TorqueBox

Mixing Java and Ruby all *willy-nilly*

Bob McWhirter
JBoss Fellow

TorqueBox

Mixing Java and Ruby all *willy-nilly*

Bob McWhirter
JBoss Fellow

Who is Bob?

- JBoss Fellow
- Chief Architect of Middleware Cloud Computing
- Founder of...
  - The Codehaus
  - Drools
  - TorqueBox

Basic Premise

- You can run Ruby in the JVM
  - Compliant
  - Fast
- You can wire existing enterprise Java features into Ruby

Why?

- Ruby has a lot of varied point solutions.
- Traditionally web-centric frameworks.
- Performance issues.

What exactly is **TorqueBox**?

JBoss AS6	TorqueBox Deployers	JRuby

JBoss AS6++

Just as good as regular **JBoss AS6**,
plus the goodness of a **Ruby platform**.

Ruby Platform

- Ruby Rack (1.1.0)
- JNDI
- JMS (HornetQ)
- Quartz
- VFS

Rack for Web

All Rack applications are supported.

Rails, Sinatra, bare Rack apps.

Rack for Web

Applications can be deployed from where-ever they sit on disk.

No bundling/jarring is required.

Rack for Web

Using Rack reloading extensions, or Rails' own development mode, applications can be modified while running.

Without redeploying.

Deploying Rails apps

$JBOSS_HOME/server/default/deploy/myapp-rails.yml

```
---
application:
  RAILS_ROOT: /Users/bob/applications/my_app
  RAILS_ENV: development
web:
  context: /
```

(But you can bundle)

```
jar cvf myapp.rails -C myapp/ .
```

Everything else

Due to lack of specs/standards, we've invented our own APIs for bindings to JNDI, JMS, Quartz, etc.

Messaging

queues.yml

```
/queues/foo:
  durable: true

/queues/bar:
  durable: false
```

Messaging

messaging.tq

```
subscribe MyConsumer,
          '/queues/foo', :filter=>'...',
          :config=>{
            :answer=>42
          }
```

In a Rails app...

```
app/
    processors/
        my_consumer.rb
config/
    messaging.tq
    queues.yml
    topics.yml
```

```
deploy/
    queues.yml
    topics.yml
```

Either

MyConsumer

my_consumer.rb

```
class MyConsumer < TorqueBox::Message::MessageProcessor

  def on_message(body)
    puts "Processing #{body}”
  end

end
```

MyConsumer #2

my_consumer.rb

```
class MyConsumer

  def process!(jms_message)
    puts "Processing #{jms_message.body}”
  end

end
```

MyConsumer #3

my_consumer.rb

```
class MyConsumer < TorqueBox::Message::MessageProcessor

  def on_message(body)
    puts "Processing #{body} of #{message}”
  end

end
```

A client

my_client.rb

```
#!/usr/bin/env jruby

require 'torquebox/messaging/client'

TorqueBox::Messaging::Client.connect( true, :auto,
                                      'localhost' ) do |session|
  queue    = session.createQueue( '/queues/foo' )
  producer = session.createProducer( queue )
  message  = session.createTextMessage( "hello #{Time.now}" )
  producer.send( message )
  session.commit
 end
```

A client

my_client.rb

```
#!/usr/bin/env jruby

require 'torquebox/messaging/client'

TorqueBox::Messaging::Client.connect( true, :auto,
                                      'localhost' ) do |session|
  queue    = session.createQueue( '/queues/foo' )
  producer = session.createProducer( queue )
  message  = session.createTextMessage( "hello #{Time.now}" )
  producer.send( message )
  session.commit
 end
```

A client

my_client.rb

```
#!/usr/bin/env jruby

require 'torquebox/messaging/client'

TorqueBox::Messaging::Client.connect( true, :auto,
                                      'localhost' ) do |session|
  queue    = session.createQueue( '/queues/foo' )
  producer = session.createProducer( queue )
  message  = session.createTextMessage( "hello #{Time.now}" )
  producer.send( message )
  session.commit
 end
```

A client

my_client.rb

```
#!/usr/bin/env jruby

require 'torquebox/messaging/client'

TorqueBox::Messaging::Client.connect( true, :auto,
                                      'localhost' ) do |session|
  queue    = session.createQueue( '/queues/foo' )
  producer = session.createProducer( queue )
  message  = session.createTextMessage( "hello #{Time.now}" )
  producer.send( message )
  session.commit
end
```

A client

my_client.rb

```
#!/usr/bin/env jruby

require 'torquebox/messaging/client'

TorqueBox::Messaging::Client.connect( true, :auto,
                                      'localhost' ) do |session|
  queue    = session.createQueue( '/queues/foo' )
  producer = session.createProducer( queue )
  message  = session.createTextMessage( "hello #{Time.now}" )
  producer.send( message )
  session.commit
 end
```

What if I **just want a farm of processors**, not lots of full AS instances?

JBossMC + VDF	TorqueBox Deployers	JRuby

Messaging outside of AS

`trq-message-broker`

- Ruby API for firing up HornetQ

`trq-message-processor-host`

- Container for processors.

trq-message-broker

```
$ trq-message-broker -s \
--deploy queues.yml
```

trq-message-processor-host

```
$ trq-message-processor-host \
    -N naming.foocorp.com \
    --deploy messaging.tq
```

What’s going on?

JBossMC and Ruby

- Andrew Lee Rubinger's **bootstrap-vdf-minimal**
- Large-grained "enablers"
- All from Ruby

Foundation

```
require 'torquebox/container/foundation'

container = TorqueBox::Container::Foundation.new

container.start
```

Foundation jboss-beans

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">


  <bean name="Ruby" class="org.jruby.Ruby">
    <constructor factoryClass="org.jruby.Ruby" factoryMethod="getGlobalRuntime"/>
  </bean>

  <bean name="JRubyClassLoader">
    <constructor factoryMethod="getJRubyClassLoader">
      <factory bean="Ruby"/>
    </constructor>
  </bean>

  <bean name="RubyRuntimeFactory" class="org.torquebox.interp.core.SingletonRubyRuntimeFactory">
    <property name="ruby">
      <inject bean="Ruby"/>
    </property>
  </bean>

  <bean name="RuntimePoolDeployer" class="org.torquebox.interp.deployers.RuntimePoolDeployer"/>

</deployment>
```

Foundation jboss-beans

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">


  <bean name="Ruby" class="org.jruby.Ruby">
    <constructor factoryClass="org.jruby.Ruby" factoryMethod="getGlobalRuntime"/>
  </bean>

  <bean name="JRubyClassLoader">
    <constructor factoryMethod="getJRubyClassLoader">
      <factory bean="Ruby"/>
    </constructor>
  </bean>

  <bean name="RubyRuntimeFactory" class="org.torquebox.interp.core.SingletonRubyRuntimeFactory">
    <property name="ruby">
      <inject bean="Ruby"/>
    </property>
  </bean>

  <bean name="RuntimePoolDeployer" class="org.torquebox.interp.deployers.RuntimePoolDeployer"/>

</deployment>
```

Foundation jboss-beans

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">


  <bean name="Ruby" class="org.jruby.Ruby">
    <constructor factoryClass="org.jruby.Ruby" factoryMethod="getGlobalRuntime"/>
  </bean>

  <bean name="JRubyClassLoader">
    <constructor factoryMethod="getJRubyClassLoader">
      <factory bean="Ruby"/>
    </constructor>
  </bean>

  <bean name="RubyRuntimeFactory" class="org.torquebox.interp.core.SingletonRubyRuntimeFactory">
    <property name="ruby">
      <inject bean="Ruby"/>
    </property>
  </bean>

  <bean name="RuntimePoolDeployer" class="org.torquebox.interp.deployers.RuntimePoolDeployer"/>

</deployment>
```

Foundation jboss-beans

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">


  <bean name="Ruby" class="org.jruby.Ruby">
    <constructor factoryClass="org.jruby.Ruby" factoryMethod="getGlobalRuntime"/>
  </bean>

  <bean name="JRubyClassLoader">
    <constructor factoryMethod="getJRubyClassLoader">
      <factory bean="Ruby"/>
    </constructor>
  </bean>

  <bean name="RubyRuntimeFactory" class="org.torquebox.interp.core.SingletonRubyRuntimeFactory">
    <property name="ruby">
      <inject bean="Ruby"/>
    </property>
  </bean>

  <bean name="RuntimePoolDeployer" class="org.torquebox.interp.deployers.RuntimePoolDeployer"/>

</deployment>
```

Message-Processor Host

```
require 'torquebox/container/foundation'

container = TorqueBox::Container::Foundation.new

container.enable( TorqueBox::Messaging::MessageProcessorHost )

container.start
```

MPH jboss-beans.xml

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">
  <classloader><inject bean="JRubyClassLoader"/></classloader>
  <bean name="MessageProcessorDeployer"
        class="org.torquebox.messaging.deployers.MessageProcessorDeployer"/>

  <bean name="MessagingRuntimePoolDeployer"
        class="org.torquebox.messaging.deployers.MessagingRuntimePoolDeployer">
    <property name="instanceFactoryName">RubyRuntimeFactory</property>
  </bean>

</deployment>
```

MPH jboss-beans.xml

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">
  <classloader><inject bean="JRubyClassLoader"/></classloader>
  <bean name="MessageProcessorDeployer"
        class="org.torquebox.messaging.deployers.MessageProcessorDeployer"/>

  <bean name="MessagingRuntimePoolDeployer"
        class="org.torquebox.messaging.deployers.MessagingRuntimePoolDeployer">
    <property name="instanceFactoryName">RubyRuntimeFactory</property>
  </bean>

</deployment>
```

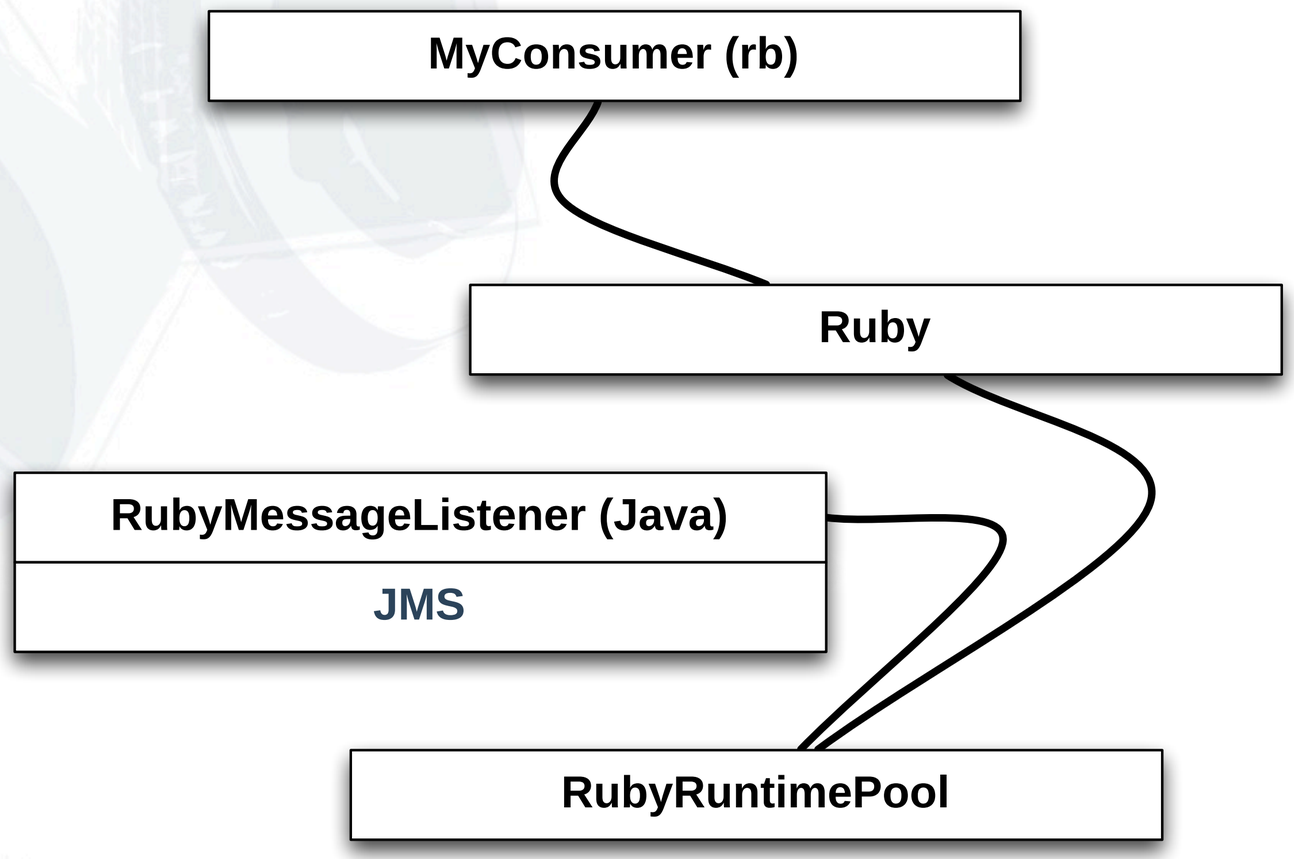
MPH jboss-beans.xml

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">
  <classloader><inject bean="JRubyClassLoader"/></classloader>
  <bean name="MessageProcessorDeployer"
        class="org.torquebox.messaging.deployers.MessageProcessorDeployer"/>

  <bean name="MessagingRuntimePoolDeployer"
        class="org.torquebox.messaging.deployers.MessagingRuntimePoolDeployer">
    <property name="instanceFactoryName">RubyRuntimeFactory</property>
  </bean>

</deployment>
```

Pathway


MyConsumer (rb)
Ruby
RubyMessageListener (Java)
JMS
RubyRuntimePool

Naming

A lot of things require JNDI.

Naming

In addition to enabling components, configuration can be included.

Local Naming

```
container.enable( TorqueBox::Naming::NamingService ) do |config|
    config.export = false
end
```

Local Naming

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">

  <bean name="LookupExecutor" class="java.util.concurrent.ThreadPoolExecutor">
    <constructor factoryMethod="newFixedThreadPool"
                 factoryClass="java.util.concurrent.Executors">
      <parameter>2</parameter>
    </constructor>
    <stop method="shutdown"/>
  </bean>

  <bean name="Naming" class="org.jnp.server.NamingBeanImpl"/>

</deployment>
```

Exported Naming

```
container.enable( TorqueBox::Naming::NamingService ) do |config|
    config.host = "my-other-host.foocorp.com"
    config.port = 10990
end
```

Exported Naming

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">

  <bean name="JNDIServer" class="org.jnp.server.Main">
    <property name="namingInfo">
      <inject bean="Naming"/>
    </property>
    <property name="port">${jnp.port:1099}</property>
    <property name="bindAddress">${jnp.host:localhost}</property>
    <property name="rmiPort">${jnp.rmiPort:1098}</property>
    <property name="rmiBindAddress">${jnp.host:localhost}</property>
    <property name="lookupExector">
      <inject bean="LookupExecutor"/>
    </property>
  </bean>

</deployment>
```

Exported Naming

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">

  <bean name="JNDIServer" class="org.jnp.server.Main">
    <property name="namingInfo">
      <inject bean="Naming"/>
    </property>
    <property name="port">${jnp.port:1099}</property>
    <property name="bindAddress">${jnp.host:localhost}</property>
    <property name="rmiPort">${jnp.rmiPort:1098}</property>
    <property name="rmiBindAddress">${jnp.host:localhost}</property>
    <property name="lookupExector">
      <inject bean="LookupExecutor"/>
    </property>
  </bean>

</deployment>
```

VDF bootstrap allows for **TorqueBox** bits to be used in other environments.

Allows us to boot up an

absolute minimal AS,

in 1 second.

The same code paths as **AS**.

JBoss VFS

FS v. VFS

Regular FS

```
home/
  bob/
    my.jar
```

mount →

VFS

```
home/
  bob/
    my.jar/
      WEB-INF/
        lib/
```

mount →

```
          another.jar/
            META-INF/
              manifest.xml
```

Why do we like **VFS**?

Because there's more than just the **filesystem**

The problem with Ruby

```
__FILE__
```

```
/path/to/this/file.rb
```

For Instance...

```
Dir[ "#{File.dirname(__FILE__)}/*.xml" ]
```

__FILE__

```
vfs:/path/to/the/current/file.rb
```

Making Ruby VFS-aware

Kernel

```
open( "vfs:/path/to/some.jar/some/content.txt" )
```

Dir

```
Dir[ "vfs:/path/to/some.jar/some/*.txt" ]
```

File

```
File.exist?( "vfs:/path/to/some.jar/some/content.txt" )
File.read( "vfs:/path/to/some.jar/some/content.txt" )
```

Now Ruby doesn’t care that your filesystem is actually VFS.

Bundles

We want Ruby to be happy with VFS for when you deploy bundles and hope to farm.

But not warbling

Warbler builds a “normal” JavaEE .war.

So, why bother?

Threading

Unlike C-Ruby, supports **real** threading.

Management

RHQ/JON, stuff you're used to.

Integration

HornetQ, Quartz, etc, already available to your apps.

Gotchas

Native gems

Native gems are not supported, except **FFI**.

Threading

While **TorqueBox** gives you real threads, many existing Ruby libraries **suck**.

Thanks!

Q&A

Resources

```
http://github.com/torquebox/   # Code
http://torquebox.org/          # Project

#torquebox                     # IRC

@torquebox                     # Twitter
@bobmcwhirter
```