

**TorqueBox**

# **The Ruby Application Platform**

**1.0.0.Beta18**

by JBoss by Red Hat

---

---

---

<b>1. What is TorqueBox?</b> .....	1
1.1. Built upon JBoss AS5 .....	1
1.2. Built upon JRuby .....	1
1.3. Open-Source .....	1
<b>2. Differences between TorqueBox &amp; Traditional Ruby</b> .....	3
2.1. The "application platform" concept .....	3
<b>3. Installation</b> .....	5
3.1. Installation using Complete Binary Distribution .....	5
3.1.1. Ensure you have Java 6 .....	5
3.1.2. Get the latest version of TorqueBox binary package .....	5
3.1.3. Unzip it somewhere handy .....	5
3.2. Installing the TorqueBox deployer into an existing JBoss AS 5 .....	6
<b>4. Scheduled Jobs</b> .....	9
4.1. What Are Scheduled Jobs? .....	9
4.2. Ruby Job Classes .....	9
4.3. <code>TorqueBox::Jobs::Base</code> module .....	10
4.3.1. Logging .....	10
4.4. Scheduling Jobs .....	10
4.4.1. Ruby-on-Rails: <code>\$RAILS_ROOT/config/jobs.yml</code> .....	11
4.4.2. <code>jobs.yml</code> Format .....	11
<b>5. Task Queues</b> .....	13
5.1. What Are Task Queues? .....	13
5.2. Ruby Task Queue Classes .....	13
5.3. <code>TorqueBox::Queues::Base</code> module .....	13
5.4. Logging .....	14
5.5. Enqueuing work .....	14
5.5.1. Using <code>TorqueBox::Queues</code> .....	14
5.5.2. Using Raw JMS .....	15
<b>6. Cryptography</b> .....	17
6.1. Cryptographic Key Stores .....	17
6.2. Configure Key Stores: <code>crypto.yml</code> .....	17
6.2.1. Ruby-on-Rails and <code>crypto.yml</code> .....	17
6.3. Managing Key Stores .....	18
6.3.1. Create a New Key Store .....	18
6.3.2. Create a New Key Pair .....	18
6.3.3. Inspect the Key Store .....	19
<b>7. SOAP Endpoints</b> .....	21
7.1. Introduction to SOAP & Data-Binding .....	21
7.2. Ruby Endpoints .....	21
7.2.1. Basic Implementation & Configuration .....	22
7.2.2. Operation Implementation .....	24
7.3. Data-Binding .....	25
7.3.1. Basics .....	26
7.3.2. Collections .....	26

<b>8. Telecom (SIP and Media)</b> .....	29
8.1. Making phone calls .....	29
8.2. Handling phone calls (Ruby Telco Classes) .....	30
8.3. Media Support .....	31
<b>9. Ruby-on-Rails Support</b> .....	35
9.1. Ruby-on-Rails .....	35
9.2. Rails on TorqueBox .....	35
9.3. Preparing your Rails application .....	35
9.3.1. Using the application template .....	35
9.3.2. Manually configuring an application .....	36
9.4. Scheduled Jobs .....	37
9.5. Task Queues .....	37
9.6. Deploying .....	37
9.6.1. Deploy using the Rake tasks .....	37
9.6.2. Deploy using a descriptor .....	39
9.6.3. Deployment using a partial descriptor .....	41
9.6.4. Deployment using a bundle .....	42
9.7. Control thread concurrency using <code>pooling.yml</code> .....	43
9.7.1. Pooling for web requests .....	43
9.8. Integrate additional JEE components .....	43
9.9. Controlling the TorqueBox Server .....	43
9.9.1. Using Rake .....	43
<b>10. Capistrano Support</b> .....	45
10.1. Capistrano .....	45
10.2. App-Server separate from App .....	45
<b>11. Rack Support</b> .....	47
11.1. General <code>config.ru</code> Support .....	47
11.2. Rack deployment descriptor ( <code>*-rack.yml</code> ) .....	47
11.2.1. Location & Naming .....	47
11.2.2. Contents of the descriptor .....	47
A. GNU Lesser General Public License version 3 .....	51
B. MIT License .....	55

# What is TorqueBox?

TorqueBox provides an enterprise-grade environment that not only provides complete Ruby-on-Rails compatibility, but also goes beyond the functionality offered in traditional Rails environments.

## 1.1. Built upon JBoss AS5

Instead of building a Ruby Application Platform from the ground-up, TorqueBox leverages the existing ninja-grade functionality JBoss has been shipping for years in the JBoss Application Server. JBoss AS includes high-performance clustering, caching and messaging functionality. By building Ruby capabilities on top of this foundation, your Ruby applications gain more capabilities right out-of-the-box.

## 1.2. Built upon JRuby

JRuby is a fast, compliant implementation of the Ruby language upon the Java Virtual Machine. Pure Ruby applications run un-modified within the JRuby interpreter. By binding JRuby to the components within JBoss, their functionality is exposed in a manner suitable to Rubyists.

## 1.3. Open-Source

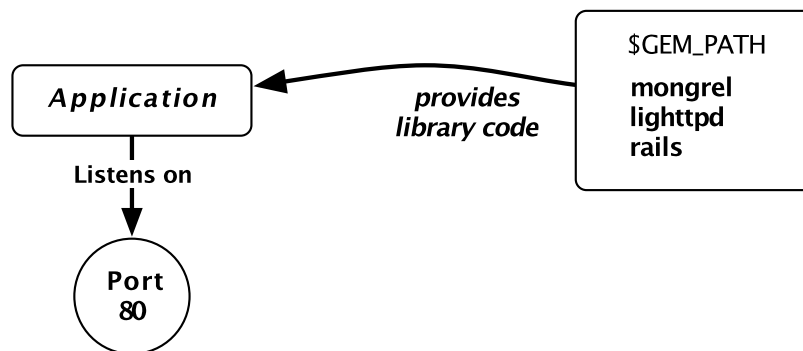
TorqueBox is a product of the JBoss Community, and is completely open-source software. TorqueBox is licensed under the LGPL. You may download the binaries or the source-code, modify it if you desire, and use it, even for profit, without any licensing costs.



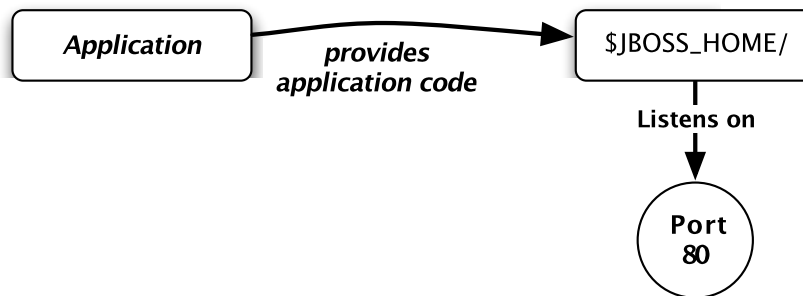
# Differences between TorqueBox & Traditional Ruby

## 2.1. The "application platform" concept

Traditionally, Ruby applications were responsible for their services from the ground-up. You literally ran the application. It would import support libraries to handle HTTP listening, for example.



An application platform provides the foundations for any and all application functionality. The deliverable application itself does not need to handle the networking layers, the messaging facilities or the clustering logic. This is provided to the application "for free".





# Installation

## 3.1. Installation using Complete Binary Distribution

The latest Complete Binary Distribution contains:

- The TorqueBox server, ready-to-run
- A complete JRuby installation

### 3.1.1. Ensure you have Java 6

TorqueBox requires Java JDK 6.

To determine which version, if any, is installed on your system, at a command-line, attempt to run the `java` command with the `-version` argument.

```
$ java -version
java version "1.6.0_07"
Java(TM) SE Runtime Environment (build 1.6.0_07-b06-153)
Java HotSpot(TM) 64-Bit Server VM (build 1.6.0_07-b06-57, mixed mode)
```

If the version is at least 1.6, your version of Java is sufficient.

If you have no Java installed, or a version less than 1.6, you'll need to install a Java Development Kit. For many systems, it is easy to install the open-source OpenJDK.

For installation on Ubuntu, Fedora, OpenSuse, or Debian, please refer to the [installation instructions provided](http://openjdk.java.net/install/) [http://openjdk.java.net/install/] by the OpenJDK project. If you find a `java` on your system, ensure that it is not actually `gcj`. The `gcj` is insufficient for running the TorqueBox server.

For Apple OSX systems, Apple provides a JDK version 6.

### 3.1.2. Get the latest version of TorqueBox binary package

You can obtain the latest version of TorqueBox from the TorqueBox repository. As of this writing, the latest version is 1.0.0.Beta18.

<http://repository.torquebox.org/maven2/releases/org/torquebox/torquebox-bin/1.0.0.Beta18/torquebox-bin-1.0.0.Beta18.zip> [http://repository.torquebox.org/maven2/releases/org/torquebox/torquebox-bin/torquebox-bin-1.0.0.Beta18.zip]

### 3.1.3. Unzip it somewhere handy

We'll install TorqueBox under your user's `$HOME` directory.

```
$ unzip torquebox-bin-1.0.0.Beta18.zip
$ cd torquebox-1.0.0.Beta18
```

Before using the TorqueBox server, you must set up your environment. To make it easier to upgrade without having to reconfigure your environment, it is useful to create a symlink to the versioned directory produced when you unpackaged the distribution.

```
$ ln -s torquebox-bin-1.0.0.Beta18 torquebox-current
```

Next, `$TORQUEBOX_HOME`, `$JBASS_HOME` and `$JRUBY_HOME` need to be set, and adjusting your `$PATH` will make working with the package easier. You can either run the following commands each time on the command-line, or add them to your `.bash_profile`.

First, the various `$X_HOME` variables are set so that each subsystem can find its supporting files.

```
export TORQUEBOX_HOME=$HOME/torquebox-current
export JBASS_HOME=$TORQUEBOX_HOME/jboss
export JRUBY_HOME=$TORQUEBOX_HOME/jruby
```

Next, we make sure that JRuby's binaries are first in our executable `$PATH`, before any previously-installed Ruby packages.

```
export PATH=$JRUBY_HOME/bin:$PATH
```

By doing this, commands such as `rake`, `gem`, and `rails` will load from the TorqueBox-provided JRuby installation.

## 3.2. Installing the TorqueBox deployer into an existing JBoss AS 5

To enable an existing JBoss AS5 installation for Ruby support, a single JAR file, the *TorqueBox Deployer* needs to be installed.

Before installing the TorqueBox Deployer into an existing JBoss AS, you need to ensure that JBoss is not running. Once the AS is stopped, follow the directions below to enable your AS to become Ruby-compatible.

1. Locate the latest version of the TorqueBox Deployer JAR from the TorqueBox repository:

- [Official releases](http://repository.torquebox.org/maven2/releases/org/torquebox/torquebox-core/) [http://repository.torquebox.org/maven2/releases/org/torquebox/torquebox-core/]
- [Interim snapshot releases](http://repository.torquebox.org/maven2/snapshots/org/torquebox/torquebox-core/) [http://repository.torquebox.org/maven2/snapshots/org/torquebox/torquebox-core/]

2. Download the JAR with the name in the format of *torquebox-core-1.0.0.Beta18-deployer.jar*.

3. Determine the location of your JBoss AS5 server and the configuration you plan to use. For clustering the *all* configuration is required. For non-clustering, the *web* or *default* configurations are acceptable.

The `$JBOSS_HOME` environment variable should be set to the top of your JBoss AS5 installation directory. By convention, the `$JBOSS_CONF` environment variable should contain the name of the configuration you are using.

4. Copy the deployer JAR into the `deployers/` directory of the configuration.

```
$ cp torquebox-core-1.0.0.Beta18-deployer.jar  
$JBOSS_HOME/server/$JBOSS_CONF/deployers
```



# Scheduled Jobs

## 4.1. What Are Scheduled Jobs?

Scheduled jobs are simply components that execute on a possibly-recurring schedule instead of in response to user interaction. Scheduled jobs fire asynchronously, outside of the normal web-browser thread-of-control. Scheduled jobs have full access to the entire Ruby environment. This allows them to interact with database models and other application functionality.

## 4.2. Ruby Job Classes

Each scheduled job maps to exactly one Ruby class. The path and filename should match the class name of the job contained in the file.

File name	Class name
mail_notifier.rb	MailNotifier
mail/notifier.rb	Mail::Notifier

### Example 4.1. Skeleton scheduled job class (`mail/notifier.rb`)

```
module Mail
  class Notifier

    # implementation goes here

  end
end
```

Each job class should implement a no-argument `run()` method to perform the work when fired.

### Example 4.2. Scheduled job implementation (`mail/notifier.rb`)

```
module Mail
  class Notifier

    def run()

      # perform work here

    end

  end
end
```

From within the class's `run()` method, the full application environment is available.

### 4.3. TorqueBox::Jobs::Base module

The optional `TorqueBox::Jobs::Base` module provides access to helpful functionality. It is not required to be included in your Ruby job class. By including it, though, extra functionality is introduced into your classes.

#### Example 4.3. Using `TorqueBox::Jobs::Base` module

```
require 'torquebox/jobs/base'

module Mail
  class Notifier
    include TorqueBox::Jobs::Base

    def run()

      # Use the logger provided by TorqueBox::Jobs::Base
      log.info( "Executing the job" )

      # perform work here
    end
  end
end
```

#### 4.3.1. Logging

To gain access to a logging device, the `log()` method is available. Messages of various level can be logged.

Method	Use
<code>trace()</code>	Tracing program execution
<code>debug()</code>	Development-time debug information
<code>info()</code>	Information messages for the user
<code>warn()</code>	Warnings for the user
<code>error()</code>	Severe errors during execution

The log messages will be logged in the normal `server.log`.

```
10:02:35,074 INFO [Notifier] Executing the job
10:02:40,074 INFO [Notifier] Executing the job
```

### 4.4. Scheduling Jobs

The job schedule defines the time(s) that a job should execute. This may be defined to be single point in time, or more often, as recurring event. The job schedule is defined with a file named `jobs.yml`.

#### 4.4.1. Ruby-on-Rails: \$RAILS\_ROOT/config/jobs.yml

For Ruby-on-Rails applications, the schedule should be placed at \$RAILS\_ROOT/config/jobs.yml.

#### 4.4.2. jobs.yml Format

Within jobs.yml, a block of information is provided for each job. The block starts with arbitrary name for the job. Each block must also define the job class and the schedule specification. Optionally a description may be provided.

#### Example 4.4. Example jobs.yml

```
mail_notifier
  job:      Mail::Notifier
  cron:     0 */5 * * * ?
  description: Deliver queued mail notifications
```

The cron attribute should contain a typical crontab-like entry. It is composed of 7 fields (6 are required).

Seconds	Minutes	Hours	Day of Month	Month	Day of Week	Year
0-59	0-59	0-23	1-31	1-2 or JAN-DEC	1-7 or SUN-SAT	1970-2099 (optional)

For several fields, you may denote subdivision by using the forward-slash (/) character. To execute a task every 5 minutes, \*/5 in the minutes field would specify this condition.

Spans may be indicated using the dash (-) character. To execute a task Monday through Friday, MON-FRI should be used in the day-of-week field.

Multiple values may be separated using the comma (,) character. The specification of 1,15 in the day-of-month field would result in the job firing on the 1st and 15th of each month.

Either day-of-month or day-of-week must be specified using the ? character, since specifying both is contradictory.



# Task Queues

## 5.1. What Are Task Queues?

Task queues allow for asynchronous execution of code outside of the calling (or client) thread of execution.

Queues are normally used for long-running tasks that are triggered by some user interaction. This may include tasks such as sending an email, fulfilling a financial trade, or executing a long-running report against database.

## 5.2. Ruby Task Queue Classes

A single task queue maps to an individual Ruby class. Each queue may support a variety of related tasks. Each task is represented by a method that takes a single *payload* argument.

### Example 5.1. Example task queue class (forex/execute\_trade.rb)

```
module Forex
  class ExecuteTrade

    def buy(payload={})
      # perform work
    end

    def sell(payload={})
      # perform work
    end

  end
end
```

## 5.3. TorqueBox::Queues::Base module

The optional `TorqueBox::Queues::Base` module provides access to helpful functionality, including simplified client access for enqueueing work. It is not required to be included in your Ruby job class. By including it, though, extra functionality is introduced into your classes.

### Example 5.2. Using TorqueBox::Queues::Base module

```
require 'torquebox/queues/base'

module Forex
  class ExecuteTrade

    include TorqueBox::Queues::Base
```

```

def buy(payload={})
  log.info( "Buy #{payload[:quantity]}" )
end

def sell(payload={})
  log.info( "Sell #{payload[:quantity]}" )
end

end
end

```

The task queue class has full access to the Ruby environment, allowing you to use other models and classes in from the application. Some frameworks, such as ActiveRecord even allow inclusion of models in the payload delivered to the task method.

## 5.4. Logging

To gain access to a logging device, the `log()` method is available. Messages of various level can be logged.

Method	Use
<code>trace()</code>	Tracing program execution
<code>debug()</code>	Development-time debug information
<code>info()</code>	Information messages for the user
<code>warn()</code>	Warnings for the user
<code>error()</code>	Severe errors during execution

The log messages will be logged in the normal `server.log`.

```

10:02:35,074 INFO [ExecuteTrade] Buy 100
10:02:40,074 INFO [ExecuteTrade] Sell 100

```

## 5.5. Enqueuing work

### 5.5.1. Using `TorqueBox::Queues`

The `TorqueBox::Queues` module is provided in order to enqueue work. A class method `enqueue(...)` is available send a payload to any action on any queue.

The `enqueue(...)` method takes three parameters: `queue_name`, `task_name` and `payload`.

#### Example 5.3. Enqueuing work with `enqueue(...)`

```

# load the task queues class

```

```
require 'torquebox/queues'

class TradeController < ApplicationController

  def buy
    # call enqueue(...) with task-name and payload
    TorqueBox::Queues.enqueue( 'Forex::ExecuteTrade', :buy, { :quantity=>100
  } )
  end
end
```

## 5.5.2. Using Raw JMS

The Ruby task queues are implemented using the Java Messaging Service (JMS). Each class maps to a single JMS destination. The destination may be retrieved using JNDI.

### 5.5.2.1. JMS Destinations

Each destination is named according to the pattern of `appname.Converted.Class.Name`.

If your application is named "myapp", the JMS destination matching the `Forex::ExecuteTrade` task class would be `myapp.Forex.ExecuteTrade`.

### 5.5.2.2. Message format

An `javax.jms.ObjectMessage` is expected to contain the raw payload. The payload may optionally be a marshalled object if the `IsRubyMarshal` boolean property is set to `true`.

The `TaskName` string property of the message must be set to the name of the task method which should handle the message.



# Cryptography

## 6.1. Cryptographic Key Stores

Cryptographic key stores may be configured for the application and used by any service requiring key material. Keystores themselves are encrypted to secure the key material itself.

## 6.2. Configure Key Stores: `crypto.yml`

A file named `crypto.yml` should contain a section for each keystore you wish to make available within your application.

Each key store defined within `crypto.yml` must provide a path to the store, along with the password required to access the store.

### Example 6.1. Example `crypto.yml`

```
truststore:
  store: truststore.jks
  password: foobar

keystore:
  store: keystore.jks
  password: foobar
```

Each keystore is labelled with an identifier used to access it from other services. While any number of key stores may be configured with arbitrary identifiers, many services look for specific key stores by default.

Some services, such as SOAP Endpoints attempt to use a key store named `truststore` if inbound signature verification is enabled. Likewise, the key store named `keystore` is the default for signing outbound responses.

If the path to the keystore in the `store` parameter is an absolute path, it will be used as specified. If the path is relative, the application framework is responsible for defining those semantics.

### 6.2.1. Ruby-on-Rails and `crypto.yml`

To use cryptographic key stores from your Ruby-on-Rails application, place `crypto.yml` in your application's `config/` directory. Relative paths to key stores specified in the `store` parameter are relative to the directory `$RAILS_ROOT/auth/`.

```
RAILS_ROOT/
  config/
    crypto.yml
  auth/
```

```
truststore.jks
keystore.jks
```

### 6.3. Managing Key Stores

Key stores are kept in the Java Key Store format and managed using the `keytool` command-line utility that ships with the JDK.

#### 6.3.1. Create a New Key Store

When you generate or import your first key into the key store, it will be created as needed. The path to the key store is specified using the `-keystore` option.

```
$ keytool -keystore auth/keystore.jks
```

When a key store is first created while importing or generating a new key, you will be prompted to provide and verify the password to protect the entire keystore. This is the same password you provide through `crypto.yml`.

```
$ keytool -keystore auth/keystore.jks
Enter keystore password:
Re-enter new password:
```

#### 6.3.2. Create a New Key Pair

To create a new key pair to use for signing or encrypting functions, the `-genkey` option is used. The `keytool` utility will prompt you for several pieces of information. Each key is identified by an *alias*. By default `-genkey` attempts to create a key named `mykey`. The `-alias` option may be used to provide a different alias to register the key.

```
$ keytool -keystore truststore.jks -genkey -alias bobmcwhirter
Enter keystore password:
What is your first and last name?
  [Unknown]:  Bob McWhirter
What is the name of your organizational unit?
  [Unknown]:  TorqueBox
What is the name of your organization?
  [Unknown]:  JBoss
What is the name of your City or Locality?
  [Unknown]:  Wytheville
What is the name of your State or Province?
  [Unknown]:  Virginia
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Bob McWhirter, OU=TorqueBox, O=JBoss, L=Wytheville, ST=Virginia, C=US
correct?
  [no]:  yes
```

```
Enter key password for <bobmcwhirter>
    (RETURN if same as keystore password):
Re-enter new password:
$
```

### 6.3.3. Inspect the Key Store

The `-list` option may be used to list the contents of a key store.

```
$ keytool -keystore truststore.jks -list
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

bobmcwhirter, May 13, 2009, PrivateKeyEntry,
Certificate fingerprint (MD5):
 E1:73:2B:34:26:80:CA:55:7C:E8:BC:C6:A2:F6:D0:10
mykey, May 13, 2009, PrivateKeyEntry,
Certificate fingerprint (MD5):
 24:3E:29:E2:6A:5F:AA:24:A2:F3:25:68:B6:6E:92:FF
```



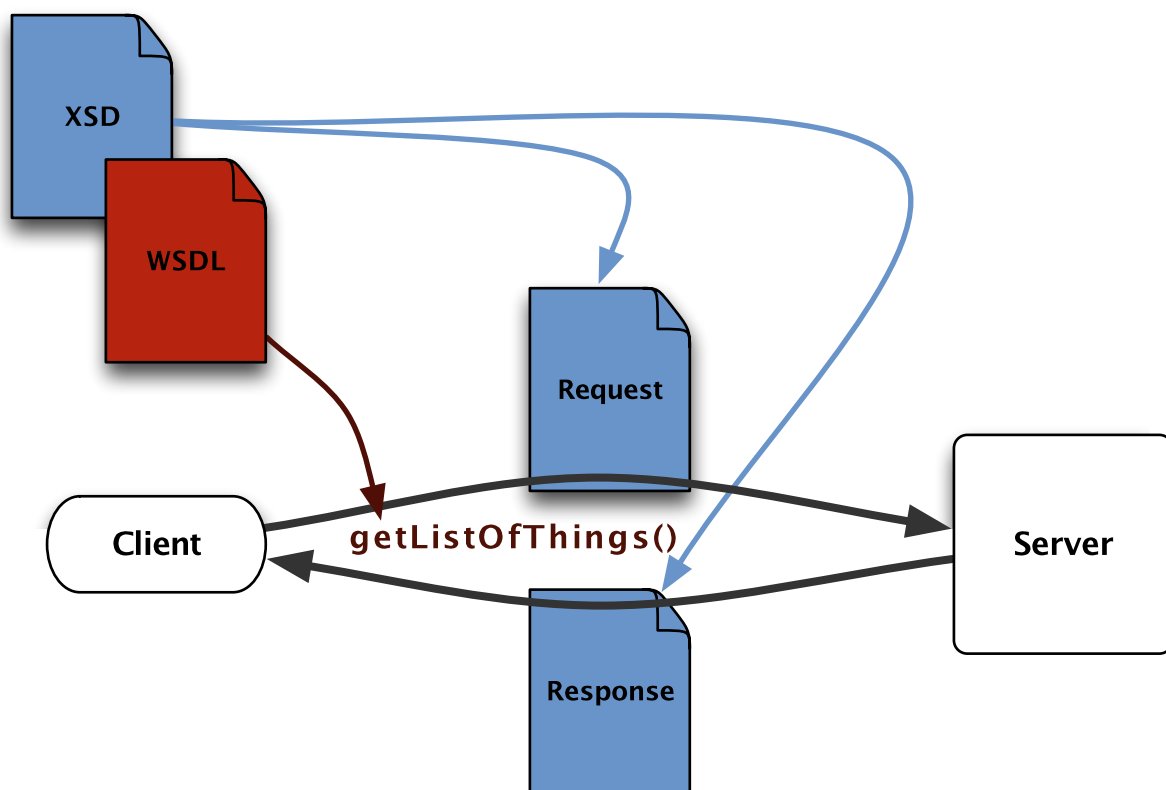
# SOAP Endpoints

## 7.1. Introduction to SOAP & Data-Binding

*Simple Object Access Protocol*, or SOAP, is one component of many service-oriented architecture projects. A SOAP service is defined using a *Web Services Description Language*, or WSDL document. The WSDL document is consumed by both the server and clients as the contract of communication.

Clients invoke operations against the server, optionally involving cryptographic signatures or encryption. Servers respond to the operations synchronously, and produce a response document.

Both the request and response documents are defined using XML Schema Definition language (XSD). Advanced SOAP frameworks use these document descriptions to provide language-native *data-bindings* to facility reading and writing these documents.



## 7.2. Ruby Endpoints

Ruby endpoints may only be used if you already have (or are willing to write) a WSDL service definition document. The Ruby endpoints functionality will not generate a WSDL by inspecting your class. If you do not already have a WSDL, we suggest you take a RESTful strategy for web-services.

### 7.2.1. Basic Implementation & Configuration

Each SOAP service is implemented as a single Ruby class. Each operation provided by the service maps to a similarly-named method on the class.

In order to make SOAP service development easier, include the `TorqueBox::Endpoints::Base` module in your implementation.

#### Example 7.1. Basic SOAP implementation `amazon/ec2_endpoint.rb`

```
require 'torquebox/endpoints/base'

module Amazon
  class Ec2Endpoint

    include TorqueBox::Endpoints::Base

    # method per operation

  end
end
```

#### 7.2.1.1. Endpoints with Ruby-on-Rails: `$RAILS_ROOT/app/endpoints/`

In Ruby-on-Rails applications, the `$RAILS_ROOT/app/endpoints/` directory should contain your WSDL and your Ruby implementation classes.

#### 7.2.1.2. Configuration

The implementation needs to be configured to link up the services it provides with particular WSDL definition and the operations contained within it.

If the WSDL is present in the same directory, and is named the same as the class, minus the `_endpoint` suffix, it will implicitly be used. For example, `amazon/ec2.wsdl` would be used for `amazon/ec2_endpoint.rb` unless otherwise specified using the `wSDL_location` configuration parameter.

Other configuration parameters include `target_namespace`, `port_name`, and `security`. Configuration is accomplished at the class level using an `endpoint_configuration` block.

#### Example 7.2. Service configuration of a Ruby Endpoint

```
require 'torquebox/endpoints/base'

module Amazon
  class Ec2Endpoint
```

```
include TorqueBox::Endpoints::Base

endpoint_configuration do
  wsdl_location      'public/system/wsdl/ec2.wsdl'
  target_namespace  'http://ec2.amazonaws.com/doc/2008-12-01/'
  port_name          'AmazonEC2'
end

# method per operation

end
end
```

Security on the service may be configured using a `security` block within the `endpoint_configuration` block. The security block is broken into two sections: `inbound` and `outbound`.

### Example 7.3. Security configuration of a Ruby Endpoint

```
require 'torquebox/endpoints/base'

module Amazon
  class Ec2Endpoint

    include TorqueBox::Endpoints::Base

    endpoint_configuration do
      wsdl_location      'public/system/wsdl/ec2.wsdl'
      target_namespace  'http://ec2.amazonaws.com/doc/2008-12-01/'
      port_name          'AmazonEC2'

      security do
        inbound do
          verify_timestamp
          verify_signature
        end
      end
    end

    # method per operation

  end
end
```



### Note

Changes to the `endpoint_configuration` block are never re-parsed after your application is first deployed. A redeployment will be required to enable or disable security, or to alter the WSDL location or service bindings.

### 7.2.1.2.1. Inbound Security

Inbound security supports the follow directives:

- `verify_timestamp` to verify that the message is timely, and not a potential reply-attach in progress.
- `verify_signature` to verify the cryptographic signatures of inbound requests.

When verifying signatures, the security conduit will use the *cryptographic key store* named `truststore` by default. To use a different trust store, use the `truststore` option, and provide the identifier to a configured trust store.

```
security do
  inbound do
    truststore :my_other_truststore
    verify_timestamp
    verify_signature
  end
end
```

### 7.2.1.2.2. Outbound Security

Outbound security is currently unsupported.

## 7.2.2. Operation Implementation

Each operation defined by the WSDL may be mapped to a method on the implementation class. Appropriate conversion of `camelCaseNames` to `underscored_names` is performed. For example, the operation `DescribeInstances` would become the method `describe_instances()`.

This fragment of WSDL:

```
<operation name="DescribeInstances">
  <soap:operation soapAction="DescribeInstances" />
  ...
</operation>
```

would map to this method in the Ruby endpoint implementation class:

```
def describe_instances
  # implementation goes here
end
```

Within each operation method, the `request` method provides access to the data-bound request document. See [Data-Binding](#) for more information about how data-binding operates. The return value from the method converted to the appropriate SOAP response document and returned to the caller.

To create an appropriate response object, the `create_response` factory method is available. The object returned matches the default response type for the called operation.

Additionally, if security on inbound messages is enabled, and signature on the request has been verified, the `principal` method will provide access to the X.509 identity of the caller.

```
def describe_instances

  response = create_response

  request.instancesSet.each do |instance_id|
    log.info( "requesting information about instance #{instance_id} by
#{principal}" )
    reservation_info = response.reservationSet.create
    reservation_info.ownerId = ...
  end

  return response

end
```



### Caution

The entire design of the operation methods may be subject to change, pending input from the community. This is a tentative design only.

## 7.3. Data-Binding

Data-binding is the facility to map a data structure defined by XML Schema Description language into reasonable language-natural class definitions. It's much easier working with object trees with attributes than attempting to walk and produce XML documents.

When the WSDL is deployed by TorqueBox, the schema within is analyzed and Ruby classes are generated to support data-binding. This allows the Ruby endpoint implementation class to work purely in terms of *objects* instead of *documents*.

### 7.3.1. Basics

In general, XSD describes a straight-forward tree or graph of objects. Each type becomes a Ruby type, while each element becomes an attribute of some parent object. Primitive types, such as strings, booleans, or numerics are mapped to native Ruby types and vice-versa.

### 7.3.2. Collections

For XSD types that act purely as collections, containing repetition of a single element, the native Ruby Array is extended to provide intuitive access to the elements.

For example, this schema defines a DescribeKeyPairs type, which is seen as a <DescribeKeyPairs> element.

```
<xs:element name="DescribeKeyPairs" type="tns:DescribeKeyPairsType" />

<xs:complexType name="DescribeKeyPairsType">
  <xs:sequence>
    <xs:element name="keySet" type="tns:DescribeKeyPairsInfoType" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DescribeKeyPairsInfoType">
  <xs:sequence>
    <xs:element name="item" type="tns:DescribeKeyPairsItemType"
      minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DescribeKeyPairsItemType">
  <xs:sequence>
    <xs:element name="keyName" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

And example XML document matching this would look like the following.

```
<DescribeKeyPairs>
  <keySet>
    <item>
      <keyName>key-1</keyName>
    </item>
    <item>
      <keyName>key-2</keyName>
    </item>
  </keySet>
</DescribeKeyPairs>
```

Using intelligent collection data-binding, the item element is not transliterated into the Ruby class. The <keySet> element translates into `Array` of `DescribeKeyPairsItemType` objects.

Navigating from a root `DescribeKeyPairsType` object to the key names works intuitively.

```
root.keySet.each do |item|
  puts "Describe key #{item.key}"
end
```



# Telecom (SIP and Media)

Telecom support allows to build powerful converged VoIP applications by providing means for an application to handle/send SIP Messages. By including a [JSR 309](http://jcp.org/en/jsr/detail?id=309) (Media Server Control API) implementation, such as the [Mobicents JSR 309 implementation](http://www.mobicents.org/mms-jsr309-main.html), in your application, you can also control a Media Server remotely and so implement IVR, PBX, Call centers, Conference kind of applications.

Note that Mobicents Sip Servlets 1.0 is needed to be able to use this feature and that it is highly recommended to have knowledge about SIP ([SIP Servlets 1.1 Java Specification](http://jcp.org/en/jsr/detail?id=289)) and Media ([Media Server Control API Java Specification](http://jcp.org/en/jsr/detail?id=309)) to use it effectively.

## 8.1. Making phone calls

To be able to initiate the setup of a call from your TorqueBox application here is what you need to do in your controller :

### Example 8.1. Code Excerpt to set up a call

```
# get the sip factory from the servlet context
@sip_factory =
  $servlet_context.get_attribute('javax.servlet.sip.SipFactory')
puts @sip_factory

# create a new sip application session
@app_session =
  request.env['servlet_request'].get_session().get_application_session();

# create a new sip servlet request to start a call to the sip phone with
# from header equals to "sip:my_jruby_app_rocks@mobicents.org" and the to
# header equals to the sip_uri from the complaint variable
@sip_request = @sip_factory.create_request(@app_session, 'INVITE',
  'sip:my_jruby_app_rocks@mobicents.org', @complaint.sip_uri);

# actually sending the request out to the sip phone
@sip_request.send();
```

This piece of code will create a SIP Request and send it out. But when the user will answer OK, you need to have some code to handle the response and this can be done by defining your own SIP controller

## 8.2. Handling phone calls (Ruby Telco Classes)

In your Ruby or Rails application, all SIP classes should be placed under `$RAILS_ROOT/app/sip/`. No special naming convention is required, but the class name must match the path to the file containing it.

File name	Class name
<code>\$RAILS_ROOT/app/sip/telco_handler</code>	<code>SipHandler</code>
<code>\$RAILS_ROOT/app/sip/telco/handler</code>	<code>Telco::Handler</code>

Additionally, each Telco class must descend, at some point, from `TorqueBox::Sip::Base`.

### Example 8.2. SIP Telco class example (`telco/handler.rb`)

```
require 'torquebox/sip/base'
module Telco
  class Handler < TorqueBox::Sip::Base
    # Handle INVITE request to setup a call by answering 200 OK
    def do_invite(request)
      request.create_response(200).send
    end

    # Handle BYE request to tear down a call by answering 200 OK
    def do_bye(request)
      request.create_response(200).send
    end

    # Handle REGISTER request so that a SIP Phone can register with the
    # application by answering 200 OK
    def do_register(request)
      request.create_response(200).send
    end

    # Handle a successful response to an application initiated INVITE to set
    # up a call (when a new complaint is filed through the web part) by send an
    # acknowledgment
    def do_success_response(response)
      response.create_ack.send
    end
  end
end
```

Note that this is an example but the handler class can extend all the `do_XXX` methods as defined per the Sip Servlets 1.1 Java Specification.

To help you getting started with you can play with and checkout the code of our [sample application demonstrating Announcement and DTMF Detection](http://www.mobicens.org/mss-pure-jruby-telco.html) [http://www.mobicens.org/mss-pure-jruby-telco.html]

## 8.3. Media Support

Mobicents Sip Servlets (and its bundled Media Server) and Torquebox allows you to include Media in your applications as well now.

This allows you to control a remote Media Server using the MGCP protocol and let you create Telco Applications such as IVR, Conferences, Announcement, DTMF recognition, Call Centers, PBX, ...

To be able to use the Mobicents JSR 309 implementation, you may want to create a maven pom.xml at the root directory of your JRuby application containing the following :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.test</groupId>
    <artifactId>torquebox-media-example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Torquebox Media Demo Application</name>
    <url>http://www.mobicents.org/mss-pure-jruby-telco.html</url>

  <dependencies>
    <!-- media dependencies -->
    <dependency>
      <groupId>org.mobicents.external.jsr309</groupId>
      <artifactId>mscontrol</artifactId>
      <version>0.3</version>
    </dependency>

    <dependency>
      <groupId>org.mobicents.jsr309</groupId>
      <artifactId>mobicents-jsr309-impl</artifactId>
      <version>2.0.0.BETA2</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.mobicents.servlet.sip</groupId>
      <artifactId>sip-servlets-spec</artifactId>
      <version>1.1.11-SNAPSHOT</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
```

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>

        <outputDirectory>/opt/torquebox/builder/torquebox/torquebox-docs/en-US/lib/
        java</outputDirectory>
        <includeScope>runtime</includeScope>
      </configuration>
    </execution>
  </executions>
</plugin>
</plugins>
</build>

<repositories>
  <repository>
    <id>JbossRepository</id>
    <name>Jboss Repository</name>
    <url>http://repository.jboss.org/maven2</url>
    <snapshots>
      <enabled>>true</enabled>
    </snapshots>
    <releases>
      <enabled>>true</enabled>
    </releases>
  </repository>
</repositories>
</project>
```

and type the following command in a shell

```
mvn clean install
```

or copy the dependencies mentioned in the maven pom above from the jboss repository located at <http://repository.jboss.org/maven2> in the following directory of their JRuby application *lib/java*

The next step is to start using the JSR 309 API to control the Media Server and build Media applications, you can read the specification at this address : <http://jcp.org/en/jsr/detail?id=309>

To help you getting started with you can play with and checkout the code of our [sample application demonstrating Announcement and DTMF Detection](http://www.mobicients.org/mss-pure-jruby-telco.html) [http://www.mobicients.org/mss-pure-jruby-telco.html]



# Ruby-on-Rails Support

TorqueBox provides an enterprise-grade environment that not only provides complete Ruby-on-Rails compatibility, but also goes beyond the functionality offered in traditional Rails environments.

## 9.1. Ruby-on-Rails

Ruby-on-Rails (also referred to as "RoR" or "Rails") is one of the most popular Model-View-Controller (MVC) frameworks for the Ruby language. It was originally created by David Heinemeier Hansson at [37signals](http://37signals.com/) [http://37signals.com/] during the course of building many actual Ruby applications for their consulting business.

Rails has straight-forward components representing models, views, and controllers. The framework as a whole values convention over configuration. It has been described as "opinionated software" in that many decisions have been taken away from the end-user.

It is exactly the opinionated nature of Rails that allows it to be considered a simple and agile framework for quickly building web-based applications. Additionally, since Ruby is an interpreted language instead of compiled, the assets of an application can be edited quickly, with the results being immediately available. In most cases, the application does not need to be restarted to see changes in models, views or controllers reflected.

## 9.2. Rails on TorqueBox

Since Ruby-on-Rails ostensibly doesn't do anything new, but simply does it with a different language, TorqueBox uses the power of [JRuby](http://jruby.org/) [http://jruby.org/] to deploy regular Ruby-on-Rails apps within the JBoss AS appserver.

Your Rails application gets deployed alongside any other WAR or EAR, complete with the ability to run on a cluster. The full JBoss Web stack is utilized to provide enterprise-grade serving for Rails applications

## 9.3. Preparing your Rails application

While TorqueBox is 100% compatible with Ruby-on-Rails, there are a few steps that must be taken to ensure success.

### 9.3.1. Using the application template

You can use the included application template to setup a new Rails application or modify an existing one to work with TorqueBox.

#### 9.3.1.1. Creating a new Rails application

To create a new Rails application using the template, simply use the `-m` parameter when you execute the `rails` command.

```
$ rails -m $TORQUEBOX_HOME/share/rails/template.rb
```

This will perform the necessary setup to quickly get started with TorqueBox.

### 9.3.1.2. Applying template to an existing application

To apply the template to an existing application, simply use the `rails:template` rake task.

```
$ rake rails:template LOCATION=$TORQUEBOX_HOME/share/rails/template.rb
```

## 9.3.2. Manually configuring an application

### 9.3.2.1. Include the JDBC Gems for Database Connectivity

ActiveRecord applications deployed on TorqueBox benefit from using the Java-based JDBC database drivers. These drivers are provided as a handful of gems which you may include into your application through `config/environment.rb`.

JDBC gems for many popular databases is pre-installed with the TorqueBox server. You simply must reference the `activerecord-jdbc-adapter` from your `environment.rb` within the `Rails::Initializer.run` block.

```
Rails::Initializer.run do |config|  
  
  config.gem "activerecord-jdbc-adapter",  
            :lib=>'jdbc_adapter'  
  
end
```

All databases will require inclusion of the `activerecord-jdbc-adapter`. No other gems need to be required or loaded, since ActiveRecord will perform further discovery on its own. Database gems supporting Derby, H2, HSQLDB, MySQL, PostgreSQL, and SQLite3 are supplied in the TorqueBox binary distribution.

### 9.3.2.2. Include the TorqueBox Ruby packages

In order to gain access to the advanced features of TorqueBox, you must include the TorqueBox packages into your project. The TorqueBox gems are also included into your application through `config/environment.rb`.

```
Rails::Initializer.run do |config|  
  
  config.gem "torquebox-gem"  
  config.gem "torquebox-rails"  
  
end
```

### 9.3.2.2.1. torquebox-gem

The `torquebox-gem` provides access to advanced functionality, such as scheduled jobs and task queues.

### 9.3.2.2.2. torquebox-rails

The `torquebox-rails` provides additional `rake` tasks to your application. These tasks assist in deploying, undeploying, and executing the TorqueBox server. To make these tasks available to your project, you also need to add a `require` statement to the Rakefile at the top of your application.

```
require 'torquebox/tasks'
```

Invoking `rake -T` will show the tasks now available within your project.

```
rake torquebox:rails:check_frozen      # Check for that Rails has been
f...
rake torquebox:rails:deploy            # Deploy the Rails app
rake torquebox:rails:undeploy          # Undeploy the Rails app
rake torquebox:server:check            # Check your installation of the
...
rake torquebox:server:run              # Run TorqueBox server
```

### 9.3.2.3. Eliminate or replace "native" Gems

"Native" gems that rely upon machine-specific compiled code do not function with JRuby and TorqueBox. You must replaced these gems with pure-Ruby or pure-Java implementations. In the future, native gems using the FFI facilities will be usable.

## 9.4. Scheduled Jobs

Rails applications support *scheduled jobs* located under `$RAILS_ROOT/app/jobs/`.

## 9.5. Task Queues

Rails applications support *task queues* located under `$RAILS_ROOT/app/queues/`.

## 9.6. Deploying

The TorqueBox Application Server is capable of serving many applications simultaneously. To add your application to the server, you must *deploy* it.

### 9.6.1. Deploy using the Rake tasks

The TorqueBox-Rails support package includes Rake tasks to deploy and undeploy your application from an instance of the TorqueBox Server.

First, the variable `$JBOSS_HOME` must be set to the path of the top of your JBoss installation as described in [Chapter 3, Installation](#)

```
$ export JBOSS_HOME=/path/to/torquebox/jboss
```

If you're using any configuration other than `default`, you must also set `$JBOSS_CONF`.

```
$ export JBOSS_CONF=web
```

Once these variables are set, you may perform a default deployment using the `jboss:rails:deploy` task.

```
$ rake torquebox:rails:deploy
```

To undeploy your application, the `jboss:rails:undeploy` task is available

```
$ rake torquebox:rails:undeploy
```

The TorqueBox Server does not need to be running for these commands to work.

By default, these tasks deploy your application to root of your TorqueBox Server's web space, without any virtual host configuration. To access the application once deployed, you should use your browser to access `http://localhost:8080/`.

When the application is deployed, a deployment descriptor is written to the `$JBOSS_HOME/server/$JBOSS_CONF/deploy/` directory with a filename based upon the directory name of your `$RAILS_ROOT`.

For instance, if your application was deployed from `/Users/bob/myapp/`, the deployment descriptor would be named `myapp-rails.yml`.

Rewriting or simply updating the last-modified time (using a command such as `touch`) of this descriptor will cause the TorqueBox server to redeploy the application. The `torquebox:rails:deploy` task simply emits this file.

Removing the descriptor will cause the TorqueBox server to undeploy the application. This is what the `torquebox:rails:undeploy` task does.

### 9.6.1.1. Deploying a non-root context

By default, the `torquebox:rails:deploy` task will attach your application to the root context. If you would rather deploy to a non-root context, you may provide it as an argument to the task invocation.

```
$ rake torquebox:rails:deploy['/my-application']
```

The root of your application would then be accessible at `http://localhost:8080/my-application`.

## 9.6.2. Deploy using a descriptor

To customize some of the aspects of deployment, instead of using the Rake tasks, you may manually create a *deployment descriptor*. A deployment descriptor is a small text file that is placed in the `deploy/` directory of the server in order to have the application deployed.

### 9.6.2.1. Location & Naming

The deployment descriptor needs to be placed within the `deploy/` directory of the AS configuration in use. If you are using the default configuration, the path would be:

```
$JBOSS_HOME/server/default/deploy/
```

The descriptor is a YAML file, and must end with the suffix of `-rails.yml`. The prefix is arbitrary, but is usually some form of your application's name.

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

### 9.6.2.2. Contents of the descriptor

The descriptor has 2 main sections:

1. General application configuration
2. Web-specific configuration
3. Sip-specific configuration

#### 9.6.2.2.1. General Application Configuration

The application section describes the `RAILS_ROOT` and `RAILS_ENV` for the deployed application. Under traditional (mongrel, lighttpd) deployments, this information is picked up through the current working directory or environment variables. Since the TorqueBox Server runs from a different location, the current working directory has no meaning. Likewise, as multiple applications may be deployed within a single TorqueBox Server, a single global environment variable to set `$RAILS_ENV` is nonsensical.

#### Example 9.1. Application configuration in \*-rails.yml

```
application:  
  RAILS_ROOT: /path/to/myapp  
  RAILS_ENV:  development
```

### 9.6.2.2.2. Web-specific configuration

Traditional Rails applications are deployed individually, without respect to hostnames or context-path. Running under TorqueBox, you may host several apps under a single host, or multiple apps under different hostnames.

Both the virtual-host and context-path configuration are nested under the *web* section.

#### 9.6.2.2.2.1. Virtual Hosts

Virtual hosts allow one application to respond to *www.host-one.com*, while another running within the same JBoss AS to respond to *www.host-two.com*. If no host is specified, then the application will respond to all requests directed at the TorqueBox Server.

#### Example 9.2. Virtual host configuration in \*-rails.yml

```
web:
  host: www.host-one.com
```

#### 9.6.2.2.2.2. Context paths

In addition to virtual hosts, applications within a single TorqueBox Server may be separated purely by a *context path*. For a given host, the context path is the prefix used to access the application. Traditional Rails apps respond from the top of a site. By using a context path, you can mount applications at a location under the root.

For example, <http://www.host-one.com/app-one/> could point to one application, while <http://www.host-one.com/app-two/> could point to another separate application.

#### Example 9.3. Context path configuration in \*-rails.yml

```
web:
  context: /app-one
```

The context path and virtual host configurations can be used at the same time, if desired.

#### Example 9.4. Virtual host with context path configuration in \*-rails.ymls

```
web:
  host: www.mycorp.com
  context: /app-one
```

### 9.6.2.2.3. SIP-specific configuration

The sip configuration section allows you to define the appname (application name) of the SIP Servlets application (mandatory) and the name of the class that will handle the SIP messages :

### Example 9.5. SIP configuration in \*-rails.yml

```

sip:
  rubycontroller: SipHandler

```

### 9.6.2.3. Complete \*-rails.yml Deployment Descriptor Example

```

application:
  RAILS_ROOT: /path/to/myapp
  RAILS_ENV: development
web:
  host: www.mycorp.com
  context: /app-one
sip:
  rubycontroller: SipHandler

```

## 9.6.3. Deployment using a partial descriptor

While the *myapp-rails.yml* descriptor may contain all the abovementioned information, you may also deploy using a combination of descriptor and additional files in your *config/* directory. The bare minimum required in the descriptor is the `RAILS_ROOT` setting for the `application:` block.

The primary purpose is to allow an application to be fully self-contained, specifying all vital production information within the same codebase as the rest of the application. This is useful for Capistrano-based deployments.

### 9.6.3.1. *config/web.yml*

The entire `web:` section of *myapp-rails.yml* may be supplied from within the application's own *config/web.yml* file, if present. If both *myapp-rails.yml* and *config/web.yml* exist, the *myapp-rails.yml* configuration values take precedent.

This allows for typical values to be placed in *config/web.yml* for production deployment, while allowing developers to override them through *myapp-rails.yml* during development deployments.

### Example 9.6. *config/web.yml*

```

host: torquebox.org
context: /

```

### 9.6.3.2. *config/rails-env.yml*

If present in your application, *config/rails-env.yml* can specify the default value for `RAILS_ENV`. A value specified in *myapp-rails.yml* deployment descriptor may still override the value provided by the application's own *config/rails-env.yml*.

### Example 9.7. Example `config/rails-env.yml`

```
RAILS_ENV: production
```

Similar to `config/web.yml`, usage of `config/rails-env.yml` can simplify production deployment by providing the correct value from within the app.

## 9.6.4. Deployment using a bundle

Rails applications may be deployed as atomic *bundles*. A bundle is simply an archive of the application's directory. The TorqueBox server deploys bundles created with the Java `jar` tool. Rake tasks are provided to assist with the creation and deployment of bundles.

### 9.6.4.1. Creating a bundle

The `torquebox:rails:bundle` rake task may be used to create a bundle of the application. The task invokes the Java `jar` commandline tool to bundle up the project directory, *excluding* `tmp/` and `log/` directories.

```
$ rake torquebox:rails:bundle
```

The resulting bundle will be placed at the root of the application, as a file named `myapp.rails`. To inspect the contents, you may use the `jar` tool.

```
$ jar tf
myapp.rails
META-INF/
META-INF/MANIFEST.MF
app/
app/controllers/
app/controllers/application_controller.rb
...
```

### 9.6.4.2. Deploying a bundle

To deploy a bundle, simply copy it to the `deploy/` directory of the server. The `torquebox:rails:deploy:bundle` rake task may be used to both create and deploy a bundle.

```
$ rake torquebox:rails:deploy:bundle
```

If you wish to deploy manually, a command similar to the following may be used

```
$ rake torquebox:rails:bundle
$ cp myapp.rails $JBOSS_HOME/server/default/deploy/
```

If you redeploy a bundle, the server will remove the previous version, and hot-redeploy the bundle just copied.

## 9.7. Control thread concurrency using `pooling.yml`

Since the TorqueBox platform supports Rails 2.2+, which are threadsafe, the default mode of operation is to share a single Ruby runtime across threads.

If your application is not designed to be threadsafe, you can instead pool the runtimes allowing a single-threaded model. Typically, if your application creates and uses global variables to manage state for a single request, you may have problems with the default multithreaded behavior.

To enable pooling-mode instead of shared multithreading you need to add a YAML file at `$RAILS_ROOT/config/pooling.yml`. This file is optional, and only required if you wish to enable pooling.

The `pooling.yml` file contains a section for each configurable pool of runtimes.

### 9.7.1. Pooling for web requests

A section named `web` defines the pooling configuration for all regular web requests. This does not include SOAP endpoints. SIP requests do participate in the web pool, though.

Web pooling is defined by creating a section named `web`, and specifying `min` and `max` parameters for the pool. The minimum number of runtimes will be created before your application starts. As the available items in the pool are exhausted, new ones will be asynchronously created, up to the maximum specified.

Currently there is no reaping performed on the pool reduce its size.

#### Example 9.8. Example `config/pooling.yml`

```
web:
  min: 2
  max: 30
```

## 9.8. Integrate additional JEE components

The `$RAILS_ROOT/config/` directory may also hold other JEE and JBoss deployment descriptors. It is conceptually equivalent to `META-INF/` or `WEB-INF/` in JARs and WARs.

For instance, it may contain `mydb-ds.xml` to deploy a datasource, `web.xml` to configure additional Servlets and Filters, or a `jboss-web.xml` to manage the enterprise naming context.

## 9.9. Controlling the TorqueBox Server

### 9.9.1. Using Rake

To start the TorqueBox server from within your Rails application, the Rake task `torquebox:server:run` is provided.

```
$ rake torquebox:server:run
TorqueBox Server OK:
/Users/bob/oddthesis/jboss/jdk1.6/current/server/default
=====
JBoss Bootstrap Environment
```

To stop the server, simply interrupt the terminal using *control-c*. While the server is running, applications may be repeatedly deployed and undeployed.

# Capistrano Support

## 10.1. Capistrano

Capistrano is a deployment tool to assist in moving code from a repository to a production server.

## 10.2. App-Server separate from App

The TorqueBox server in production is a separate entity from the applications being deployed upon it. When deploying with mongrel or other traditional Ruby solutions, part of the deployment normally includes restarting the server. TorqueBox does not need to be restarted nearly as often.

Generally the TorqueBox server is installed in a system-wide location since it may support multiple applications simultaneously. For install, you may simply unpackage and install it under `/opt/torquebox-server`.

The TorqueBox server can be managed through a normal `init.d/` script, or through Daniel Bernstein's `daemontools`. The Capistrano support provided by TorqueBox can work with either process for managing the server, when necessary.

Either way, two settings in your application's `deploy.rb` are required to be able to deploy to a remote TorqueBox server.

```
set :jboss_home,      "/opt/torquebox-server/jboss"  
set :jboss_config,   :default
```

These should point to the same location as `$JBOSS_HOME` on the server. The `:jboss_config` matches the `$JBOSS_CONF`, and is only required if a configuration other than default is desired.



### Note

Capistrano support is still evolving and will be fully documented once it settles.



# Rack Support

## 11.1. General `config.ru` Support

TorqueBox current supports general `config.ru`-based applications. In your application's directory, your Rack application can be booted from a `config.ru` (or other filename) you provide. The Ruby runtime provided to your application is quite rudimentary. If you desire to use RubyGems or other libraries, it is up to you to require the necessary files (for instance, `require 'rubygems'`).

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

## 11.2. Rack deployment descriptor (`*-rack.yml`)

To customize some of the aspects of deployment, instead of using the Rake tasks, you may manually create a *deployment descriptor*. A deployment descriptor is a small text file that is placed in the `deploy/` directory of the server in order to have the application deployed.

### 11.2.1. Location & Naming

The deployment descriptor needs to be placed within the `deploy/` directory of the AS configuration in use. If you are using the default configuration, the path would be:

```
$JBOSS_HOME/server/default/deploy/
```

The descriptor is a YAML file, and must end with the suffix of `-rack.yml`. The prefix is arbitrary, but is usually some form of your application's name.

```
$JBOSS_HOME/server/default/deploy/myapp-rack.yml
```

### 11.2.2. Contents of the descriptor

The descriptor has 2 main sections:

1. General application configuration
2. Web-specific configuration

### 11.2.2.1. General Application Configuration

The application section describes the `RACK_ROOT` and `RACK_ENV` for the deployed application. Under traditional (mongrel, lighttpd) deployments, this information is picked up through the current working directory or environment variables. Since the TorqueBox Server runs from a different location, the current working directory has no meaning. Likewise, as multiple applications may be deployed within a single TorqueBox Server, a single global environment variable to set `$RACK_ENV` is nonsensical.

#### Example 11.1. Application configuration in `*-rack.yml`

```
application:  
  RACK_ROOT: /path/to/myapp  
  RACK_ENV:  development
```

### 11.2.2.2. Web-specific configuration

Traditional Rails applications are deployed individually, without respect to hostnames or context-path. Running under TorqueBox, you may host several apps under a single host, or multiple apps under different hostnames.

Both the virtual-host and context-path configuration are nested under the `web` section.

#### 11.2.2.2.1. Virtual Hosts

Virtual hosts allow one application to respond to `www.host-one.com`, while another running within the same JBoss AS to respond to `www.host-two.com`. If no host is specified, then the application will respond to all requests directed at the TorqueBox Server.

#### Example 11.2. Virtual host configuration in `*-rack.yml`

```
web:  
  host: www.host-one.com
```

#### 11.2.2.2.2. Context paths

In addition to virtual hosts, applications within a single TorqueBox Server may be separated purely by a `context path`. For a given host, the context path is the prefix used to access the application. Traditional Rails apps respond from the top of a site. By using a context path, you can mount applications at a location under the root.

For example, `http://www.host-one.com/app-one/` could point to one application, while `http://www.host-one.com/app-two/` could point to another separate application.

#### Example 11.3. Context path configuration in `*-rack.yml`

```
web:
```

```
context: /app-one
```

The context path and virtual host configurations can be used at the same time, if desired.

#### **Example 11.4. Virtual host with context path configuration in \*-rack.ymls**

```
web:  
  host: www.mycorp.com  
  context: /app-one
```



---

# Appendix A. GNU Lesser General Public License version 3

Version 3, 29 June 2007

Copyright 2009 JBoss Middleware, a division of Red Hat, Inc. <http://jboss.com/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

## 0. Additional Definitions.

As used herein, this License refers to version 3 of the GNU Lesser General Public License, and the GNU GPL refers to version 3 of the GNU General Public License.

The Library refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An Application is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A Combined Work is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the Linked Version.

The Minimal Corresponding Source for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The Corresponding Application Code for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

## 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

## 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a. under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b. under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

## 3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a. Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b. Accompany the object code with a copy of the GNU GPL and this license document.

## 4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a. Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b. Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c. For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d. Do one of the following:

1. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
2. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the users computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e. Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

## 5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b. Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

## 6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License or any later version applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you

may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

---

## Appendix B. MIT License

Copyright 2009 JBoss Middleware, a division of Red Hat, Inc. <http://jboss.com/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

