TorqueBox

Toby Crawley

Charlotte.rb

May 2011

whoami

- @tcrawley
- C > Java > PHP > Java > Ruby > Java?
- Red Hat Senior Engineer
- member of project:odd

project:odd

Goal

To convert you all to TorqueBox users!

TorqueBox

the power of **JBoss** with the expressiveness of **Ruby**

TorqueBox: what?

- A “real” Application Server for Ruby
- 100% open-source, LGPL license
- Based on JBoss AS and JRuby
- Recently released 1.0.0!

Yes, it's Java

@avalanche123
Bulat Shakirzyanov

"Java is a DSL for taking large XML files and converting them to stack traces" *

23 Nov via Twitter for Android ☆ Favorite ↻ Retweet ↩ Reply

Retweeted by fakeEvanMiller and 100+ others

* Quote by Scott Bellware

I promise...

- No XML
- No Java *
- No war files *
- Only Ruby and YAML

* Unless you really want to

The Competition

Unicorn, Thin, Passenger, Trinidad, Warbler...

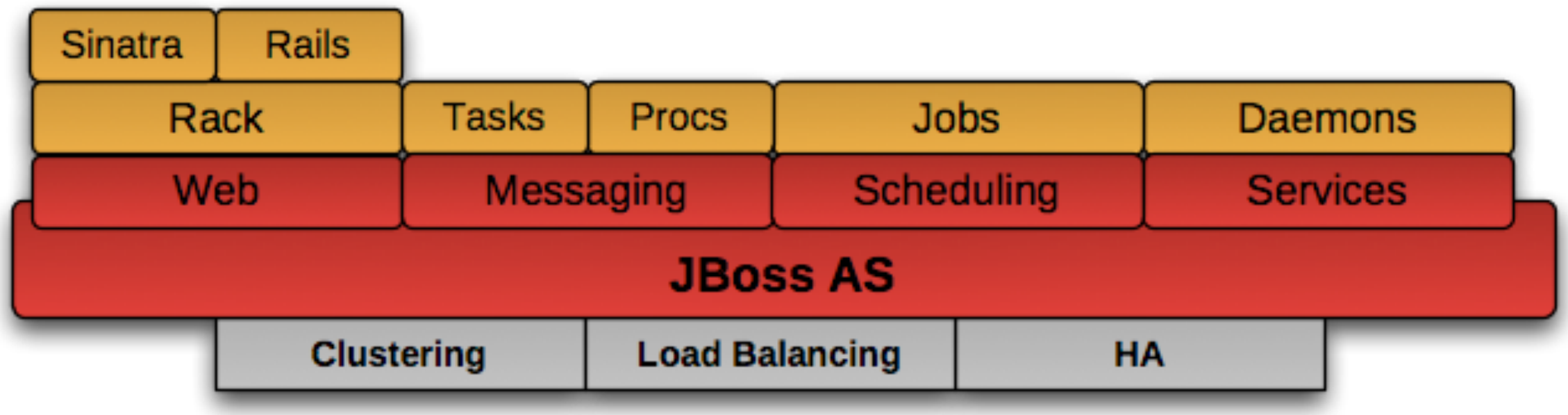
...all address only the web question.

TorqueBox: why?

- “Native” support for Rack apps
- Built-in:
  - background processing
  - scheduling
  - daemons (services)
  - clustering
- Easily scalable

JBoss AS

the good parts


Sinatra
Rails
Rack
Tasks
Procs
Jobs
Daemons
Web
Messaging
Scheduling
Services
JBoss AS
Clustering
Load Balancing
HA

AS = Application Server

- Not just “web server + interpreter”
- More like initd than httpd

JRuby

a good idea done well

JRuby: why?

- Very fast runtime
- Real threads
- Java libraries
- Java tools
- Healthy community

Setting Up TorqueBox

in a few easy steps!

Easy Install

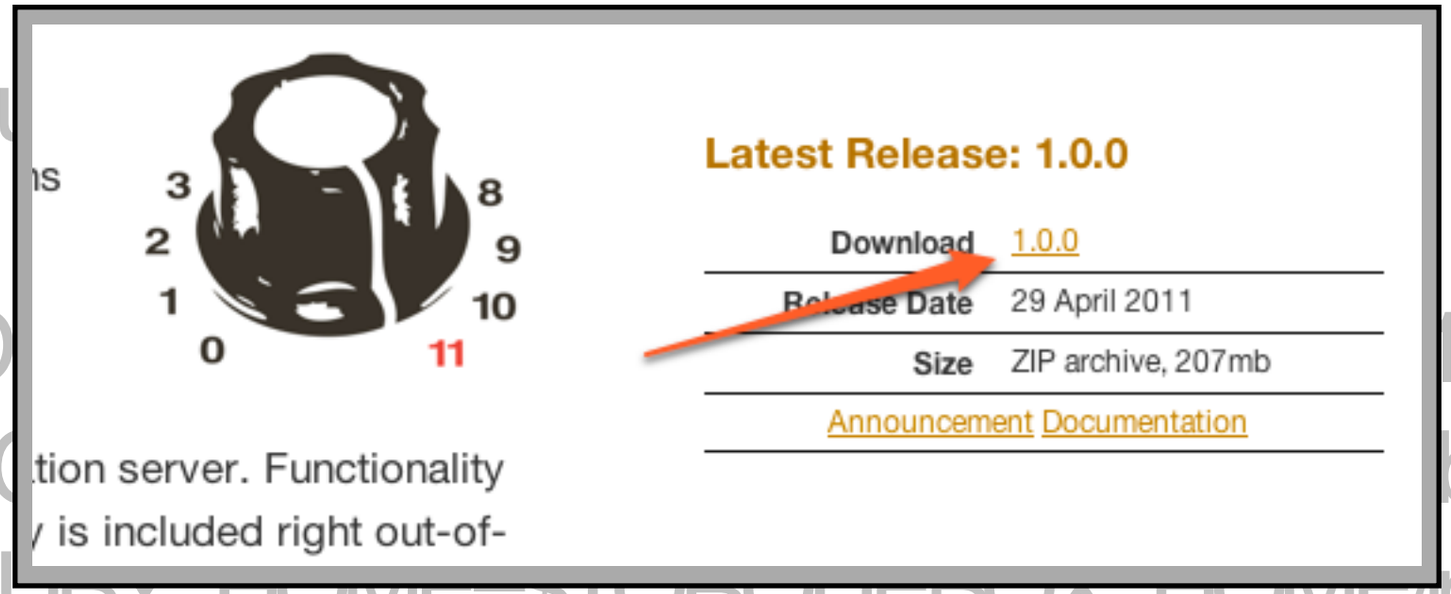
(download 1.0.0 from torquebox.org)

```
$ unzip torquebox-dist-1.0.0-bin.zip

$ export TORQUEBOX_HOME=$PWD/torquebox-1.0.0
$ export JBOSS_HOME=$TORQUEBOX_HOME/jboss
$ export JRUBY_HOME=$TORQUEBOX_HOME/jruby

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

Easy Install

(download 1.0.0 from torquebox.org)

```
$ unzip torqu
```




```
$ export TO                                  1.0.0
$ export JB                                  boss
$ export JRUBY_HOME=$TORQUEBOX_HOME/jruby

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

Easy Install

(download 1.0.0 from torquebox.org)

```
$ unzip torquebox-dist-1.0.0-bin.zip

$ export TORQUEBOX_HOME=$PWD/torquebox-1.0.0
$ export JBOSS_HOME=$TORQUEBOX_HOME/jboss
$ export JRUBY_HOME=$TORQUEBOX_HOME/jruby

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

Easy Install

(download 1.0.0 from torquebox.org)

```
$ unzip torquebox-dist-1.0.0-bin.zip

$ export TORQUEBOX_HOME=$PWD/torquebox-1.0.0
$ export JBOSS_HOME=$TORQUEBOX_HOME/jboss
$ export JRUBY_HOME=$TORQUEBOX_HOME/jruby

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

Easy Install

(download 1.0.0 from torquebox.org)

```
$ unzip torquebox-dist-1.0.0.CR1-b
```

```
$ export TORQUEBOX_HOME=
$ export JBOSS_HOME=$TORQ
$ export JRUBY_HOME=$TORQUEBOX_HOME/jruby
```

Make sure the jruby found in your path is in $JRUBY_HOME/bin.

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

Rake Tasks

Rakefile

```
require "torquebox-rake-support"
```

Database Connectivity

Gemfile

```
gem "activerecord-jdbc-adapter"

gem "jdbc-postgres"
# gem "jdbc-sqlite3"
# gem "jdbc-mysql"
```

Rails Template

- Adds TorqueBox rake tasks
- Adds the JDBC sqlite3 gems
- Adds TorqueBox session_store
- Adds Backgroundable module

Rake Tasks

rake **torquebox:run**
Run TorqueBox server

rake **torquebox:deploy**[context_path]
Deploy the app in the current directory

rake **torquebox:undeploy**
Undeploy the app in the current directory

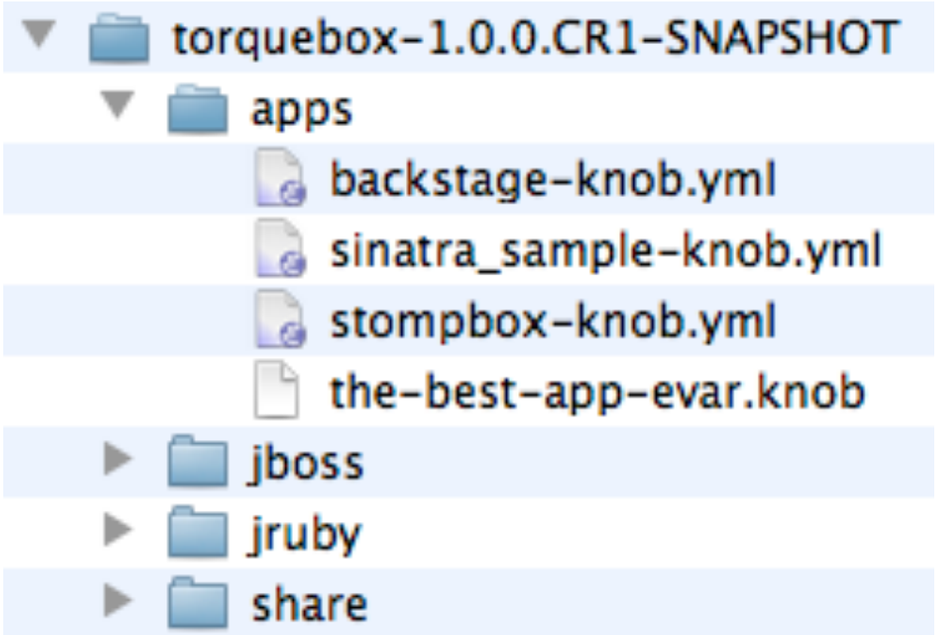
Deployment Descriptors

torquebox:deploy creates a *deployment descriptor* in the **$TORQUEBOX_HOME/apps/** directory

Hot Deployment

$TORQUEBOX_HOME/apps/

- anything added to apps/ will get deployed
- anything removed from apps/ will get undeployed
- anything updated in apps/ will get redeployed
- TorqueBox deployers make JBoss grok YAML



Hot Deployment

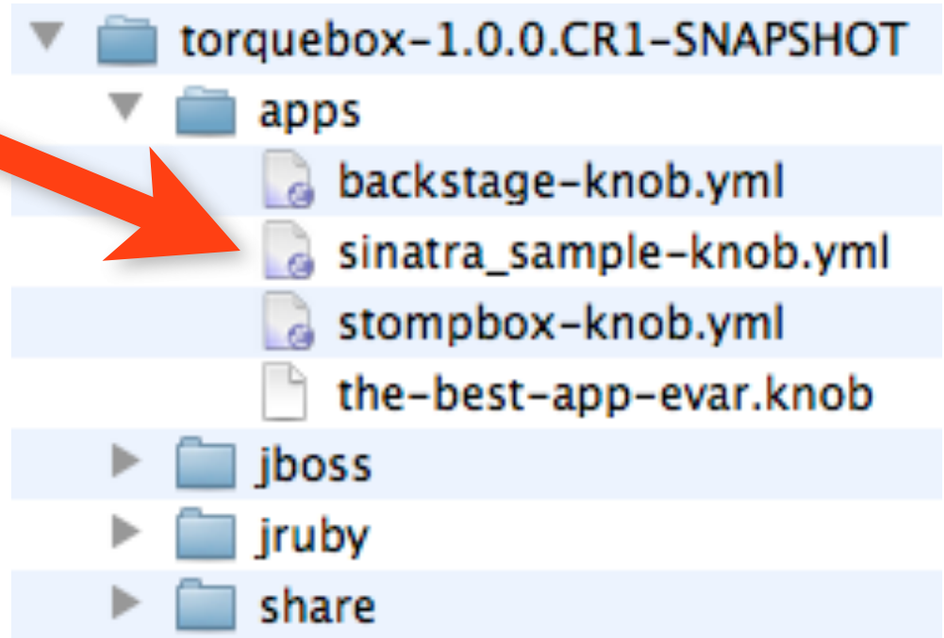
X_HOME/apps/

deployment descriptors

to apps/

d

- anything removed from apps/ will get undeployed
- anything updated in apps/ will get redeployed
- TorqueBox deployers make JBoss grok YAML

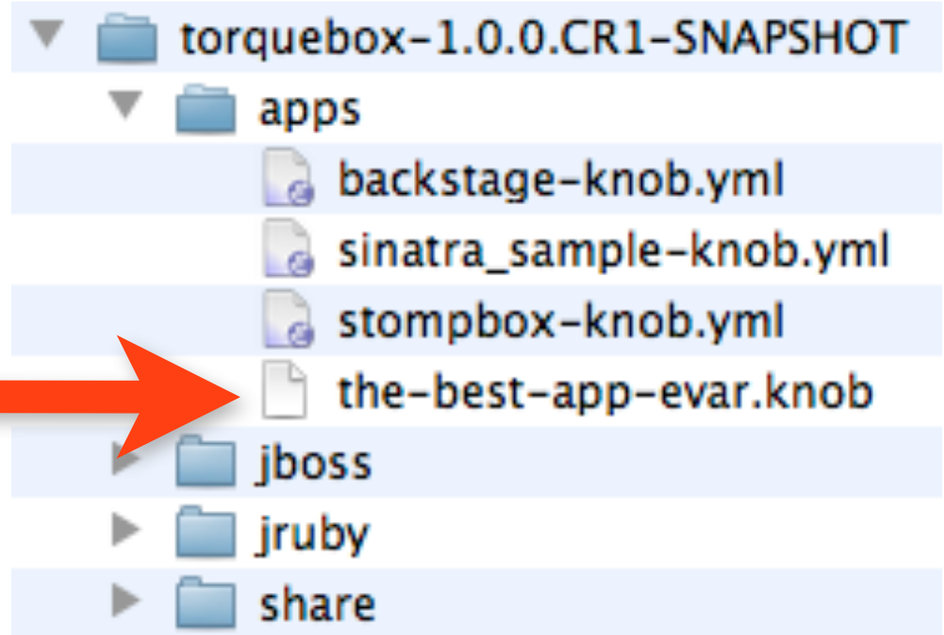


Hot Deployment

$TORQUEBOX_HOME/apps/

- anything added to apps/ will get deployed
- anything removed from apps/ will get undeployed
- TorqueBox deployers make JBoss grok YAML

knob files (zip archives)



Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:     /path/to/myapp
  env:      development
web:
  context: myapp
  host:    www.yourhost.com
  static:  public
environment:
  MAIL_HOST: mail.yourhost.com
  REPLY_TO:  you@yourhost.com
```

Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:       /path/to/myapp
  env:        developmen
web:
  context: myapp
  host:       www.yourhost.
  static:     public
environment:
  MAIL_HOST: mail.yourho
  REPLY_TO:  you@yourhos
```

The fully-qualified path to the app. This will be the value of either RAILS_ROOT or RACK_ROOT

Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:     /path/to/myapp
  env:      development
web:
  context: myapp
  host:     www.yourhost.
  static:  public
environment:
  MAIL_HOST: mail.yourho
  REPLY_TO:  you@yourhos
```

The runtime mode of the app. This will be either RAILS_ENV or RACK_ENV

Deployment Descriptors

apps/myapp-kn

```
application:
  root:     /pat
  env:      deve
web:
  context: myapp
  host:    www.yourhost.com
  static:  public
environment:
  MAIL_HOST: mail.yourhost.com
  REPLY_TO:  you@yourhost.com
```

The app's ***context path***
(or "sub URI"):
http://localhost:8080/myapp
Can be set via rake:
rake torquebox:deploy[myapp]
The default is root:
http://localhost:8080/

Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:     /path/to/myapp
  env:      development
web:
  context: myapp
  host:     www.yourhost.com
  static:   public
environment:
  MAIL_HOST: mail.yourhost.com
  REPLY_TO:  you@yourhost.com
```

A list of virtual hostnames to which to bind the app.

Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:     /path/to/myapp
  env:      development
web:
  context: myapp
  host:     www.yourhost.com
  static:   public
environment:
  MAIL_HOST: mail.yourhost.com
  REPLY_TO:  you@yourhost.com
```

The location of the app's static content, either absolute or relative to the app's root.

Deployment Descriptors

apps/myapp-knob.yml

```
application:
  root:       /path/to/myapp
  env:        development
web:
  context: myapp
  host:       www.yourhost.com
  static:     public
environment:
  MAIL_HOST: mail.yourhost.com
  REPLY_TO:  you@yourhost.com
```

Any environment variables required by the app.

Deployment Descriptors

- **config/torquebox.yml**
- ***internal*** descriptors have the same structure as the ***external*** ones in **apps/**
- may be used to provide your own reasonable defaults

Components

Put 'em together, and you have an AS

Web

make rack, not war

Web

- All rack-based frameworks supported: rails, sinatra, etc
- No packaging required: apps deploy from where they sit on disk
- No redeploy necessary to see changes when using rack reloading or rails development mode

Scheduling

get regular later

Jobs

app/jobs/newsletter_sender.rb

```
class NewsletterSender

  def run()
    subscriptions = Subscription.find(:all)
    subscriptions.each do lel
      send_newsletter( e )
    end
  end

end
```

Jobs

config/torquebox.yml

```
jobs:
  monthly_newsletter:
    description: first of month
    job: NewsletterSender
    cron: '0 0 0 1 * ?'

  process_tps_reports:
    job: TPSReportProcessor
    cron: '0 0 0 0 MON ?'
```

Jobs

- More portable. What is the first day of the week on BSD again? What's cron on Windows?
- Self contained within the app. No external systems to manage and keep in sync.
- Full application environment loaded and available.

Messaging

asynchronicity

Background Processing

- Tasks
- Backgroundable methods

Tasks

app/tasks/email_task.rb

```
class EmailTask < TorqueBox::Messaging::Task
  def welcome(payload)
    person = Person.find_by_id(payload[:id])
    person.send_welcome_spam if person
  end
end
```

Tasks

app/tasks/email_task.rb

```
class EmailTask < TorqueBox::Messaging::Task
  def welcome(payload)
    person = Person.find_by_id(payload[:id])
    person.send_welcome_spam if person
  end
end
```

Tasks

app/tasks/email_task.rb

```
class EmailTask < TorqueBox::Messaging::Task
 def welcome(payload)
   person = Person.find_by_id(payload[:id])
   person.send_welcome_spam if person
  end
end
```

Tasks

app/tasks/email_task.rb

```
class EmailTask < TorqueBox::Messaging::Task
  def welcome(payload)
    person = Person.find_by_id(payload[:id])
    person.send_welcome_spam if person
  end
end
```

Tasks

app/controllers/people_controller.rb

```
class PeopleController < ApplicationController
  def create
    @person = Person.new(params[:person])
    respond_to do |format|
      if @person.save
        EmailTask.async(:welcome, :id => person.id)
        # respond appropriately
      end
    end
  end
end
```

Tasks

app/controllers/people_controller.rb

```
class PeopleController < ApplicationController
  def create
    @person = Person.new(params[:person])
    respond_to do |format|
      if @person.save
        EmailTask.async(:welcome, :id => person.id)
        # respond appropriately
      end
    end
  end
end
```

Backgroundable

Inspired by DelayedJob's handle_asynchronously, it's trivial to create implicit background Tasks.

Backgroundable

lib/something.rb

```
include TorqueBox::Messaging

class Something
  include Backgroundable
  always_background :foo
  def foo; end
  def bar; end
end
...
@something.foo
@something.background.bar
```

Backgroundable

lib/something.rb

```
include TorqueBox::Messaging

class Something
  include Backgroundable
  always_background :foo
  def foo; end
  def bar; end
end
...
@something.foo
@something.background.bar
```

Backgroundable

lib/something.rb

```
include TorqueBox::Messaging

class Something
  include Backgroundable
  always_background :foo
  def foo; end
  def bar; end
end
...
@something.foo
@something.background.bar
```

Backgroundable

lib/something.rb

```
include TorqueBox::Messaging

class Something
  include Backgroundable
  always_background :foo
  def foo; end
  def bar; end
end
...
@something.foo
@something.background.bar
```

Background Processing

Call it from your **controllers**, **models**, and **observers**, or even other **tasks**. Even in non-Rails apps!

Background Processing

- No extra tables in your database
- No external system to manage
- Little to no config required at all
- System gets redeployed w/app
- Efficient loading of rails environment
- Automatic load balancing and retries
- Works on Windows, if you care

TorqueBox::Messaging

- JMS (Java Message Service) is an API for messaging
- HornetQ is the JBoss JMS implementation

Queues

Tasks and Backgroundable are built on top of Queues. Of course, you may build your own messaging based apps by defining your own **Queues**, **Topics**, and their message **Processors** yourself.

Queues

config/torquebox.yml

```
queues:
  /queues/questions:

  /queues/answers:
    durable: false
```

Topics

- behavior is different, but interface is the same.
- all subscribers of a **topic** see each message, but only one subscriber will see any message from a **queue**
- use topics: section of torquebox.yml to define topics

Processors

You can create a processor class to receive messages from a Topic or Queue

Processors

app/models/print_handler.rb

```
include TorqueBox::Messaging

class PrintHandler < MessageProcessor
  def initialize(opts)
    @printer = opts['printer'] || default
  end
  def on_message(body)
    puts "Processing #{body} of #{message}"
  end
end
```

Processors

config/torquebox.yml

```
messaging:
  /topics/orders:
    - PrintHandler
    - ShoutHandler
  /queues/receipts:
    PrintHandler:
      concurrency: 5
      config:
        printer: the_little_one
```

Processors

config/torquebox.yml

```yaml
messaging:
  /topics/orders:
    - PrintHandler
    - ShoutHandler
  /queues/receipts:
    PrintHandler:
      concurrency: 5
      config:
        printer: the_little_one
```

Processors

config/torquebox.yml

```yaml
messaging:
  /topics/orders:
    - PrintHandler
    - ShoutHandler
  /queues/receipts:
    PrintHandler:
      concurrency: 5
      config:
        printer: the_little_one
```

Processors

config/torquebox.yml

```yaml
messaging:
  /topics/orders:
    - PrintHandler
    - ShoutHandler
  /queues/receipts:
    PrintHandler:
      concurrency: 5
      config:
        printer: the_little_one
```

Processors

app/models/print_handler.rb

```
include TorqueBox::Messaging

class PrintHandler < MessageProcessor
  def initialize(opts)
    @printer = opts['printer'] || default
  end
  def on_message(body)
    puts "Processing #{body} of #{message}"
  end
end
```

Queues (again)

But how do you send a message?

Queues

contrived example

```
questions = Queue.new('/queues/questions')
answers = Queue.new('/queues/answers')

Thread.new do
  questions.publish "What time is it?"
  puts answers.receive( :timeout => 1000 )
end

puts questions.receive
answers.publish Time.now
```

Queues

contrived example

```
questions = Queue.new('/queues/questions')
answers = Queue.new('/queues/answers')

Thread.new do
  questions.publish "What time is it?"
  puts answers.receive( :timeout => 1000 )
end

puts questions.receive
answers.publish Time.now
```

Queues

contrived example

```
questions = Queue.new('/queues/questions')
answers = Queue.new('/queues/answers')

Thread.new do
  questions.publish "What time is it?"
  puts answers.receive( :timeout => 1000 )
end

puts questions.receive
answers.publish Time.now
```

Queues

contrived example

```
questions = Queue.new('/queues/questions')
answers = Queue.new('/queues/answers')

Thread.new do
  questions.publish "What time is it?"
  puts answers.receive( :timeout => 1000 )
end
```

puts questions.receive
answers.publish Time.now

Services

run along, lil’ daemon

Services

Long-running, non-web “daemons” that share the runtime environment and deployment lifecycle of your app.

Services

- Represented as a class with optional initialize(Hash), start() and stop() methods, which should each return quickly.
- Typically will start a long-running loop in a thread and respond to external events.
- Configured via services: section in torquebox.yml

Services

config/torquebox.yml

```
services:
  TimeMachine:
    queue: /queue/morris_day

  MyMudServer:

  SomeOtherService:
```

Services

app/services/time_machine.rb

```
class TimeMachine
  def initialize(opts)
    @queue = Queue.new(opts['queue'])
  end

  def start
    Thread.new do
      until @done
        @queue.publish(Time.now)
        sleep(1)
      end
    end
  end

  def stop; @done = true; end
end
```

Services

app/services/time_machine.rb

```
class TimeMachine
  def initialize(opts)
    @queue = Queue.new(opts['queue'])
  end

  def start
    Thread.new do
      until @done
        @queue.publish(Time.now)
        sleep(1)
      end
    end
  end

  def stop; @done = true; end
end
```

Services

app/services/time_machine.rb

```
class TimeMachine
  def initialize(opts)
    @queue = Queue.new(opts['queue'])
  end

  def start
    Thread.new do
      until @done
        @queue.publish(Time.now)
        sleep(1)
      end
    end
  end

  def stop; @done = true; end
end
```

Services

app/services/time_machine.rb

```
class TimeMachine
  def initialize(opts)
    @queue = Queue.new(opts['queue'])
  end

  def start
    Thread.new do
      until @done
        @queue.publish(Time.now)
        sleep(1)
      end
    end
  end

  def stop; @done = true; end
end
```

Caching

save a little for later

Caching

config/application.rb

```
config.cache_store =
:torque_box_store, :mode => :local

# or

config.cache_store =
ActiveSupport::Cache::TorqueBoxStore.new( :mode
=> :local )
```

Runtime Options

shorts or sweats?

Runtime Options

config/torquebox.yml

```
#per app!
ruby:
  version: 1.9
  compile_mode: jit
```

Clustering

less failure faster

Web

- session replication
- *intelligent* load-balancing (via mod_cluster)
- failover (via mod_cluster)

Messaging

HornetQ clusters automatically, giving you message processing capability that grows with the cluster.

Services

A service runs on every cluster node, unless marked as a singleton.

Jobs

A job runs on every cluster node, unless marked as a singleton (just like services).

Caching

Infinispan clusters automatically, "distributing" your cache.

Other Cool Stuff

BackStage

Dashboard to inspect and control Ruby components.

And a RESTful API.

TorqueBox::Backstage

Apps | Queues | Topics | Msg. Processors | Jobs | Services

Name	App	Status	Messages	Delivering
ExpiryQueue	n/a	Running	0	0
DLQ	n/a	Running	0	0
/queues/a-kitchen-sink-queue	n/a	Running	7	0
MessageProducerTask	kitchen-sink	Paused	0	0
Backgroundable	kitchen-sink	Running	0	0

StompBox

Easy **Heroku**-esque
git-based deployments.

StompBox

f | Weekend Edition | Stomp Box | git torquebox/stompbox – GitH | TorqueBox :: Full > #442 (1 | git gist: 835658 – GitHub

localhost:8080

Stomp Box :: Dashboard

Repositories Managed

ballast-sinatra
master →

ballast-sinatra
test →

buyappalachian.org
torquebox →

Pushes Received

	Date	Status	Commit	Repo
Push	February 09 - 16:13	received deploy	2dee90f	buyap

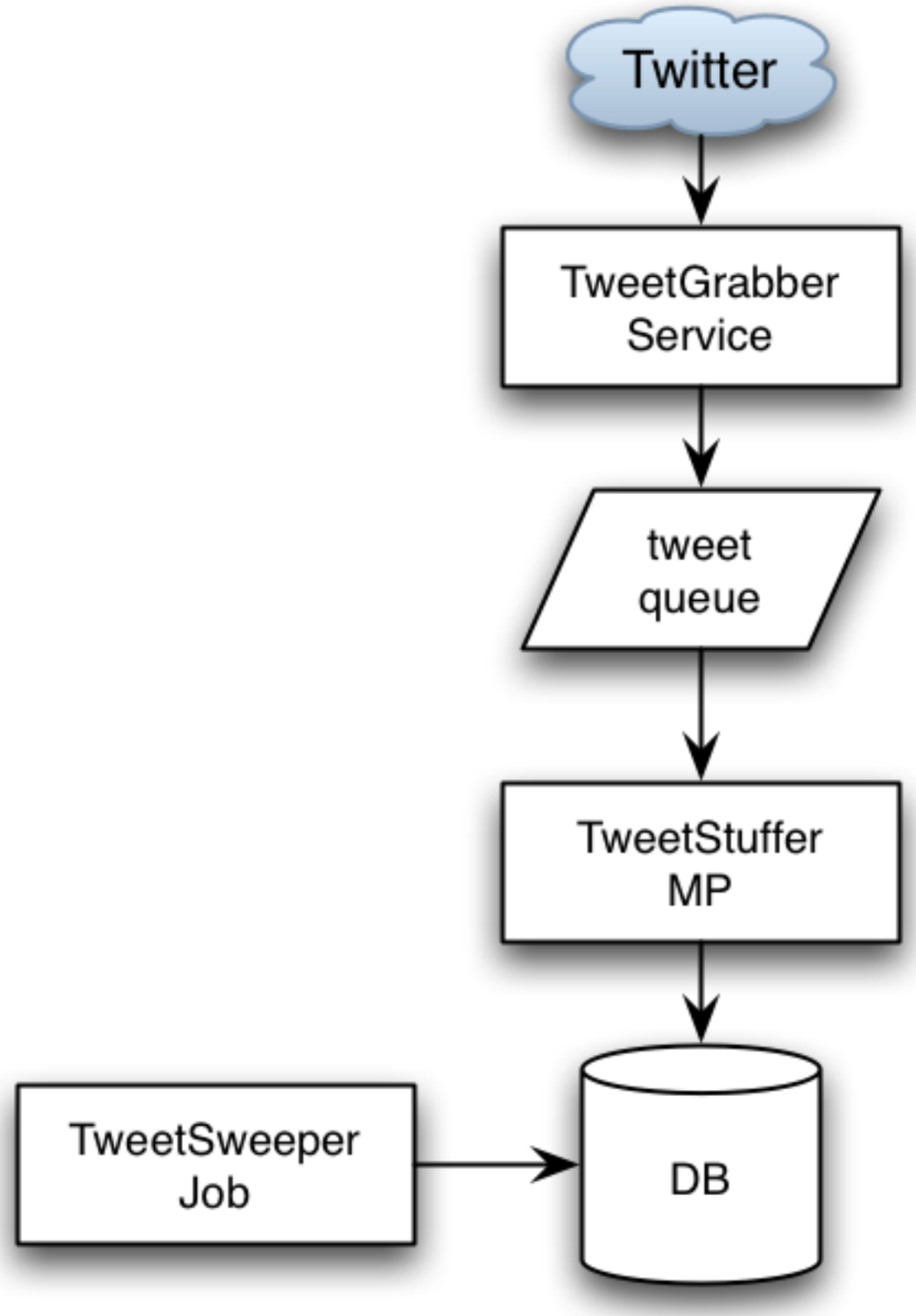
Commits

Lance
fb656070c7e8370d1ca8ccd47b9392fc26ce20b6
2011-02-09T13:12:20-08:00
Add tmp dir for auto deployment

Lance
2dee90fff7d87821126734889623e7cd9a06bd76
2011-02-09T13:12:50-08:00
Merge branch 'torquebox' of github.com:lance/buyappalachian.org into torquebox

Live Demo

wish me luck


Twitter
TweetGrabber Service
tweet queue
TweetStuffer MP
TweetSweeper Job
DB

Roadmap

Soon - 2.0 (AS7)

Then...

Authentication

Mobicents

??? - you tell us

Resources

- http://torquebox.org
- irc: #torquebox on freenode
- https://github.com/torquebox
- http://twitter.com/torquebox

Thanks!

questions?