JBoss WORLD
CHICAGO 2009

FOLLOW US:

TWITTER.COM/REDHATSUMMIT

TWEET ABOUT US:

ADD #SUMMIT AND/OR #JBOSSWORLD TO THE END OF YOUR EVENT-RELATED TWEET

presented by

JBoss WORLD CHICAGO 2009

Beyond Rails with TorqueBox

Bob McWhirter
Red Hat
4 September 2009

presented by

Agenda

- Who is this guy up here, talking to you?
- The Language Cusp
  - Ruby vs Java
  - Polyglotism
  - picture of Bill
- App servers for Java and Ruby
- Basic of Rails on TorqueBox
- Beyond Rails on TorqueBox
- How JBoss AS makes this possible and easy

Who is Bob?

- Active in open-source
- Doing Java for a dozen years
- Doing Ruby for a handful of years
- Research & Prototyping group at JBoss

codehaus

Drools
Business Logic
integration
Platform /

Groovy

TorqueBox

Java is facing competition from **other languages**

The Language Cusp

Groovy

Scala

Clojure

polyglot:

a **mixture** or **confusion** of languages

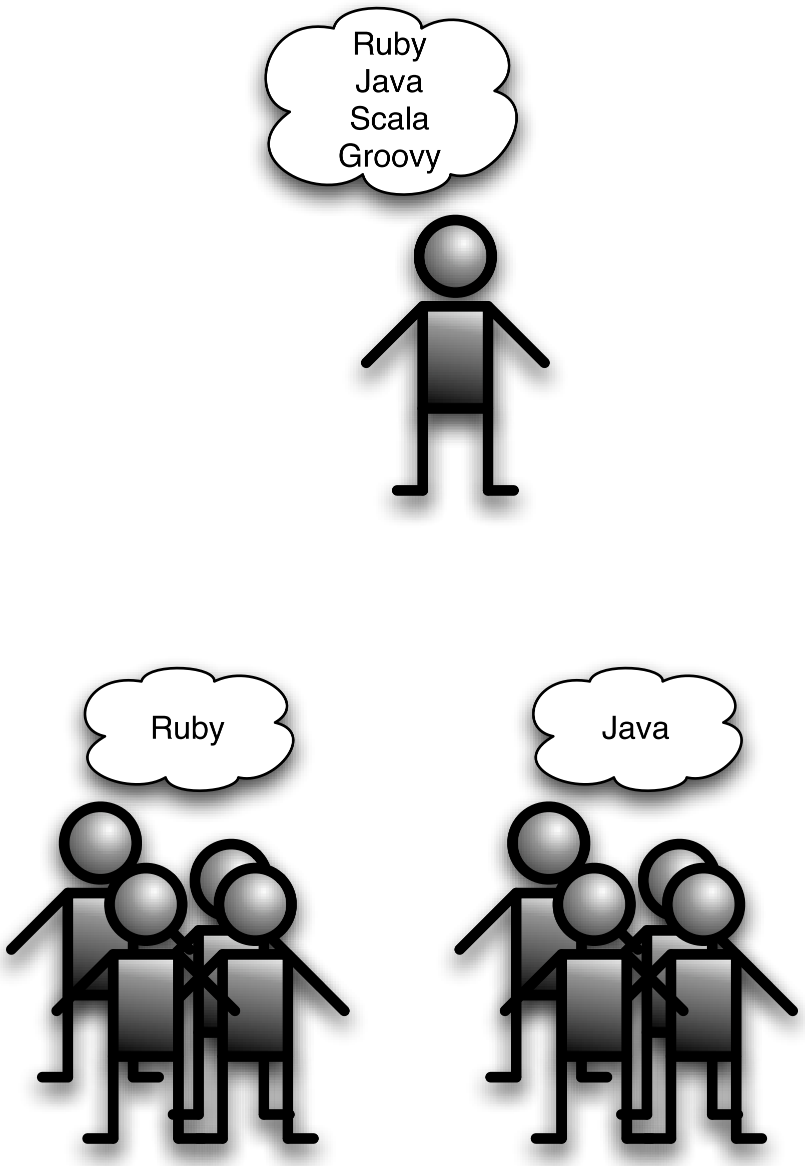
Polyglotism

“Polyglotism is the **worst idea** I ever heard”

-Bill Burke, coworker

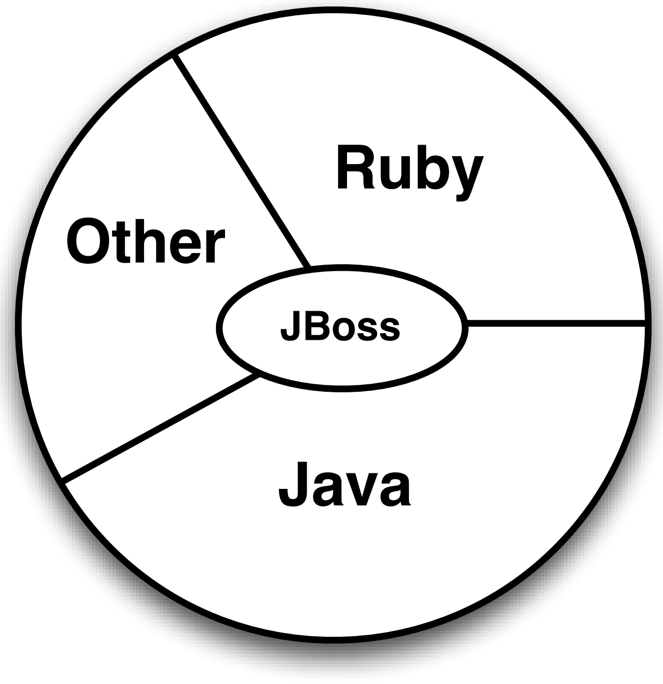
Polyglotism

Polyglotism may indeed be bad within a single developer's head or even within a single team.



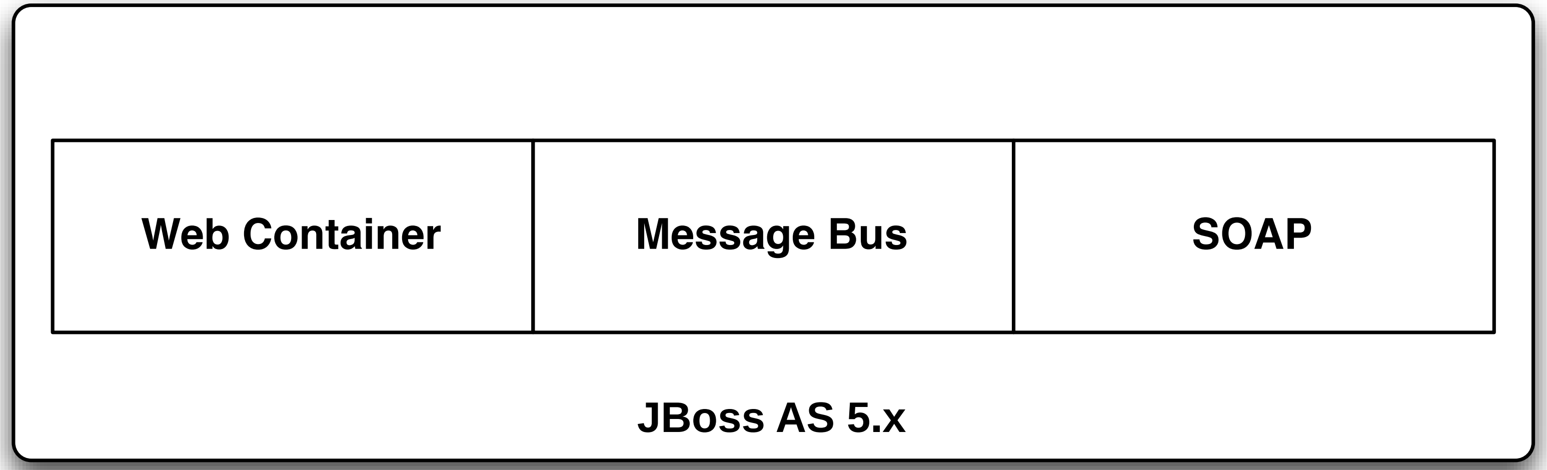
Polyglotism

Underlying infrastructure, if polyglotic, can support a **larger community** and **market**.



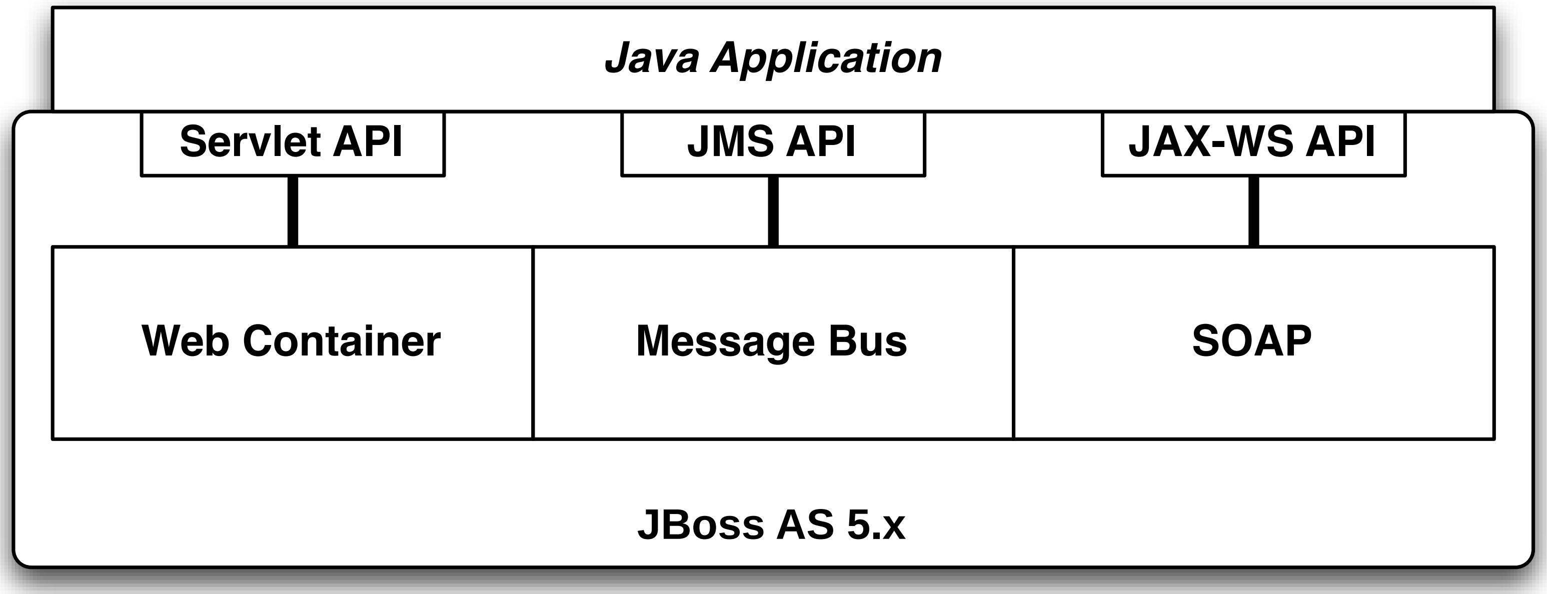
Services vs APIs

JBoss already has a full suite of enterprise-grade services.



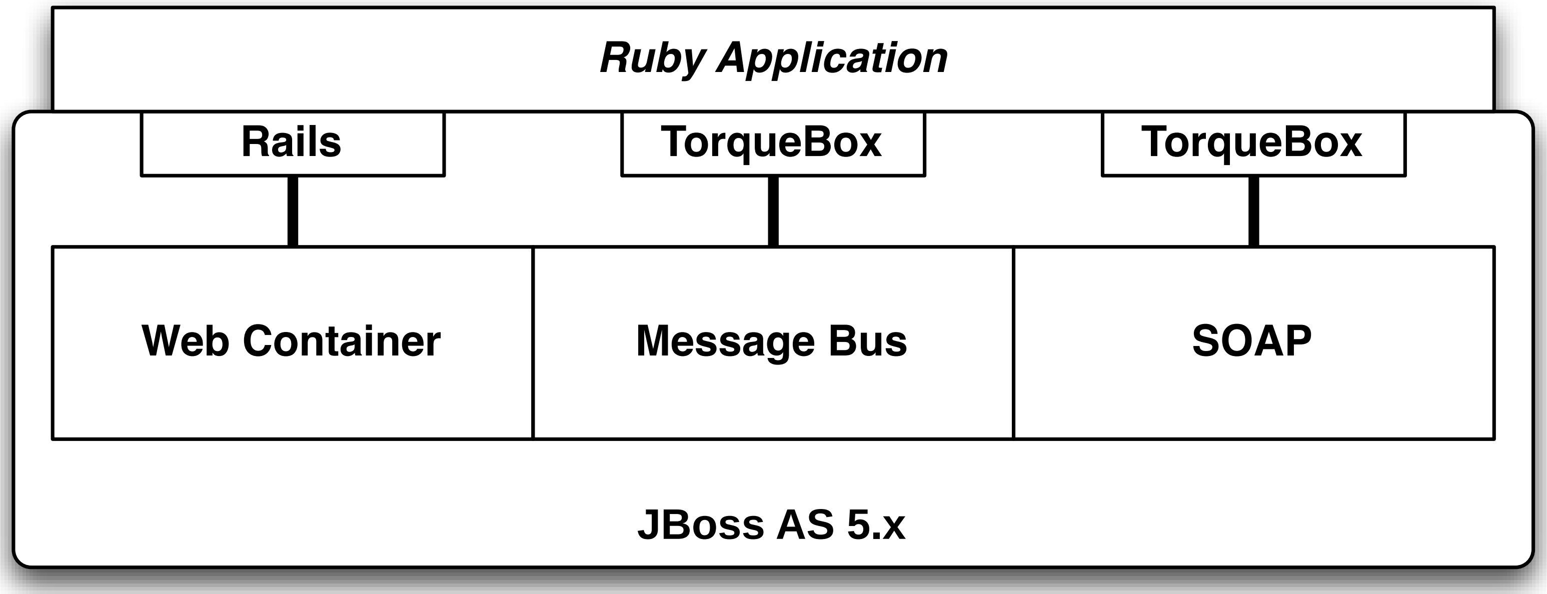
Services vs APIs

Wrapped with standard Java APIs...



Services vs APIs

Why not wrap with Ruby APIs?



Ruby App Server

Then, you end up with an enterprise-grade **Ruby app server**.

TorqueBox

Ruby App Server in 4 Steps

Step 1

Ruby on Rails

Step 1: Rails

- **JRuby**
  - The guys got regular **Rails** running well under mongrel using JRuby
  - There is also **Warbler** for creating deployable WAR files
  - **Gl*ssfish** can run Rails apps in-place

Step 1: Rails

But that's not good enough

Step 1: Rails on JBoss

- **JBoss**
  - Run Rails apps in-place under JBoss
  - No WAR-creation required
  - Runs alongside other JEE apps
  - Runs alongside other Servlets within the same application

Step 1.5

Databases

Step 1.5: Databases

- Since Java has the very nice **JDBC** drivers, let’s use them
- But don’t want to teach Rubyists **JDBC**
- Add a few **ActiveRecord** driver gems, and your Rails application accesses the DB through JDBC

Step 1.5: Databases

- **No changes** to **`config/database.yml`** required
- Rails is managing the connections itself

Step 1.75: Managed connections on Rails

- If’n you want to use a managed datasource deployed outside of the application...
  - You can make changes to **`config/database.yml`** to use a datasource
  - Datasource located via JNDI

Step 1.75: Managed connections on Rails

- You can even deploy your datasource from within your Rails application:
  - `config/mydb-ds.xml`

Step 1.97: Deployment

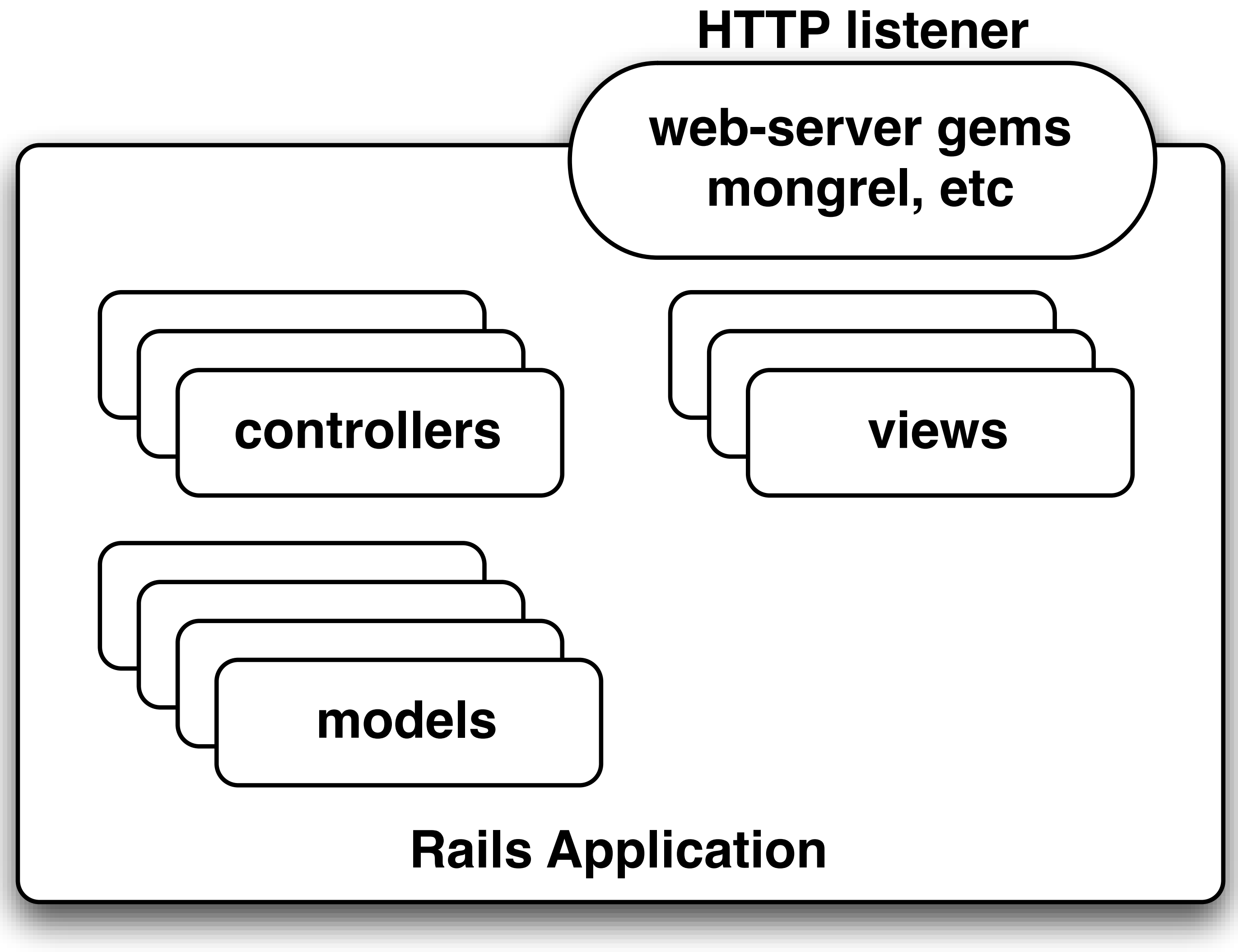
Deployment with **TorqueBox** is slightly different, but familiar to JBoss users.

Inversion of Deployment

Traditional Rails

- You pull HTTP functionality into your app
- You run your app, which listens on a port

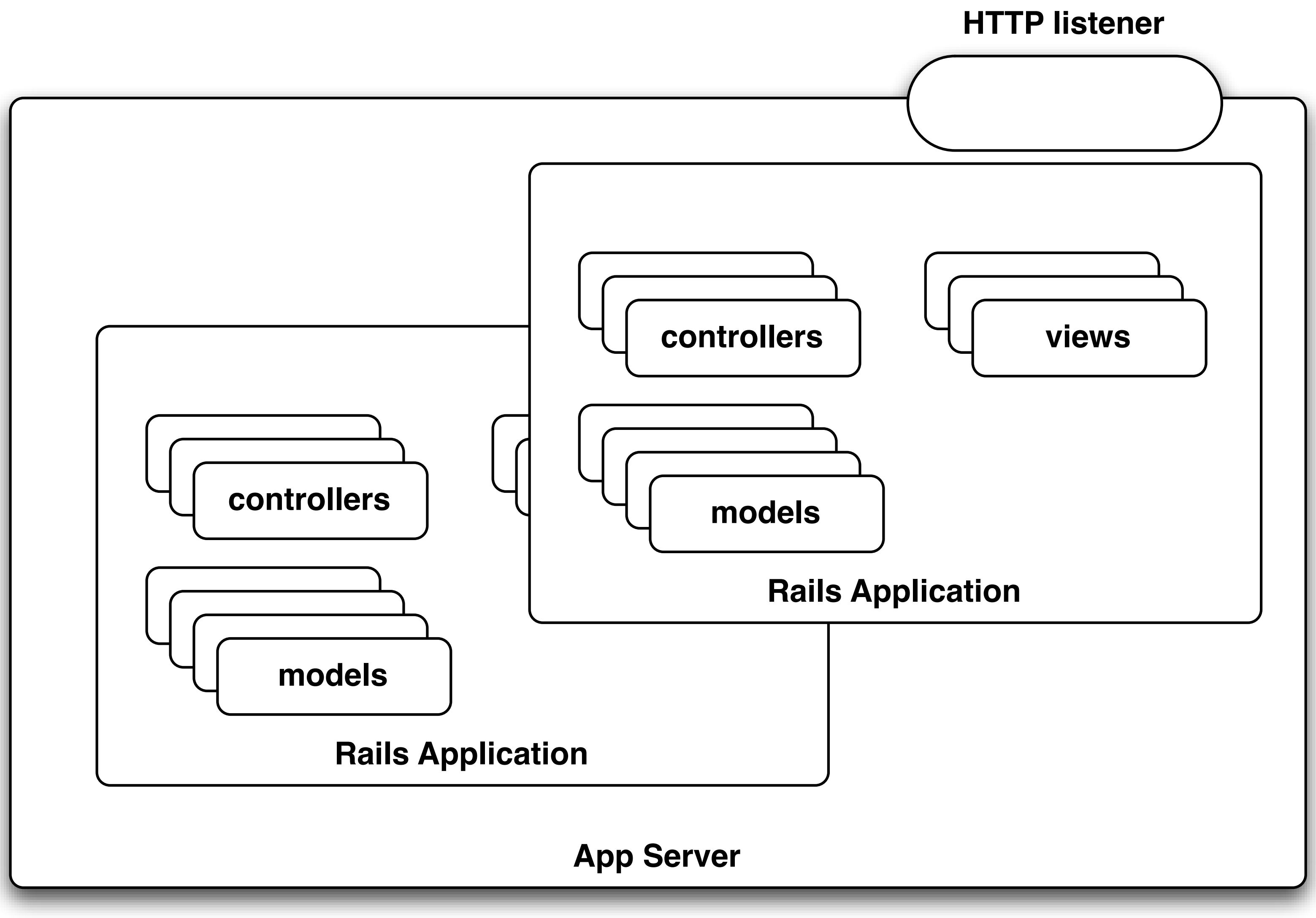
Inversion of Deployment: Traditional



Rails in an app server

- Load your app into an app-server which already listens to HTTP
- App server routes some requests to your app or other apps

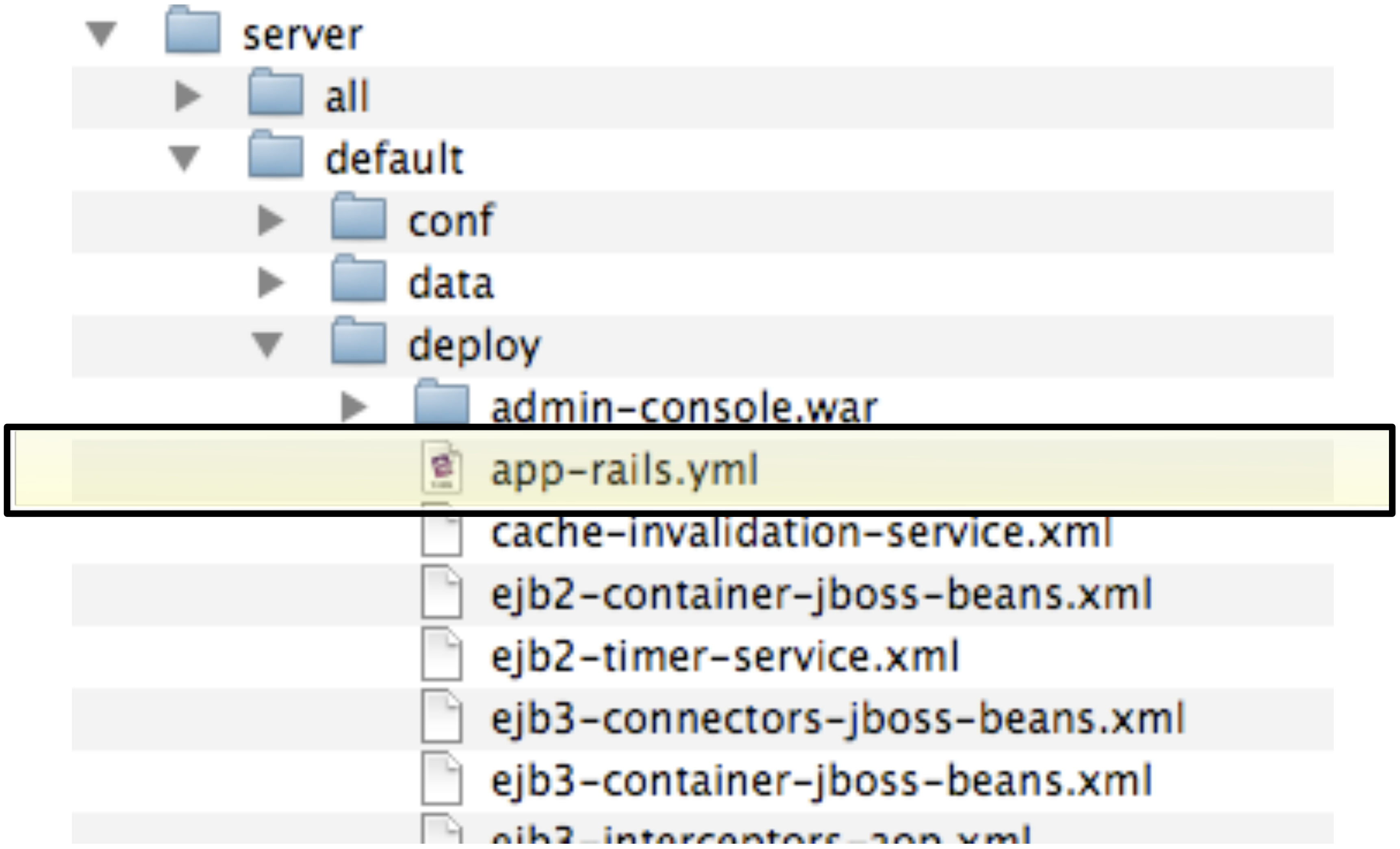
Inversion of Deployment: App server



Deployment

- You don’t “start the app”
- You “deploy” it into an App Server
- TorqueBox comes with **Rake** tasks to help
  - **rake torquebox:rails:deploy**
  - **rake torquebox:run**

Deployment Descriptor



```
▼ server
  ▶ all
  ▼ default
    ▶ conf
    ▶ data
    ▼ deploy
      ▶ admin-console.war
        app-rails.yml
        cache-invalidation-service.xml
        ejb2-container-jboss-beans.xml
        ejb2-timer-service.xml
        ejb3-connectors-jboss-beans.xml
        ejb3-container-jboss-beans.xml
        ejb3-interceptors-aop.xml
```

Simple Deployment

```
application:
  RAILS_ENV: development
  RAILS_ROOT: /path/to/my/app
web:
  context: /
```

Simple Deployment

```
application:
  RAILS_ENV: development
  RAILS_ROOT: /path/to/my/app
web:
  context: /
```

Simple Deployment

```
application:
  RAILS_ENV: development
  RAILS_ROOT: /path/to/my/app
web:
  context: /
```

Simple Deployment

```
application:
  RAILS_ENV: development
  RAILS_ROOT: /path/to/my/app
web:
  context: /
  host: www.myhost.com
```

Act like normal

- Once deployed, continue to edit
    - Models
    - Views
    - Controllers
- **Without re-deploying** your app

Go **beyond** Rails

Step 2: Scheduled Jobs

- Sometimes you’ve got a recurring task not associated with a web request
- A **cron job**

Step 2: Scheduled Jobs

- Let’s use Quartz, it comes with JBoss

config/jobs.yml

```
github.commit_poller:
  description: Poll GitHub
  job: Github::CommitPoller
  cron: 12 */10 * * * ?
```

Step 2: Scheduled Jobs

- We're used to
  - `app/controllers/**.rb`
  - `app/views/**.erb`
  - `app/models/**.rb`
- So let's go with
  - **`app/jobs/**.rb`**

Step 2: Scheduled Jobs

```
module GitHub
  class CommitPoller
    include TorqueBox::Jobs::Base
    def run()
      # do work here
    end
  end
end
```

Step 2: Scheduled Jobs

```
module GitHub
  class CommitPoller
    include TorqueBox::Jobs::Base
    def run()
      # do work here
    end
  end
end
```

Step 2: Scheduled Jobs

```
module GitHub
  class CommitPoller
    include TorqueBox::Jobs::Base
    def run()
      # do work here
    end
  end
end
```

Step 2: Scheduled Jobs

```
module GitHub
  class CommitPoller
    include TorqueBox::Jobs::Base
    def run()
      # do work here
    end
  end
end
```

Step 2: Scheduled Jobs

- Jobs will **deploy** with your app
- Jobs will **undeploy** with your app
- Jobs have complete access to your **ActiveRecord** models
- Jobs have complete access to your **lib/** classes
- Jobs can be **live edited** like anything else

Well, that was **easy**

Step 3: Async Task Queues

- Sometimes you want something non-recurring to happen
- Perhaps outside of the context of a web request
- Perhaps triggered by a web request, though

That sounds like a **message queue**.

JBoss has one of those.

Step 3: Async Task Queues

- Like you’d expect...
  - `app/queues/**.rb`
- A class per queue
- A method per task

Step 3: Async Task Queues

```
class MyQueue
  include TorqueBox::Queue::Base

  def handle_something(payload={})
    # do work here
  end
end
```

Step 3: Async Task Queues

```
class MyQueue
  include TorqueBox::Queue::Base

  def handle_something(payload={})
    # do work here
  end
end
```

Step 3: Async Task Queues

```
class MyQueue
  include TorqueBox::Queue::Base

  def handle_something(payload={})
    # do work here
  end
end
```

Step 3: Async Task Queues

```
class MyQueue
  include TorqueBox::Queue::Base

  def handle_something(payload={})
    # do work here
  end
end
```

Step 3: Enqueuing

```
MyQueue.enqueue(:do_something,{
    :quantity=>100,
    :cheese=>:gouda
})
```

Step 3: Enqueuing

```
MyQueue.enqueue(:do_something,{
    :quantity=>100,
    :cheese=>:gouda
})
```

Step 3: Enqueuing

```
MyQueue.enqueue(:do_something,{
    :quantity=>100,
    :cheese=>:gouda
})
```

Step 3: Async Task Queues

- A **JMS queue** is created for each queue class
- The payload is anything that can be serialized into bytes
  - Including **ActiveRecord** models

Sometimes you’ve got to use **SOAP**

Step 4: SOAP

- Sure, SOAP is **obnoxious**
- SOAP from Ruby is obnoxious, and **underpowered**
- **Apache CXF** is some good stuff
- Sometimes you have to do SOAP, so **at least** you can do it from Ruby

Step 4: SOAP

- Goal is **not to generate WSDL** from Ruby endpoints
- Instead, only supports binding Ruby endpoints to **existing** WSDL
- If you're doing greenfield development, prefer **REST**. Or **sockets**. Or ***pigeons***.

Step 4: SOAP

- As you’d expect, again...
  - **`app/endpoints/**.rb`**
  - **`app/endpoints/**.wsdl`**

Step 4: SOAP

```
module Amazon
  class Ec2Endpoint

    include TorqueBox::Endpoints::Base

  end
end
```

Step 4: SOAP

```
module Amazon
  class Ec2Endpoint
    include TorqueBox::Endpoints::Base
      endpoint_configuration do
        target_namespace 'http://ec2.amazonaws.com/doc/2008-12-01/'
        port_name        'AmazonEC2'
        security do
          inbound do
            verify_timestamp
            verify_signature
          end
        end
      end
  end
end
```

Step 4: SOAP

```
module Amazon
  class Ec2Endpoint
    include TorqueBox::Endpoints::Base
      endpoint_configuration do
        target_namespace 'http://ec2.amazonaws.com/doc/2008-12-01/'
        port_name        'AmazonEC2'
        security do
          inbound do
            verify_timestamp
            verify_signature
          end
        end
      end
  end
end
```

Step 4: SOAP

```
module Amazon
  class Ec2Endpoint
    include TorqueBox::Endpoints::Base
      endpoint_configuration do
        target_namespace 'http://ec2.amazonaws.com/doc/2008-12-01/'
        port_name        'AmazonEC2'
        security do
          inbound do
            verify_timestamp
            verify_signature
          end
        end
      end
  end
end
```

Step 4: SOAP

```
module Amazon

  class Ec2Endpoint

    def describe_instances

      response = create_response

      request.instancesSet.each do |instance_id|
        reservation_info = response.reservationSet.create
        reservation_info.ownerId = ...
      end

      return response

    end
  end

end
```

Step 4: SOAP

- **TorqueBox** provides...
  - full request/response **XSD data-binding** (like **JAXB**)
  - security, such as **X.509** signature verification

Now you have a pretty nice **Ruby app server**.

Not too shabby.

And **JBoss** makes it possible thanks to the new design of the **JBoss Microcontainer**.

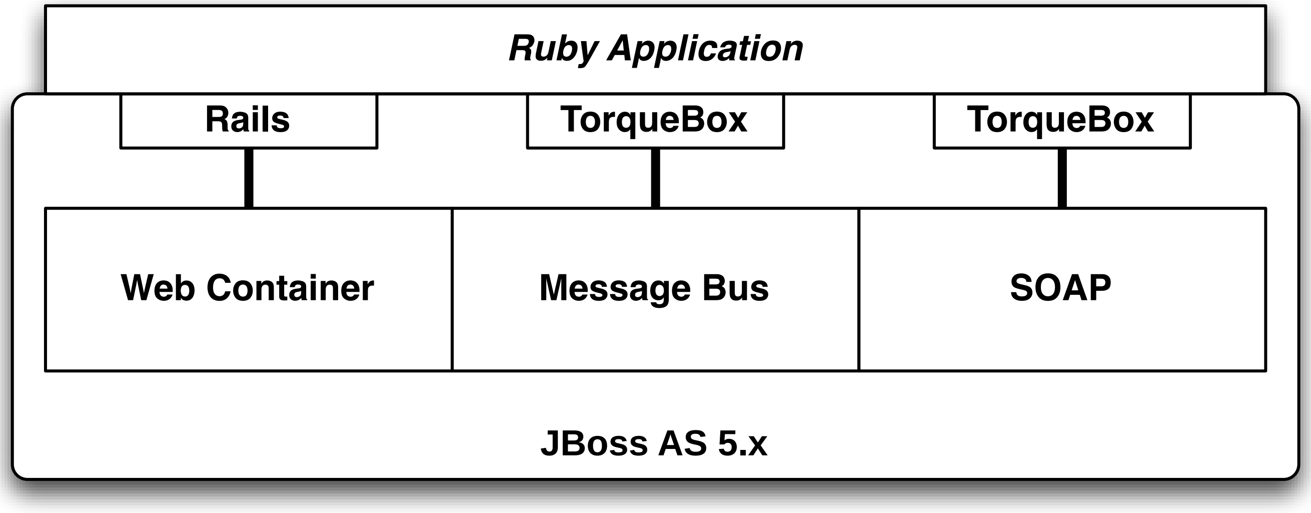
JBoss Microcontainer

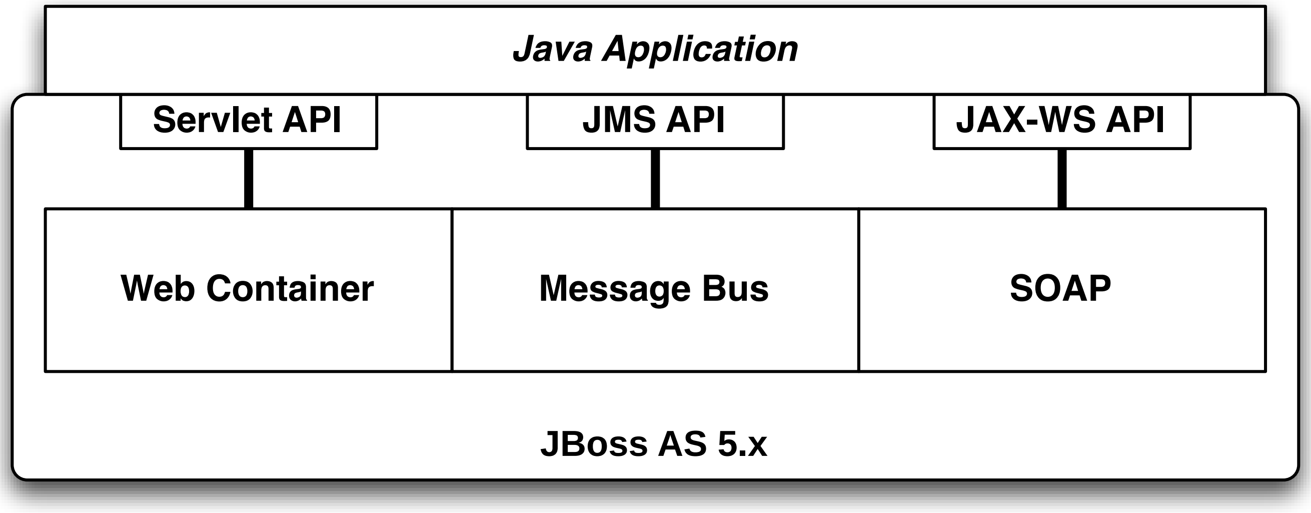
- Microcontainer is a typical **IoC** container
- Microcontainer includes a **deployers framework**, which gives you many options for standing up your **POJO**s
- You can use the **`jboss-beans.xml`** format or create something new

JBoss AS is just a usage of Microcontainer

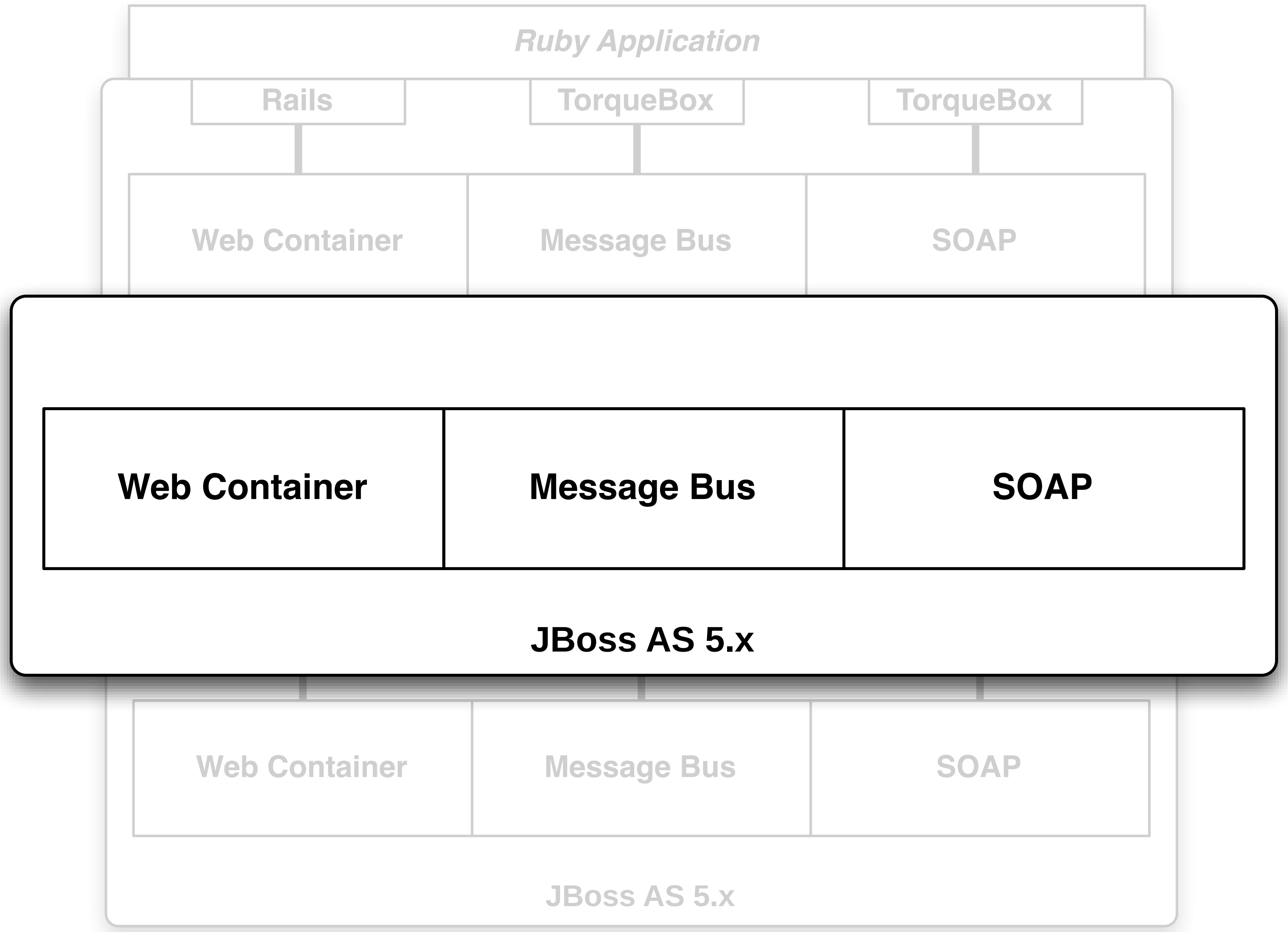
- Everything in JBoss AS is ultimately a **POJO**
- The POJOs are configured normally via Java-EE specific deployment descriptors
    - <XML>
    - @Annotations

Remember these diagrams?





Remember these diagrams?



How does it work?

- We need to deploy instances of services, which **just happen to be** based upon Ruby
    - A web application
    - A message queue
    - A SOAP servlet
    - A scheduled job

It also needs to deploy these services in a way that **just happens not to be** based upon Java-EE specifications

It’s all about the **deployers**

- **Deployers** are the key to working with Microcontainer and JBoss AS
- Services are deployed as **POJOs** and configured using **metadata**
- Deployers are the **links** between **files**, **metadata**, and **Microcontainer**

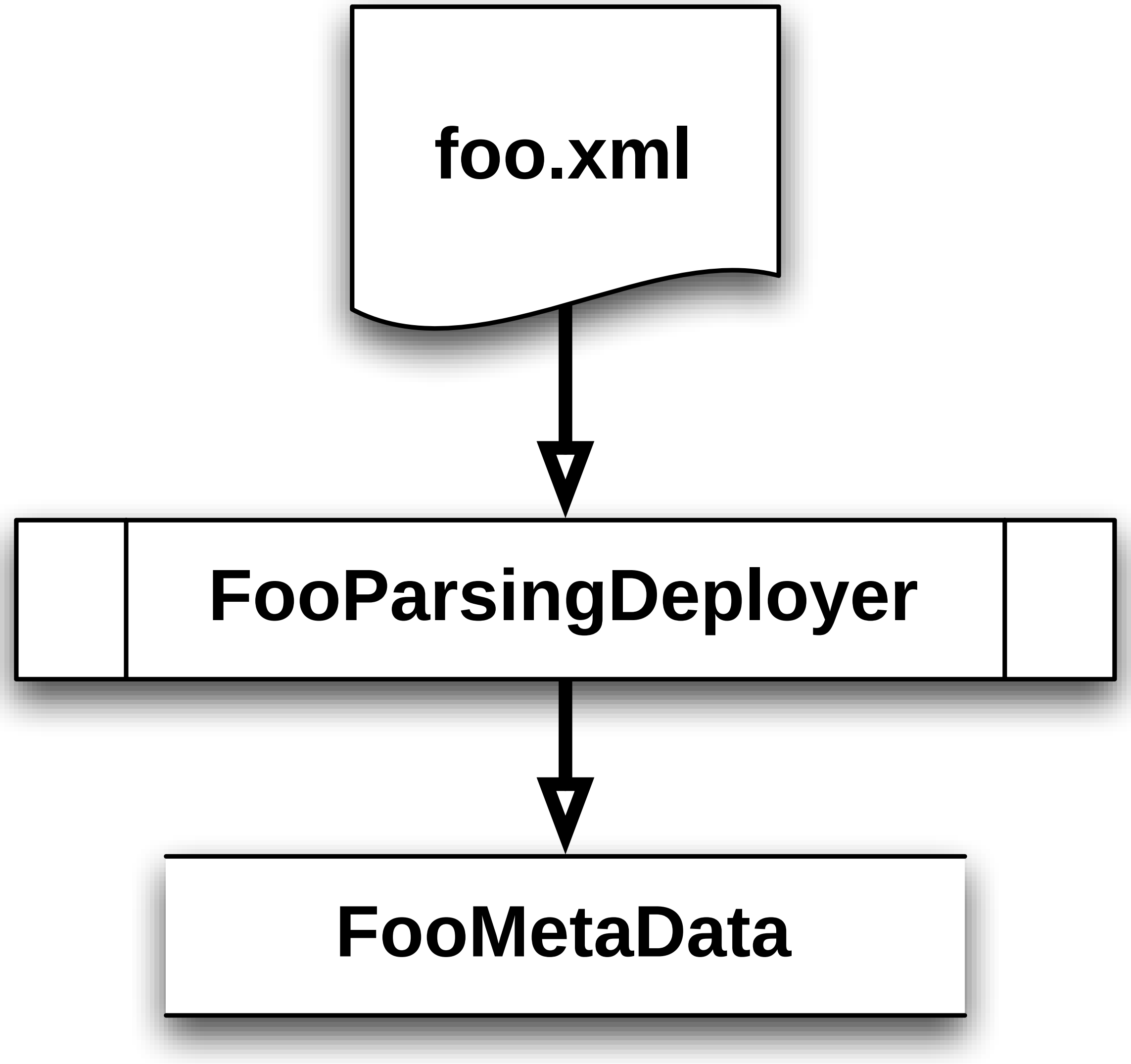
It’s also all about metadata

- **Metadata** is just the configuration and description of the service
- Metadata can come from files like **web.xml**, or annotations such as **@WebService**
- Metadata can also be constructed programatically by deployers

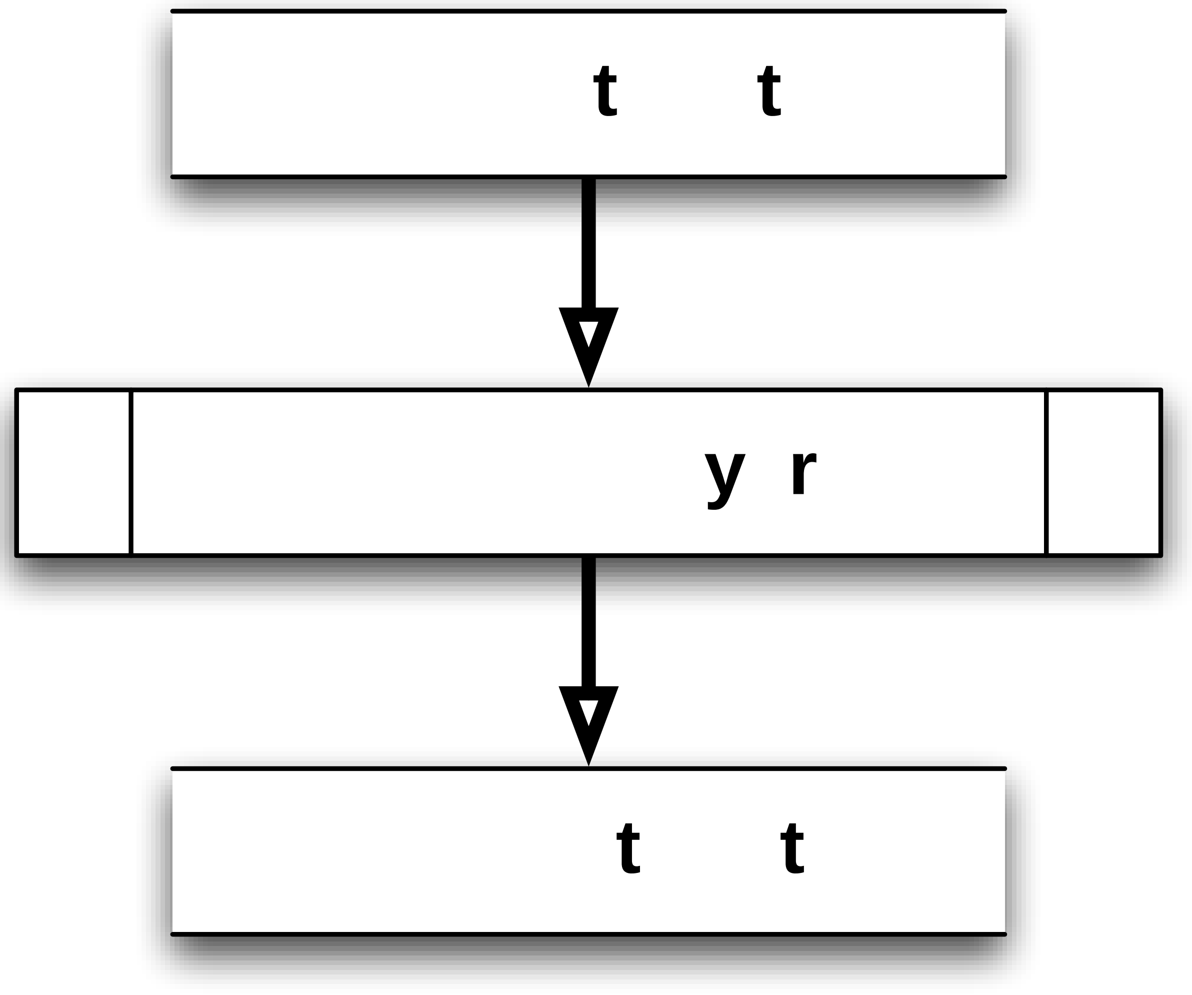
POJOs & Describing POJOs

- Everything running in **Microcontainer** is a POJO
- Deployers never instantiate the POJOs directly
- Deployers describe the POJOs, along with dependencies
- Microcontainer handles **injection** and **lifecycle**

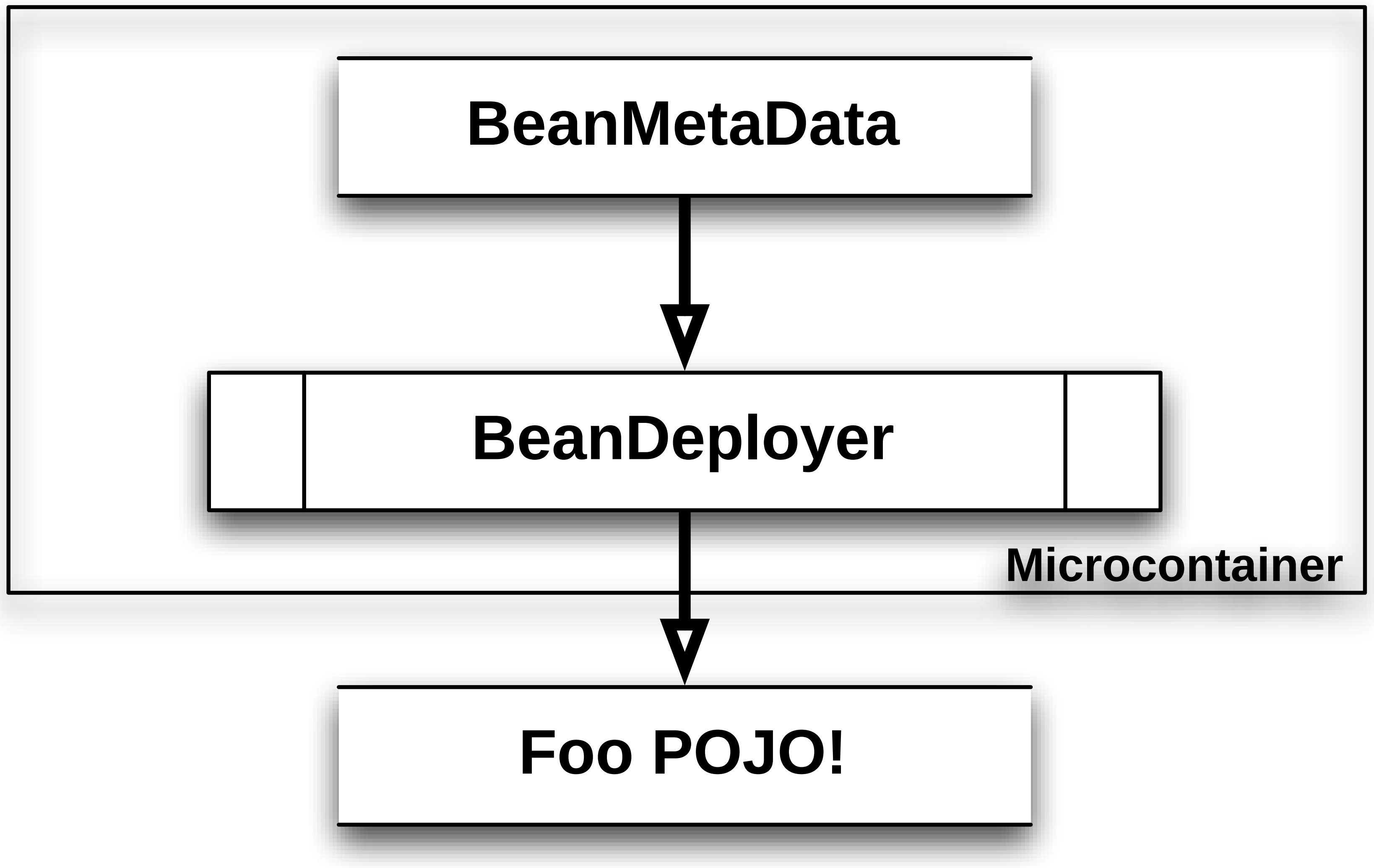
File to Metadata



Specific Metadata to POJO Metadata



POJO Metadata to a bonafide POJO



Structure Deployers

Know more about a deployable asset by its **format** and **shape**

Structure Deployers

What's this?

- `/`
  - `META-INF/`
  - `org/`
    - `torquebox/`
      - `Server.class`

Structure Deployers

What's this?

- `/`
  - `META-INF/`
  - `org/`
    - `torquebox/`
      - `Server.class`

Looks like a JAR

Structure Deployers

What's this?

- `/`
  - `WEB-INF/`
    - `classes/`
    - `lib/`
      - `jboss-foo.jar`

Structure Deployers

What's this?

- `/`
  - `WEB-INF/`
    - `classes/`
    - `lib/`
      - `jboss-foo.jar`

Looks like a WAR

Structure Deployers

What's this?

- `/`
  - `config/`
  - `app/`
    - `models/`
    - `controllers/`
    - `views/`

Structure Deployers

What's this?

- `/`
  - `config/`
  - `app/`
    - `models/`
    - `controllers/`
    - `views/`

Looks like a Rails app

Finding Metadata

- In a JAR
  - **`/META-INF`**
- In a WAR
  - **`/META-INF`**
  - **`/WEB-INF`**
- In a Rails app
  - **`/config`**

Places you can find deployment descriptors

Finding Classes

- In a JAR
  - `/*`
- In a WAR
  - `/WEB-INF/classes/*`
  - `/WEB-INF/lib/*.jar`
- In a Rails app
  - `/lib/java/*.jar`

Places you can find classes & resources

Structure Deployers

- Structure deployers are responsible for **recognizing** the **shape** of a “thing” being deployed
- And know what parts of it contain **metadata**, and what parts contain items to add to the **classpath**

Let’s **deploy**!

Deploying the Web App

- When a Rails application is noticed:
  - We set up a **Ruby runtime pool**
  - We set up a **Java Servlet Filter** to route requests through the Rails code

What’s that mean?

- We describe **the same** POJO that normal **web.xml** deployment ends up describing
- **Microcontainer** then instantiates it, and calls **start()**. *Just like a **web.xml-based web-app**.*

What’s that mean?

- We also describe (but not instantiate) our Ruby runtime pool **POJO**
- **Microcontainer** will instantiate it and **start()** it. We pull it into our servlet Filter.

Deploying scheduled jobs

- **Microcontainer** knows the **`config/`** directory may hold important metadata (from the structure deployer)
- Such as **`jobs.yml`**
- Deployers reads **`jobs.yml`**, and describes a scheduled-job POJO

Deploying scheduled jobs

- **Microcontainer** instantiates the scheduled job POJO we described
- It **injects** the quartz scheduler
- It **injects** the Ruby runtime pool
- And calls **start()**

Deploying task queues

- We see **`app/queues/**.rb`** and describe the same **POJO** that normal JMS destination deployment describes
- **Microcontainer** instantiates...
  - **injects** our Ruby runtime pool
  - and calls **start()**

Deploying SOAP endpoints

- We add more configuration to the web meta-data to wire up the **CXF Servlet**
- It’s set up **alongside** the Rails Servlet Filter
- Microcontainer manages the **injections** and **lifecycle**

Ruby App Server

- Ultimately, **TorqueBox** configures the same services that a Java-EE application configures
- But instead of **@Annotations** and **<XML>**, it's triggered by other sources
  - `**.rb`
  - `**.yml`

JBoss AS is not *just* a Java App Server

- **JBoss AS** is a collection of generic services
- By default we ship a Java personality wrapped around them
- **TorqueBox** wraps a Ruby personality around them

JBoss could be a Scala/Python/Clojure App Server

- Any language **that can run on the JVM** could be integrated with JBoss AS
- The same enterprise-grade services Java developers enjoy can be made available to other markets

Hey, thanks!

Thanks for sitting there, listening, and **ignoring Twitter** for the past hour.

You rock.

Any questions?

?

QUESTIONS?

TELL US WHAT YOU THINK:
REDHAT.COM/JBOSSWORLD-SURVEY