

Ruby on Rails (RoR) as a back-end processor for Apex

Espen Braekken
Director of Consulting
Senitel Consulting AS

First

- English?
- Norwegian?

- Questions
- Spørsmål

Agenda

- About me
- About Senitel Consulting
- Why do we need a supplement to Apex?
- Why Ruby and Rails (and friends)?
- Examples of Ruby in action
- Notes on deployment and integration
- How to get started
- Wrap up

About me

- Director of Consulting, Senitel
- Director of Sales and Services, dbWatch
- Practice Manager, Oracle
 - Performance and Technical Architecture
- Director of Engineering, e-Travel
- Technical Manager, Oracle Consulting
 - System Performance Group



Senitel Consulting AS

- Established 1996 with focus on Oracle
- Owned by employees
- Focus
 - Web development (Oracle Application Express)
 - Data warehouse / BI solutions
 - Database
 - Training and workshops
 - Oracle licensing

Why Apex?

Great integrated environment:

- Fast
- Full Oracle stack
- DBA friendly
- PL/SQL
- Same language for back-end and front-end

=> gives small Oracle teams same advantage as that of more famous and “sexy” frameworks: Ruby on Rails, Zend and more.

Why do we need a supplement to Apex?

Things that are difficult:

- Communication with the operating system
- Communication with command line scripts, potentially on other physical machines
- Receiving and parsing emails
- ssh and (s)ftp

May end up with brittle solutions that fail at random.

Warning!

- Our approach is to use Ruby as glue (ref Perl: The Glue of the internet)
 - (slightly) changed from what is in the program, based on experience last 6 months
- Learning a new language can be addictive
 - General understanding
 - Skills
 - New perspective on your native language
 - Fun!

Why Ruby and Rails (and friends)?

- Talk to command line
- Run from command line
- Integrate with multiple data sources (MySQL, MSSQL, Oracle)
- Wide choice of communication protocols
- Wide range of add-ins/tools
- Short development cycles
- Easy to deploy

A typical windows script.

```
set BIN_DIR=%~dp0
set LOG_DIR=%BIN_DIR%\..\log
```

A typical *nix script

```
if [ `uname` = "SunOS" ]; then
  BIN_DIR=`/bin/dirname $0`
else
  BIN_DIR=`/usr/bin/dirname $0`
fi
LOG_DIR=${BIN_DIR}/../log
```

In Ruby

```
bin_dir=File.dirname(__FILE__)
log_dir=bin_dir+'../log'
```

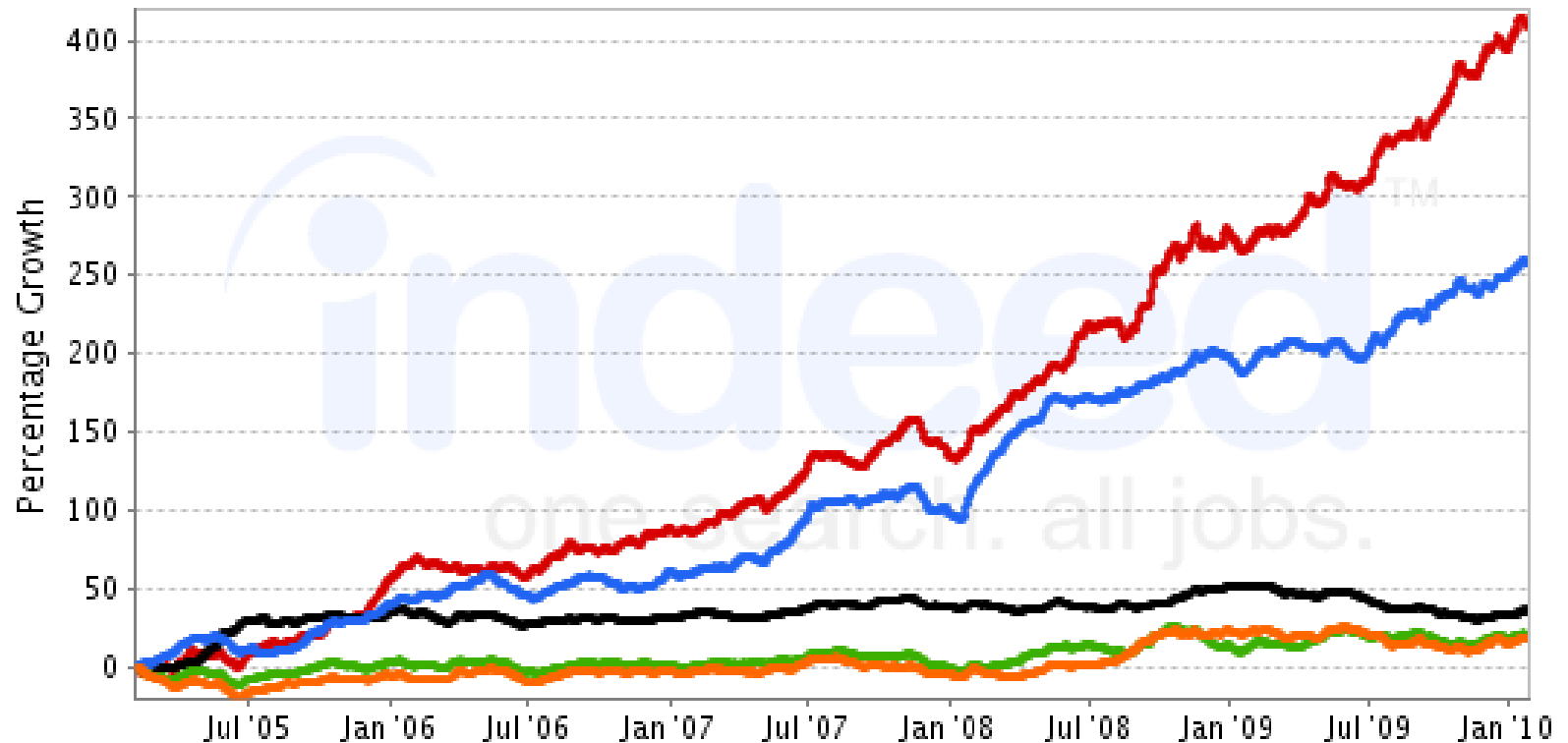
This works on all platforms.
The Ruby file IO library
will translate to
local "naming" standard.

PHP and Perl crashing the enterprise party

Scale: [Absolute](#) - Relative

Job Trends from Indeed.com

— Java — PHP — Perl — .Net — Python

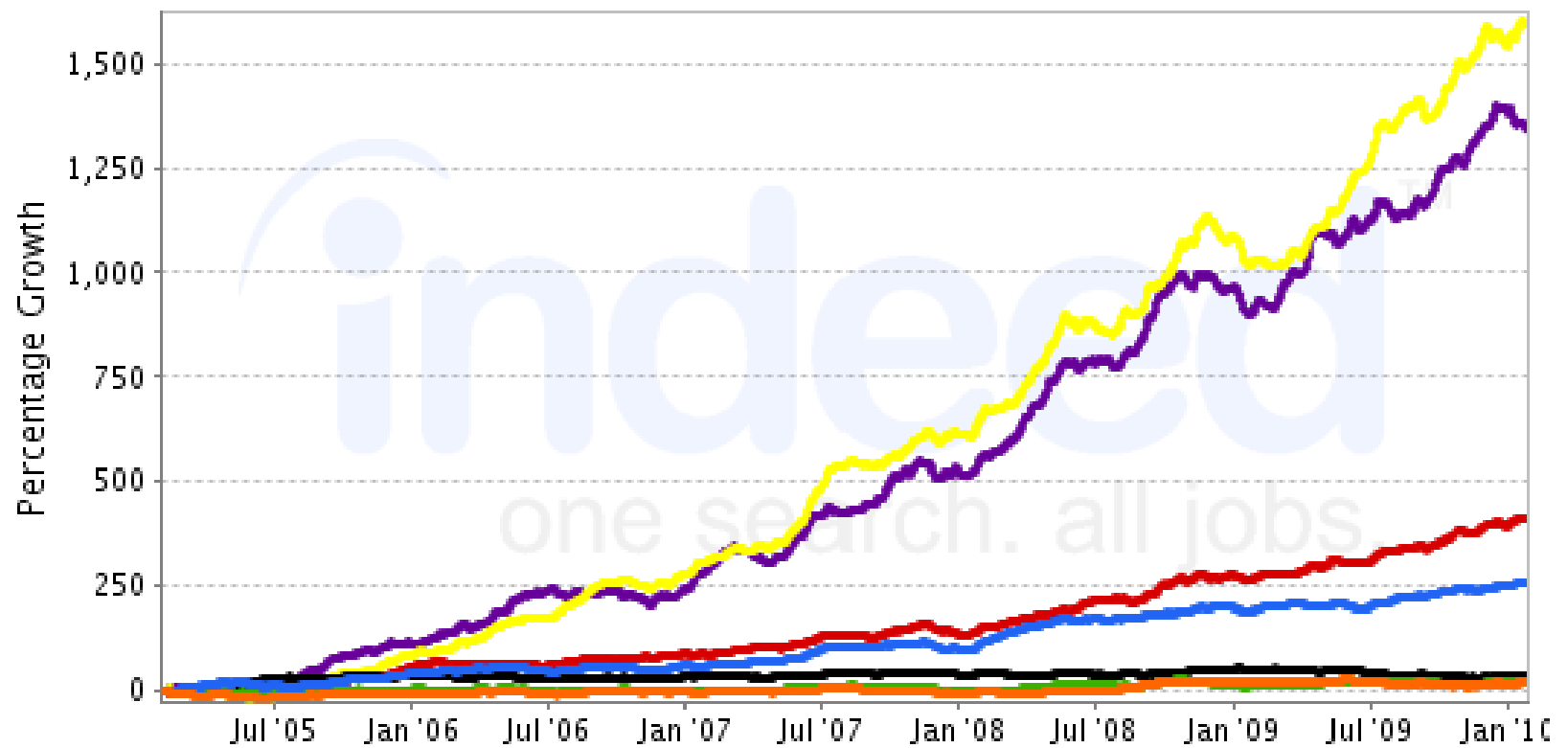


Take the A Train

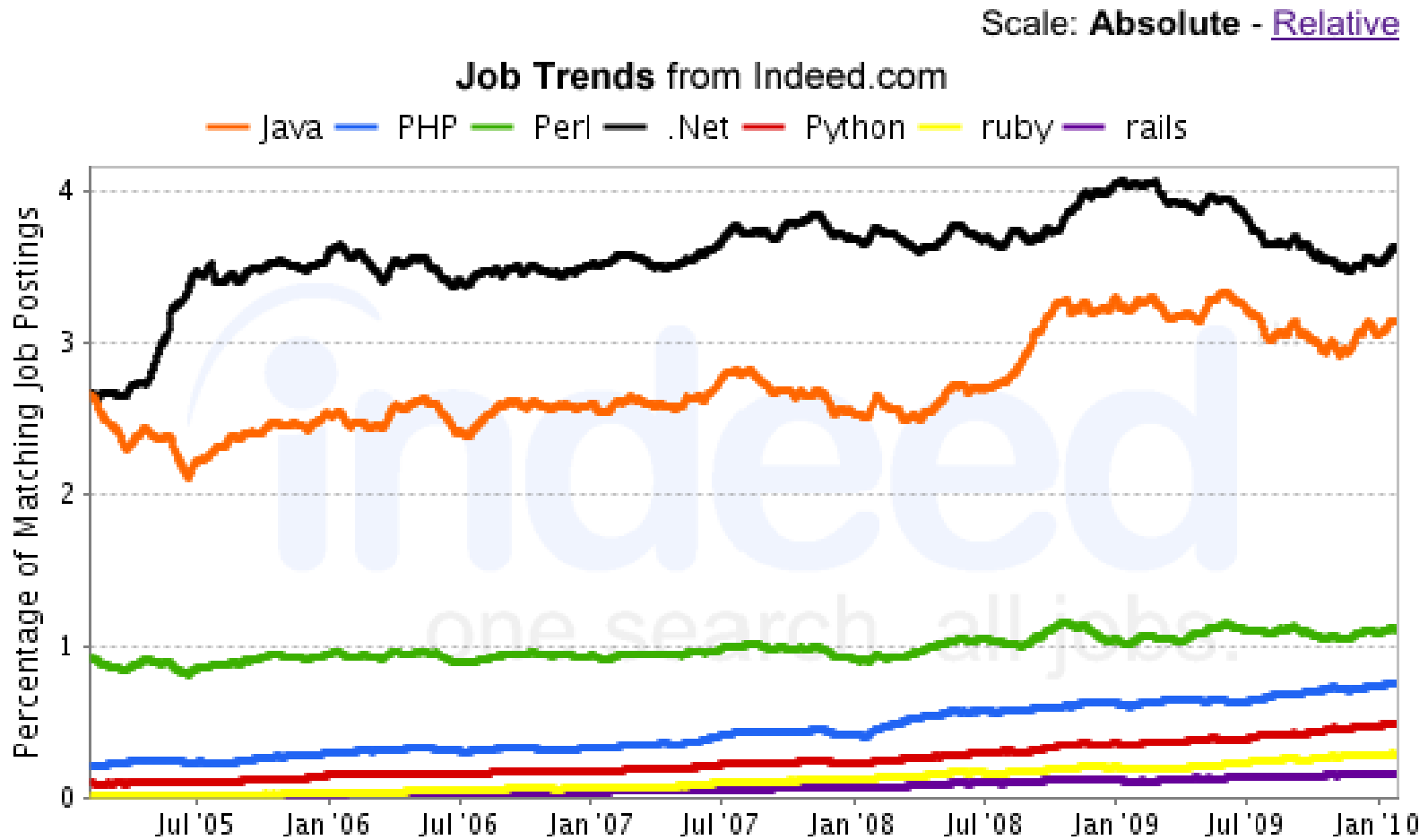
Scale: [Absolute](#) - Relative

Job Trends from Indeed.com

— Java — PHP — Perl — .Net — Python — ruby — rails



Huh – those numbers are relative!



Ruby, what is it?

- Ruby is a dynamic, reflective, general purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features.
- Ruby originated in Japan during the mid-1990s and was initially developed and designed by Yukihiro "Matz" Matsumoto.
- It was influenced primarily by Perl, Smalltalk, Eiffel, and Lisp.

Source: wikipedia

Ruby

- C like syntax.
- The standard 1.8.7 implementation is written in C, as a single-pass interpreted language
- Runs in a VM
- Available on “all” platforms:
 - Linux
 - Mac
 - Windows
 - Java

Rails, a short introduction

- A Model-View-Controller framework
- Open Source
- Web applications
- Developed by David Heinemeier Hansson (DHH) of 37signals
- Principles:
 - Convention over configuration
 - Don't repeat yourself (DRY)
- Similar frameworks
 - CakePHP
 - CodeIgniter

What does this mean?

- Convention over configuration
 - Almost no configuration file setup
 - Only define specifics
 - Everything else is based on “sensible” defaults
- Don't Repeat Yourself (DRY)
 - Extensive re-use
 - Easy to add own methods as helpers

Configuration files

```
find config -name "*" -print | grep -v svn
config
config/boot.rb
config/database.yml
config/environment.rb
config/environments
config/environments/development.rb
config/environments/production.rb
config/environments/test.rb
config/initializers
config/initializers/backtrace_silencers.rb
config/initializers/inflections.rb
config/initializers/mime_types.rb
config/initializers/new_rails_defaults.rb
config/initializers/session_store.rb
config/locales
config/locales/en.yml
config/routes.rb
```

yml, not xml

```
# sample database.yml file
development:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_development
  driver: oracle.jdbc.driver.OracleDriver

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_test
  driver: oracle.jdbc.driver.OracleDriver

production:
  adapter: jdbc
  username: apex
  password: sekret
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_production
  driver: oracle.jdbc.driver.OracleDriver
```

Rails and friends -- I

- We «lied» ... when we talk about Rails, we really mean Ruby++
- Our use of Rails is more on the back-end than the front
 - **ActiveRecord (ORM)**
 - ActionMailer
 - ActiveResource
 - ActionPack
 - ActiveSupport

Rails and friends -- II

- Back-end scripts with database functionality
- Alternatives to Rails:
 - Merb (will be merged with Rails in 3.0)
 - Sinatra (for lightweight RESTful services)

Rails, how and when to use

- We use as a supplement, not instead of
- To create good applications with Rails, you need to work with it every day

Use under these conditions:

- Create RESTful services
- When you need support of multiple data sources (Oracle and MySQL)
- **When your command line scripts suite is becoming a nightmare**

ActiveRecord, the gem in Rails

- Object Relational Mapping (ORM)
- Table names in plural
- PK is always id (number)
- FK always *table-name-in-singular_id*
- For Oracle a sequence per table:
 - *table-name-in-singular_seq*

```
# db tables used in example:  
# web_logs(id number, entry varchar2(4000))  
  
# find all, put into list  
wl_list = WebLog.find :all  
  
# create new  
wl=WebLog.new  
  
wl.entry="Today I am presenting at OUGN"  
wl.save
```

```
ActiveRecord::Base.establish_connection :user=>"username",  
:password=>"sekrit" .....
```

Not too elegant, let's use a yml file instead (next slide)

```
yml_filename="../../config/database.yml"  
my_config=YAML::load(File.open(yml_filename))["development"]  
  
ActiveRecord::Base.establish_connection my_config
```

database.yml

```
development:  
  adapter: jdbc  
  username: apex  
  password: sekret  
  url: jdbc:oracle:thin:@1.1.1.1:1521:apex_development  
  driver: oracle.jdbc.driver.OracleDriver
```

ActiveRecord, mapping objects to tables

```
class WebLog < ActiveRecord::Base  
  
end
```

ActiveRecord, relations

```
class WebLog < ActiveRecord::Base
  has_many :comments
end

class Comment < ActiveRecord::Base
  belongs_to :web_log
end
```

```
# db tables used in example:
# web_logs(id number, entry varchar2(4000))
# comments(id number, the_comment varchar2(40000), web_log_id number).

# connect omitted for brevity

# set comment
wl_id = 33
my_comment="You rock!"

# find web log entry to comment on
wl=WebLog.find_by_id wl_id

# build comment as child of web log
wlc=wl.comment.build

# set fields
wlc.the_comment = my_comment

# Now save
wl.save

# that's all folks!
```

What about stored procedures?

- There is a very comprehensive library for calling PL/SQL: **ruby-plsql**
- Developed by Oracle ACE Raimonds Simanovskis – Oracle Developer of the Year 2009
- We have not been successful in using with JRuby/JDBC
- Seems to have bindings to OCI based Oracle adapter
- Use if Java/JRuby is not important to you

Sample projects

Sample projects – I

- Dynamic creation of sql-loader scripts
 - New loader file for each input file
 - Constants and status codes
 - Modeled after Rails' implementation
 - ERb

```

load data
append
into table my_table
fields terminated by ';'
trailing nullcols
(
  prod char
  , kunde char "trim(:kunde)"
  , dekning char
  , navn char
  , personnr char
  , gateadresse char
  , postnr char
  , poststed char "trim(:poststed)"
  , gateadresseobj char "trim(:gateadresseobj)"
  , postnrobj char "trim(:postnrobj)"
  , poststedobj char "trim(:poststedobj)"
  , fradato char "to_date(substr(:fradato,1,10),'dd.mm.yyyy')"
  , tildato char "to_date(substr(:tildato,1,10),'dd.mm.yyyy')"
  , oppsigelse char "to_date(substr(:oppsigelse,1,10),'dd.mm.yyyy')"
  , bygger char "decode(length(trim(:bygger)), 4, trim(:bygger), '1899')"
  , bygning char "trim(:bygning)"
  , krypkjeller char "trim(:krypkjeller)"
  , byggete char
  , premie char
  , sc_insdte sysdate
  , sc_insby constant '<%= sc_insby %>'
  , sc_filename constant '<%= File.basename(sc_filename) %>'
  , sc_status constant '<%= sc_status.capitalize %>'
)

```

```
, sc_insby constant '<%= sc_insby %>'  
, sc_filename constant '<%= File.basename(sc_filename) %>'  
, sc_status constant '<%= sc_status.capitalize %>'
```

```
sc_filename='afile.dat'  
sc_insby='Espen'  
sc_status='Imported'  
template_file_name="afile.erb"  
template=File.read(template_file_name)  
  
controlfile=ERB.new(template).result(binding)
```

Sample projects – II

- Generic remote file management
 - Supports ssh, ftp and s3
 - Basic operations on remote folders
 - ls
 - save (put)
 - download (get)
 - Operations to/from files or in-memory objects
 - Can stream from remote file directly to LOB

```
# 1st, overload FTP so that we can download as stream
module Net
```

```
class FTP
```

```
  # note that line breaks, comma signals continuation
  def getbinaryfile_as_stream(remotefile, blocksize = DEFAULT_BLOCKSIZE,
    &block) # :yield: data
    raise "Need to provide a block" unless block
```

```
    # no retry, but can be checked in @resume .. check source
    rest_offset=nil
```

```
    retrbinary("RETR " + remotefile, blocksize, rest_offset) do |data|
      yield(data)
    end
  end
```

```
  def gettextfile_as_stream(remotefile, &block) # :yield: line
    raise "Need to provide a block" unless block
```

```
    retrlines("RETR " + remotefile) do |line|
      yield(line)
    end
  end
```

```
  # note that line breaks, comma signals continuation
  def putbinaryfile_from_object(remotefile, data,
    blocksize = DEFAULT_BLOCKSIZE)
    rest_offset=nil
    f=StringIO.new(data, 'r')
    storbinary("STOR " + remotefile, f, blocksize, rest_offset)
  end
```

```
  def puttextfile_from_object(remotefile, data)
    f=StringIO.new(data, 'r')
    storlines("STOR " + remotefile, f)
  end
```

```
end
```

```
end
```

```
# note that line breaks, comma signals continuation
def getbinaryfile_as_stream(remotefile, blocksize = DEFAULT_BLOCKSIZE,
  &block) # :yield: data
  raise "Need to provide a block" unless block

  # no retry, but can be checked in @resume .. check source
  rest_offset=nil

  retrbinary("RETR " + remotefile, blocksize, rest_offset) do |data|
    yield(data)
  end
end
```

Sample projects – III

- Calling Java code
- Ruby wrapper with exception handling, parameters etc for exporting (backing up) Apex code.

```

require 'java'

# Needs to be in CLASSPATH
import "oracle.apex.APEXExport"

# pass all params as String!
def export_application(db, user, password, applicationid)

  # get handle to Java objects
  my_exporter=APEXExport.new()
  my_export_class=my_exporter.class

  # build arg list, multi-line for readability
  argv=[]
  argv << "-db" << db
  argv << "-user" << user
  argv << "-password" << password
  argv << "-applicationid" << applicationid

  # to java
  jargs=argv.to_java(:string)

  # output is f<applicationid>.sql, in directory where code runs
  my_export_class.main(jargs)
end

def main(argv)
  es="Use: #{__FILE__} dbhost.senitel.no:1521:db user pass applicationid"
  raise es unless argv.size==4
  export_application(argv[0], argv[1], argv[2], argv[3])
end

#
# Launch main if not irb
#
if $0 == __FILE__
  begin
    main(ARGV)

  rescue Exception => e
    $stderr.puts "Error: #{e.to_s}"
    exit 1
  end
end
end

```

Deployment, alternatives

- We use Jruby and Tomcat/Glassfish for all our deployments
 - Can use any J2EE container
- This means running your Ruby code in a Java Runtime environment
- Reduces your Rails deployment to copying a war file onto the server
- Familiar ground for sysadmins, both from deployment and everyday management perspective
- J2EE is the platform; Operating system of no consequence

How to get started

- Learn the language
 - Ruby first, then a framework
- Start small
 - Replace shell/bat scripts with Ruby first
 - Then add DB interaction
- Wait with frameworks until you know Ruby
- Assess your needs
 - No UI/Html => Merb or Sinatra

Wrap up

- Work with tool, not against it
 - Apex where it shines
 - Ruby++ where needed
- Dynamic languages is one the rise
 - Ruby/Rails the fastest riser
- Don't be fooled, high productivity does not equal easy!

<http://www.slideshare.net/ebraekke/presentations>