchive deployment = ShrinkWrap.create(JAXRSArchive.class, "myapp.war");
  deployment.addClass(MyResource.class);
  deployment.addClass(NotFoundExceptionMapper.class);
  deployment.addAllDependencies();
  swarm.start().deploy(deployment);
}
```

Properties

or

YAML

project-stages.yaml

```
database:
    connection:
        url: "jdbc:h2:mem:test-db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE"
---
project:
    stage: production
database:
    connection:
        url: "jdbc:h2:mem:prod-db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE"
```

What about

"microservicey stuff"

?!

Logstash

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>logstash</artifactId>
</plugin>
```

```
-Dswarm.logstash.hostname=..
-Dswarm.logstash.port=..
```

Keycloak

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>keycloak</artifactId>
</plugin>
```

include a **keycloak.json** with your app

Discovery

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

Handles registration and discovery of services for **NetFlix Ribbon** support.

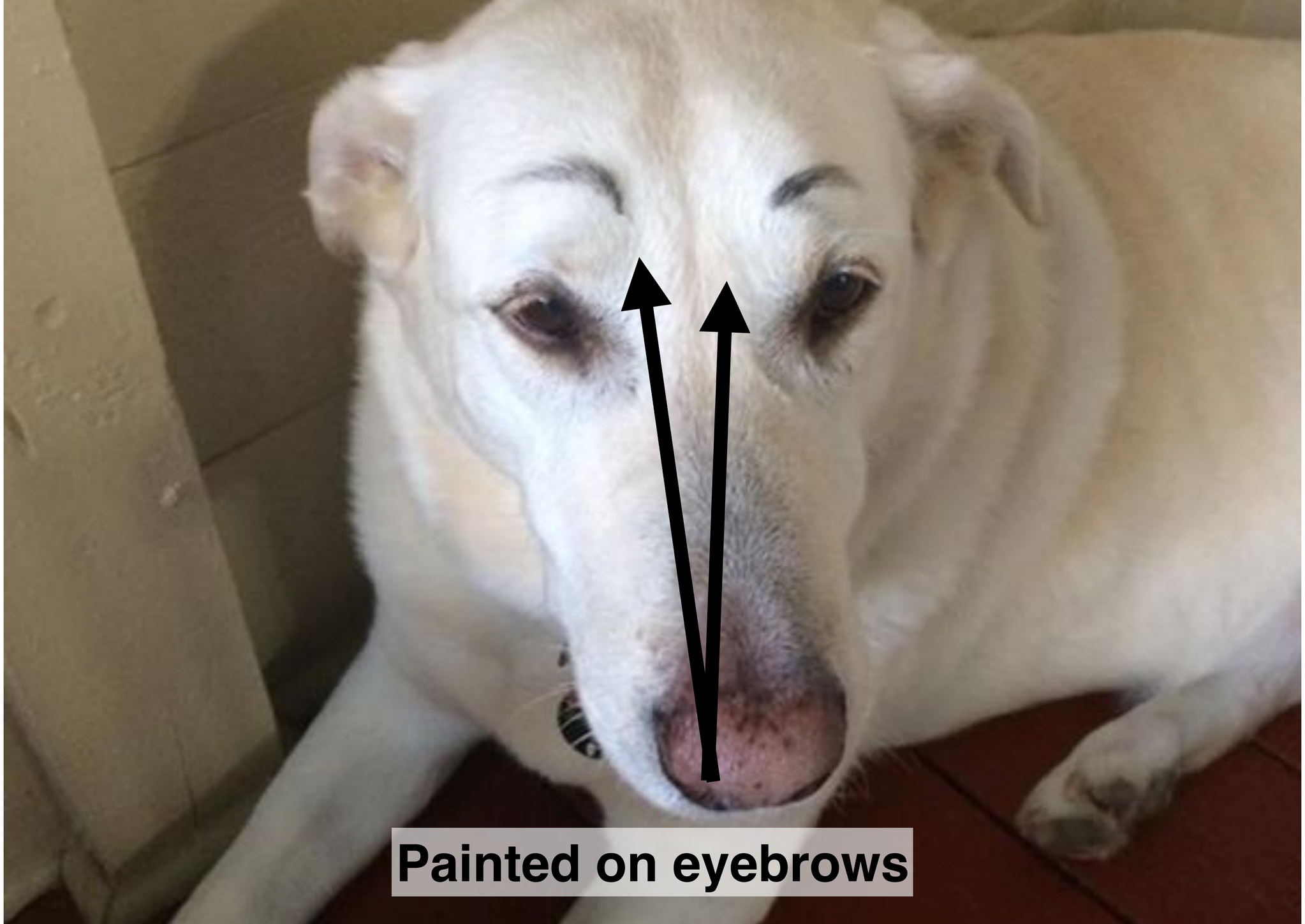
Bottom line…

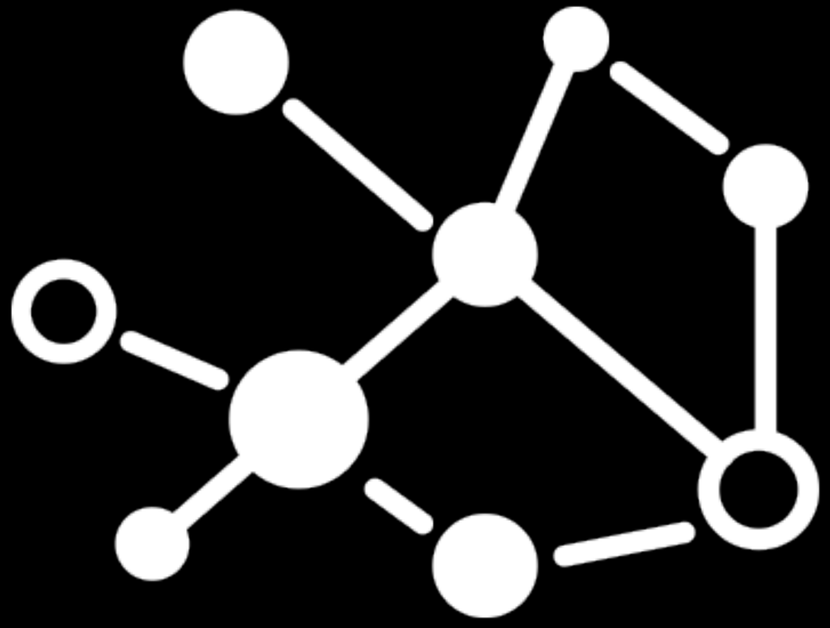
Java EE
is a
perfectly acceptable
way to write microservices.

Java EE
is an
awesomely fantastic
way to write microservices.

Surprised?


So is my dog. But he always looks surprised.


Painted on eyebrows



MicroProfile

WildFly SWARM

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>microprofile</artifactId>
</plugin>
```

Or just **auto-detect** a complaint application

JAX-RS

JSON-P

CDI

Resources

http://github.com/wildfly-swarm/wildfly-swarm/

http://github.com/wildfly-swarm/wildfly-swarm-examples/

http://wildfly-swarm.io/

#wildfly-swarm on irc.freenode.net

issues.jboss.org/browse/SWARM

https://groups.google.com/forum/#!forum/wildfly-swarm

Thanks!

If you want to chat, find the Australian or the tall guy.