

TorqueBox

Ruby on Rails (and Rack)
with Enterprise Goodness

Bob McWhirter

Red Hat

John Williams

Agenda: Part 1

- Who are these guys 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
- Rack on TorqueBox
- Sinatra on TorqueBox

Agenda: Part 2

- **Why TorqueBox**
- **Deploying with Bundles**
- **Deploying using Capistrano**

Who is Bob McWhirter?

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



Who is John Williams?

- Early user of TorqueBox
- Interested in anything open-source
- Doing RoR for 4 years
- Doing web development for 8 years
- Wrote theme music to Star Wars

TorqueBox



Who are you?

Java is facing
competition from
other languages

The Language Cusp



Clojure

polyglot:

a **mixture** or
confusion of
languages

Polyglotism

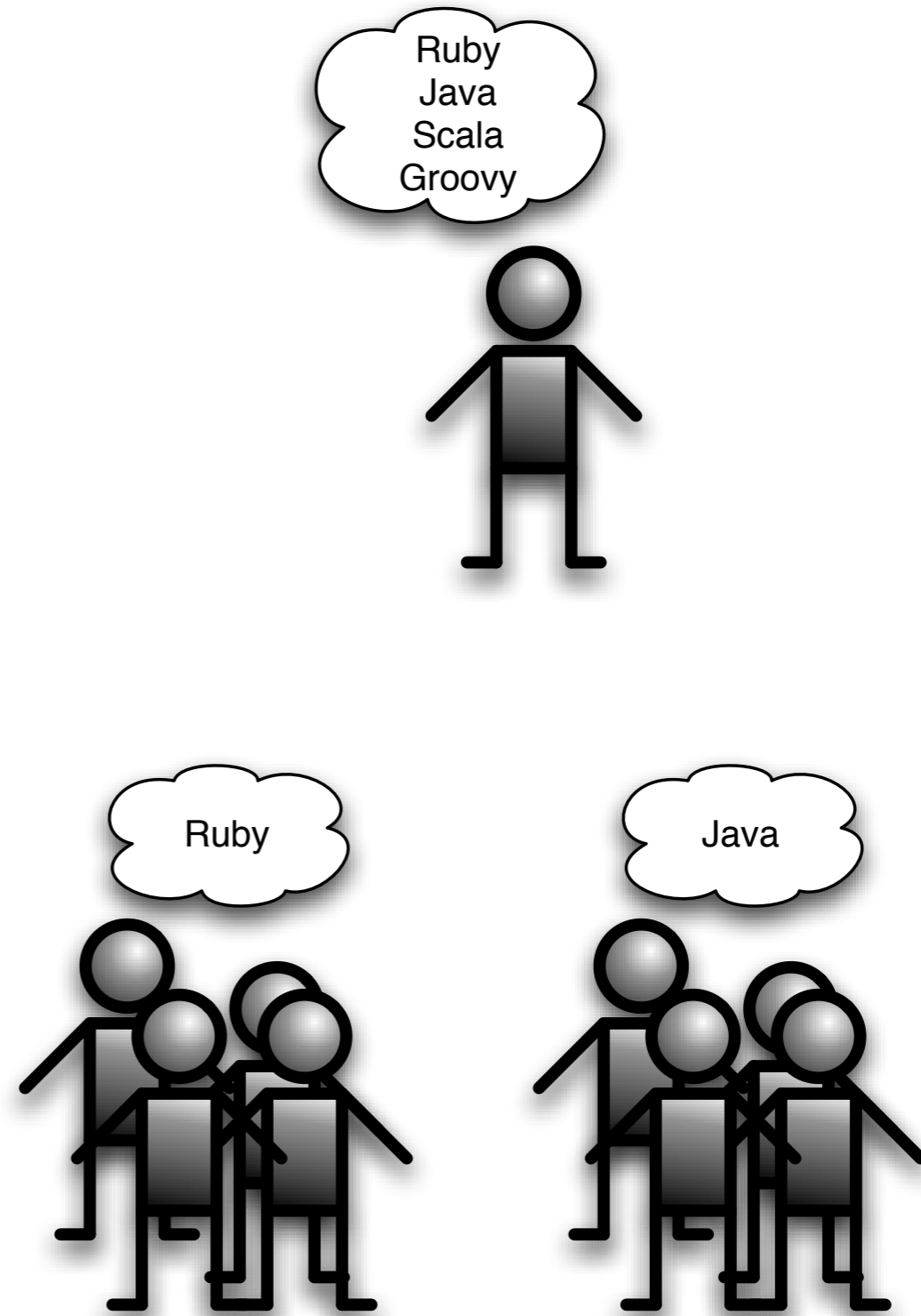
“Polyglotism
is the **worst
idea** I ever
heard”

-Bill Burke, coworker



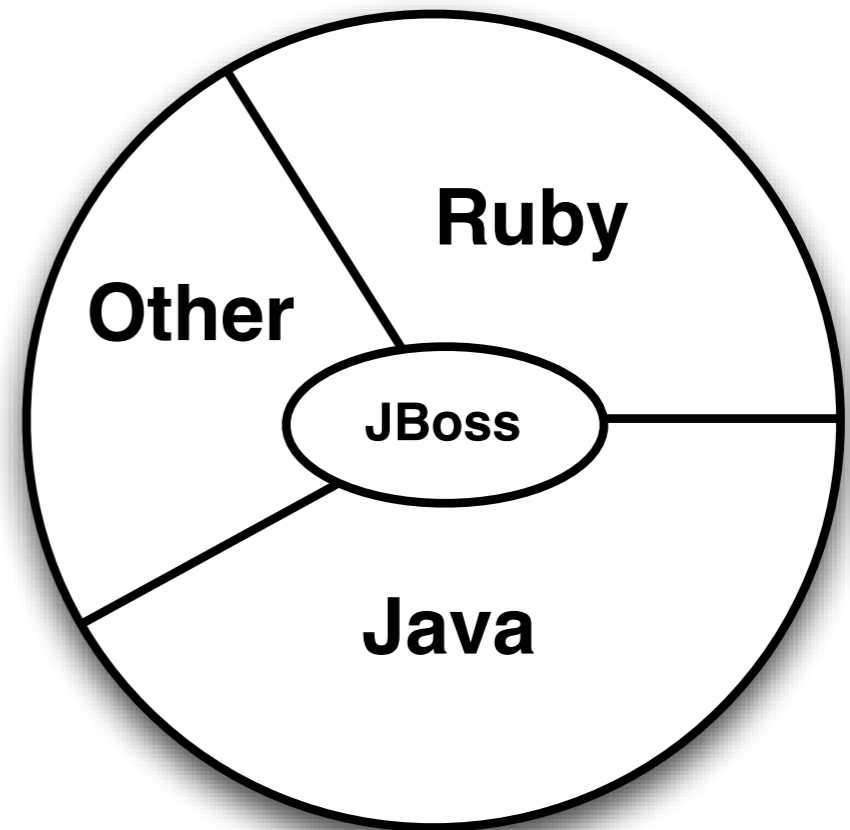
Polyglotism

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



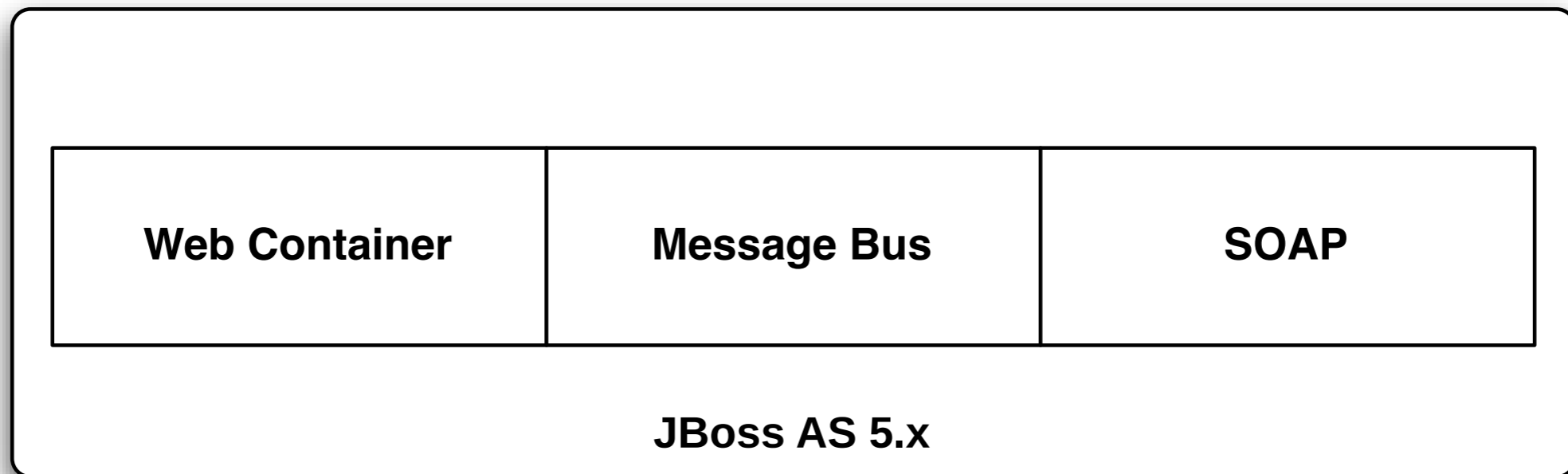
Polyglotism

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



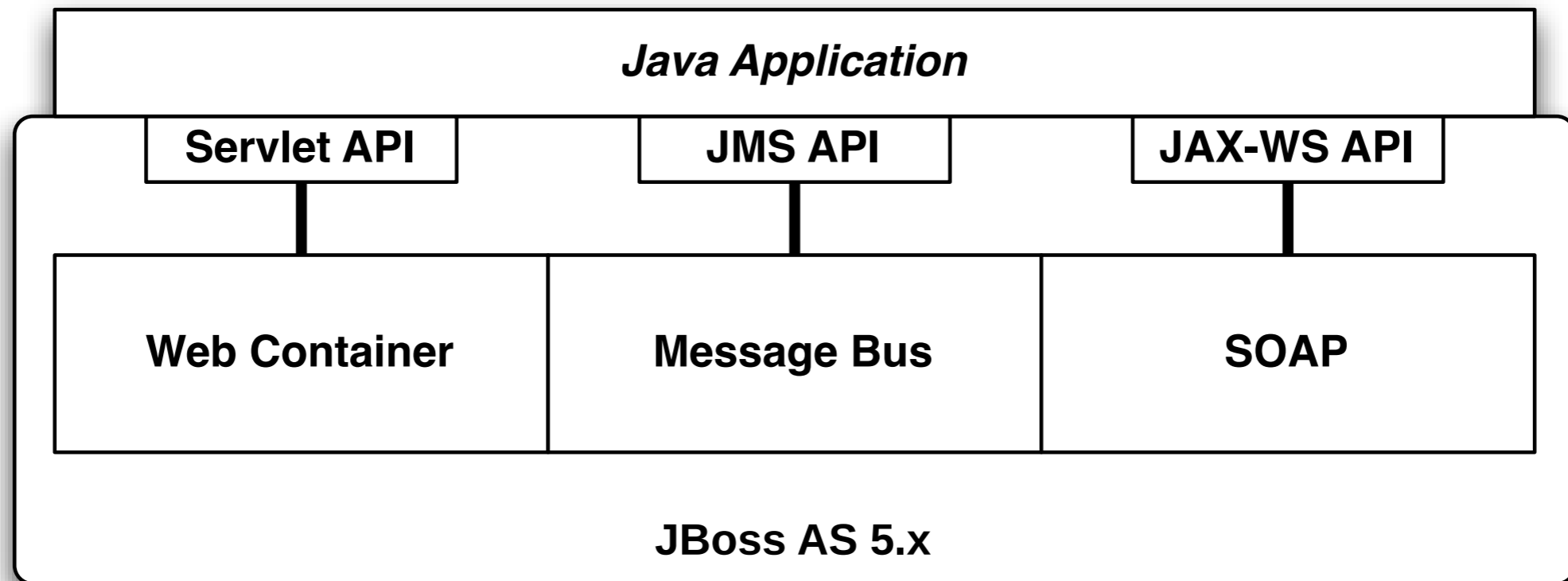
Services vs APIs

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



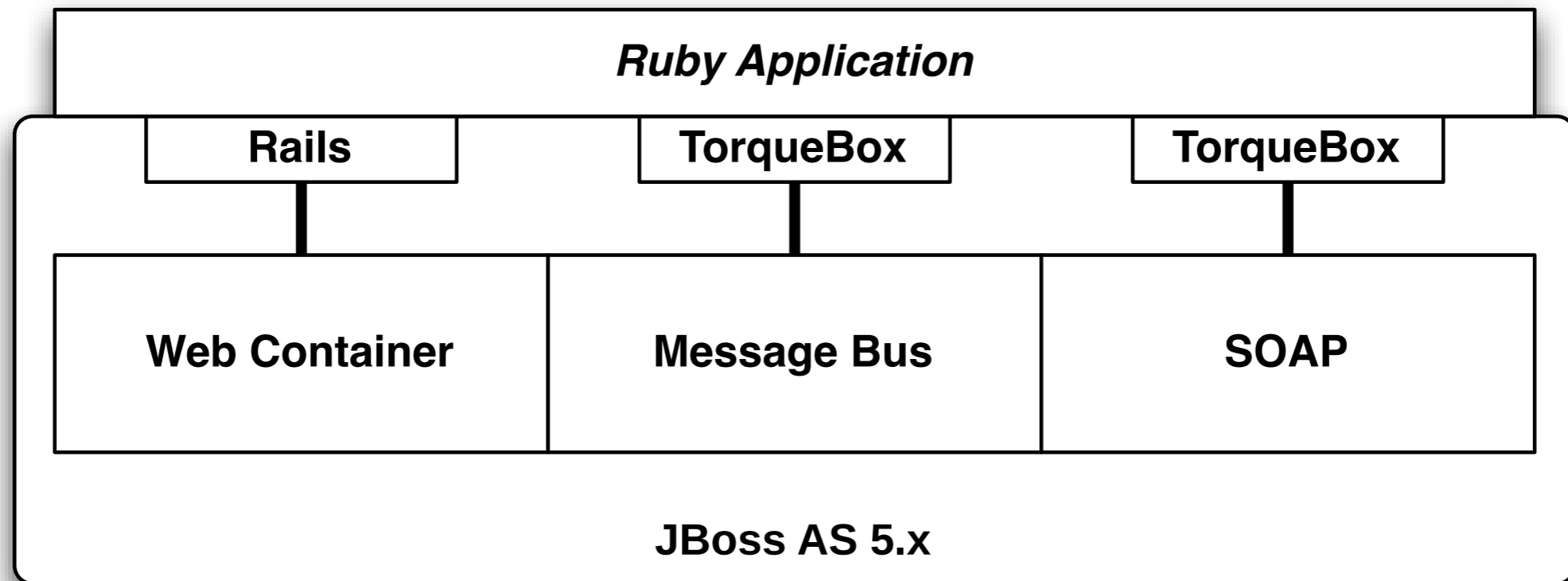
Services vs APIs

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
 - **GI*ssf** 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

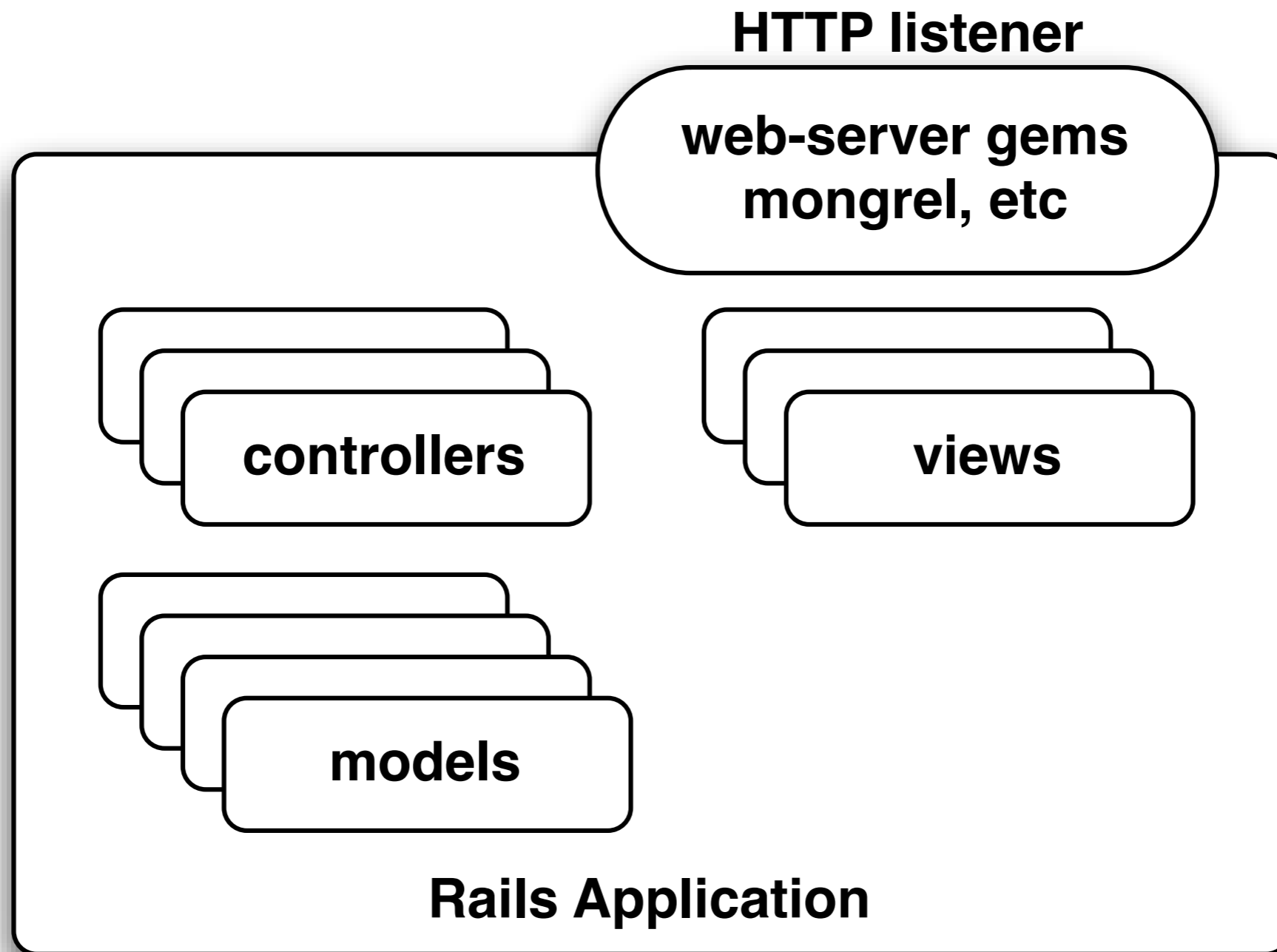
Deployment with **TorqueBox** is slightly different, but familiar to JBoss users.

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

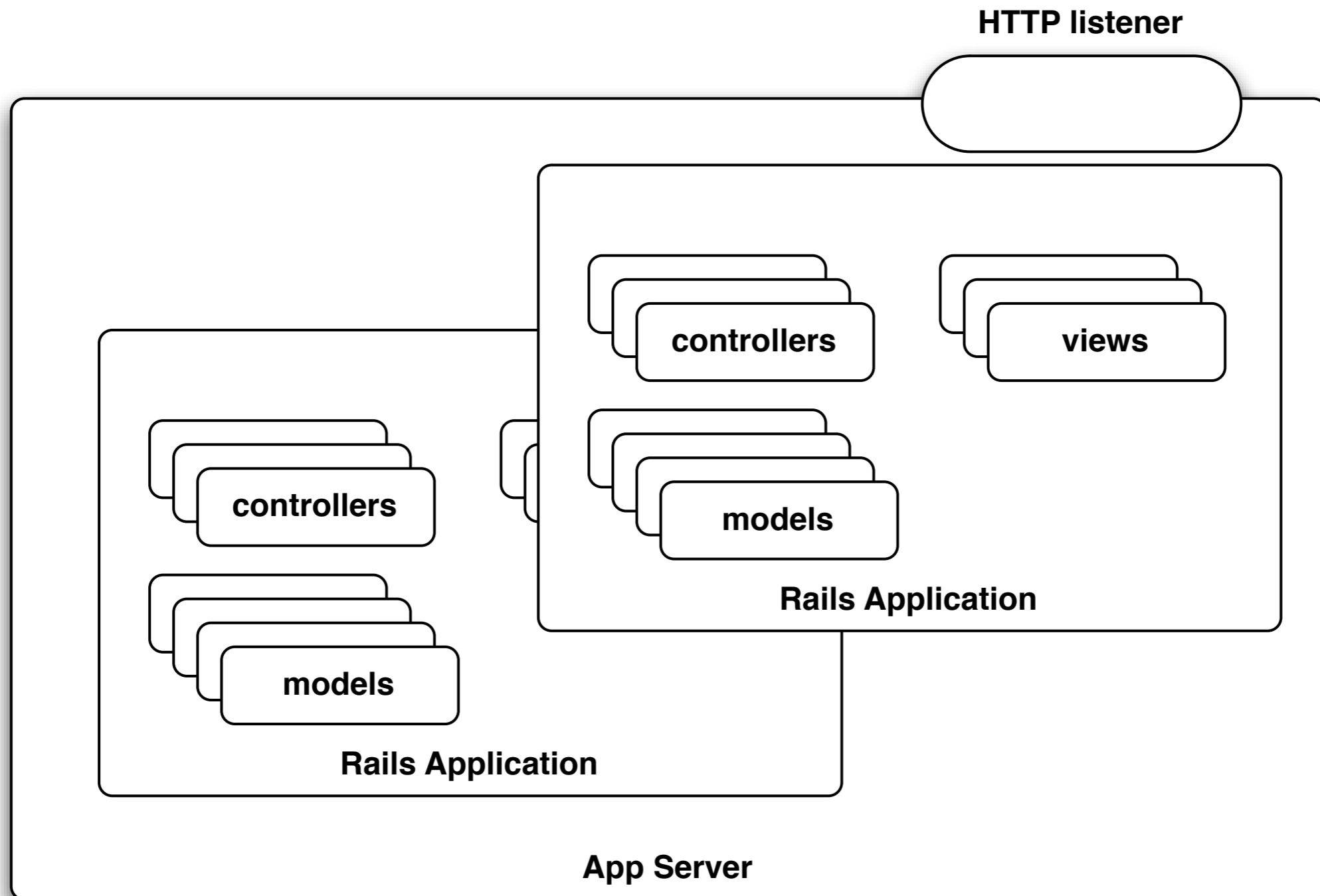


Inversion of Deployment

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

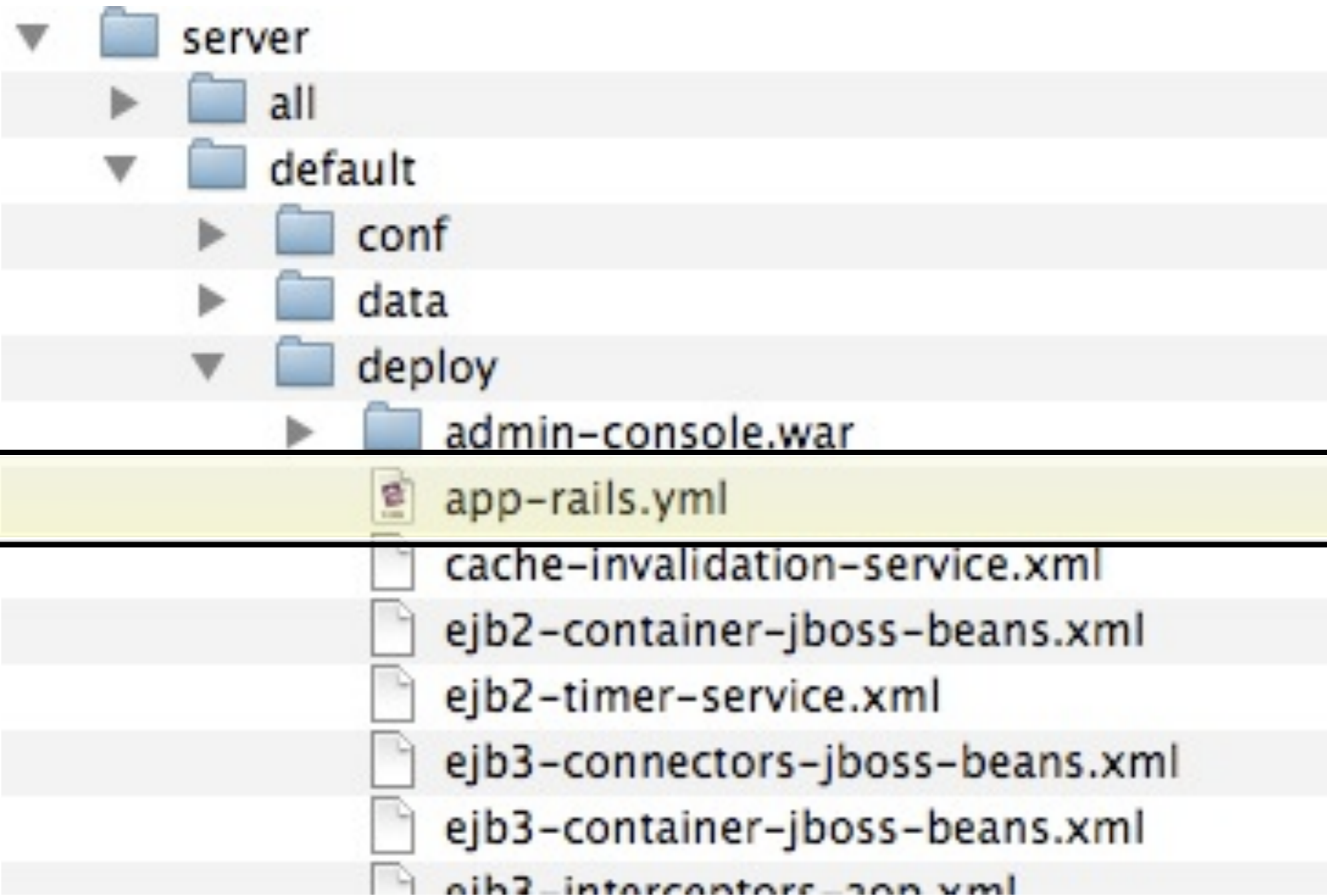
Inversion of Deployment: App server



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



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
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
end
```

Step 2: Scheduled Jobs

```
module GitHub
  class CommitPoller
    include TorqueBox::Jobs::Base
    def run()
      # do work here
    end
  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 do_something(payload={})
    # do work here
  end
end
end
```

Step 3: Async Task Queues

```
class MyQueue
```

```
  include TorqueBox::Queue::Base
```

```
  def do_something(payload={})
```

```
    # do work here
```

```
  end
```

```
end
```

Step 3: Async Task Queues

```
class MyQueue
```

```
  include TorqueBox::Queue::Base
```

```
  def do_something(payload={})
```

```
    # do work here
```

```
  end
```

```
end
```

Step 3: Async Task Queues

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

  def do_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
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
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
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.

**Wait, that was just
Ruby on Rails...**

Anything Rack

- **TorqueBox** uses **Rack** plumbing to handle **RoR**
- ~~Now~~ **Soon** will allow deploying of arbitrary **Rack** applications

config.ru

```
app = lambda{|env|  
  [ 200,  
    {'Content-Type' => 'text/html'},  
    'Hello World']  
}  
run app
```

config.ru

```
app = lambda{ |env|  
  [ 200,  
    {'Content-Type' => 'text/html'},  
    'Hello World']  
  }  
run app
```

config.ru

```
app = lambda{|env|  
  [ 200,  
    {'Content-Type' => 'text/html'},  
    'Hello World']  
}  
run app
```

config.ru

```
app = lambda{|env|
```

```
  [ 200,
```

```
    {'Content-Type' => 'text/html'},
```

```
    'Hello World']
```

```
  }
```

```
run app
```

config.ru

```
app = lambda{|env|  
  [ 200,  
    {'Content-Type' => 'text/html'},  
    'Hello World']  
}  
run app
```

config.ru

```
app = lambda{|env|  
  [ 200,  
    {'Content-Type' => 'text/html'},  
    'Hello World']  
}
```

run app

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

***-rack.yml**

application:

RACK_ROOT: /path/to/myapp

RACK_ENV: production

rackup: config.ru

web:

host: myapp.com

Sinatra!

config.ru

```
require 'app'
```

```
use TorqueBox::Rack::Reloader,  
    File.dirname(__FILE__)
```

```
run Sinatra::Application
```

`config.ru`

`require 'app'`

`use TorqueBox::Rack::Reloader,
File.dirname(__FILE__)`

`run Sinatra::Application`

`config.ru`

```
require 'app'
```

```
use TorqueBox::Rack::Reloader,  
    File.dirname(__FILE__)
```

```
run Sinatra::Application
```

config.ru

```
require 'app'
```

```
use TorqueBox::Rack::Reloader,  
    File.dirname(__FILE__)
```

```
run Sinatra::Application
```

app.rb

```
require 'rubygems'
```

```
require 'sinatra'
```

```
get '/' do
```

```
  "Hello world"
```

```
end
```

app.rb

```
require 'rubygems'
```

```
require 'sinatra'
```

```
get '/' do
```

```
  "Hello world"
```

```
end
```

app.rb

```
require 'rubygems'
```

```
require 'sinatra'
```

```
get '/' do
```

```
  "Hello world"
```

```
end
```

app.rb

```
require 'rubygems'
```

```
require 'sinatra'
```

```
get '/' do
```

```
  "Hello world"
```

```
end
```

Questions?

Deploying Rails Applications to TorqueBox

Why TorqueBox?

Why TorqueBox?

Rails Application



20 Mongrels

Rails Application



20 Mongrels



TorqueBox

Rails Application



Capistrano Deployed

Rails Application



Bundle Deployed

Methods to Deploy

Application Bundle OR Capistrano

Application Bundle

What is an Application Bundle?

Application Bundle

Web Application Archive (WAR) of
your Rails application.

It will be created with a **.rails**
file extension.

Application Bundle

How do I create it?

Application Bundle

App Bundle Rake tasks:

rake torquebox:rails:bundle

rake torquebox:rails:deploy:bundle

Application Bundle

What to do next?

Application Bundle

Drop it in your TorqueBox deploy directory.

OR

Deploy it locally using the Rake tasks using:
rake torquebox:rails:deploy:bundle

Application Bundle

Don't forget...

Application Bundle

config/web.yml

AND

config/rails-env.yml

config/web.yml

Host AND Context

config/web.yml

Example:

```
host: torquebox.org  
context: /
```

config/rails-env.yml

Rails Environment

config/rails-env.yml

Example:

```
RAILS_ENV: production
```

Capistrano

Complete Example:

<http://github.com/torquebox/ballast-rails>

Capistrano

Supports:

daemontools **AND** **init.d**

Capistrano

Don't Forget...

require 'recipes/torquebox'

Questions?