RoR Administration

1 Day Class

Course Materials

- https://github.com/nirds/1Day_Courses

Welcome

Introductions

Introductions

Name
Experience
Expectations

Curriculum
The RoR Ecosystem

- Ruby, Implementations, RubyGems
- Rails, Bundler, Environments, Patterns, Deployment, Exceptions
- Development Tools, RVM, GIT, Pry, Guard, Services, Performance
- Web Server, Application Server, Load Balancing, DB Tier, Job Workers

Schedule

Times	Day 1	If Time Allows
9am - 10:30	Intro and Implementations	
Break		
10:45 - 12	Rails, Exceptions, Workers, Caching	Web and App Servers
Lunch		
1pm - 2:30	Security and Deployments	Load Balancing and Scaling
Break		
2:45 - 4:30	Development Tools and Monitoring	

Learning Goals

- Ruby on Rails Ecosystem
- Architecture Decisions
- Common Patterns (Setups)
- Where to go for help (Resources)

Ruby Implementations

- https://github.com/cogitator/ruby-implementations/wiki/List-of-Ruby-implementations
- Comparisons:
  - http://www.rubyinside.com/ruby-implementation-shootout-a-bright-future-for-ruby-performance-1390.html
  - http://benchmarksgame.alioth.debian.org/u64q/benchmark.php?test=all&lang=jruby&id=2&data=u64q
- Benchmarking:
  - http://www.ruby-doc.org/stdlib-1.9.3/libdoc/benchmark/rdoc/Benchmark.html
  - https://github.com/acangiano/ruby-benchmark-suite
- Great Blog about Optimizing a New Impl:
  - http://blog.headius.com/2012/10/so-you-want-to-optimize-ruby.html

Ruby 2.1 features: http://rkh.im/ruby-2.1
Benchmark results and commentary: http://miguelcamba.com/blog/2013/10/05/benchmarking-the-ruby-2-dot-1-and-rubinius-2-dot-0/

RVM Demo

- Gemset for Rails Admin
  - rvm use --create 2.0.0@rails_admin
  - gem install rails
  - mkdir rvm_test
  - cd rvm_test
  - rvm --rvmrc --create

Git Demo

- mkdir git_test
- cd git_test
- git init
- touch test_file.txt
- git status
- git add .
- git commit -m “add a file”
- git status

Git Demo

- cd ..
- git clone git_test/.git clone_test
- cd clone_test
- ls
- git remote -v
- touch clone.txt
- git commit -am “add another file”

Git Hooks

- post-receive hook
- post receive email:
- https://github.com/git/git/blob/master/contrib/hooks/post-receive-email

Git Helpers

- Git-Flow:
  - http://nvie.com/posts/a-successful-git-branching-model/
  - https://github.com/nvie/gitflow
  - http://danielkummer.github.io/git-flow-cheatsheet/
- Git Remote Branch (grb gem)

Rails Composer

- Easy test app

Exceptions

- https://www.ruby-toolbox.com/categories/exception_notification
- AirBrake with Errbit
- Squash.io

Mailers

- Add an email about new posts in a topic

Workers

- Beanstalk (stalker gem and beanstalkd)
- Resque (resque_mailer) (redis)
- Sidekiq (redis)
- delayed_job (db table)

Rake Tasks

- DSL for Tasks
- Rails Tasks
- Add In Build Tasks

Caching

- http://guides.rubyonrails.org/caching_with_rails.html

DB

- SQL
  - MySQL (RDMS)
  - PostgreSQL (ORDMS)
  - Sqlite
  - MSSQL, Oracle, DB2
- NoSQL
  - MongoDB
  - CouchDB
  - Hadoop

Developer Tools

- https://www.ruby-toolbox.com/#Developer_Tools

Command Line

- Z-Shell
- tmux
- screen
- pry

Staying Current

- https://gemnasium.com/
- https://github.com/gemnasium/gemnasium-gem
- http://weblog.rubyonrails.org/

Performance

- new_relic
- https://www.ruby-toolbox.com/categories/rails_instrumentation

Benchmarking

- http://guides.rubyonrails.org/performance_testing.html
- https://github.com/acangiano/ruby-benchmark-suite

Monitoring

- Monit
- God
- Nagios

Debugging

- Pry
- Airbrake
- better_errors
- NewRelic

Caching

- Caching data
- Caching Assets
- CDN

Staying Current

- Gemnasium
- Ruby on Rails Blog

Monitoring

- God (ruby gem, very simple)
- Monit
- Nagios

Security

- Least Privilege
- Secure Defaults
- Defense in Depth

least privilege
(deploy user)
secure by default
- whitelist
- safe api - active record query api - safe way (escaping)
automatic sanitizing (cross site scripting default)

Defense in Depth
- layers of defense - layers buy more time (security)
- like not worry about script injection in Admin area -> whats the impact?

Security Issues

- Session Hijacking (firesheep)
- Brute Force Attacks (psswd stolen, sha1: 9 chars, 7 years)
- Insecure Direct Reference (proper auth)
- Mass Assignment (strong_parameters)
- CSRF (watch for GET)
- Offsite Redirects
- Unexpected Code Execution (routes, and render)
- SQL Injection (http://rails-sqli.org/)
- XSS (sanitize)
- YAML Injection (http://blog.codeclimate.com/blog/2013/01/10/rails-remote-code-execution-vulnerability-explained/)

Attacks:
1. session hijacking - firesheep (made it easy, drew attention to it, ssl for login but not for every view
- force ssl (everywhere) -> config.force_ssl = true (nginx for ssl termination)
- secure and httponly cookies (done for you on most rails versions - should have secure and httpOnly)
- strict transport security - force ssl sends this by default
- require a password for sensitive stuff
2. craft the url to send params -> make sure you are checking proper authorization for data you are presenting (roles? maybe over-kill…) -> force checking authorization
3. mass assignment - rails 4 has strong_parameters - requires a object (hash) in the params then permit what fields are allowed in that hash
4. csrf - only apply to post etc.. so you have to be safe with GETs
5. leaky routes - not a default thing - you make it accessible -> strict limits on actions in routes, also user specifications in render
Offsite redirects
Good: Verify protocol and host
Better: Create a whitelist
Best: Use an identifier instead of a URL
6. sql injection (use the query interface) but be careful know the AR api - uncommon (calculate and pluck)
rails-sqli.org
7. cross site scripting (XSS) -> malicious js -> Rails 3+ -> rails_xss, sanitize user generated html (Loofah gem) -> content security policy (tell the browser what is safe) -> raw and html_safe and link_to can inject values if u generate html from user settings ….

BrakeMan

- http://brakemanscanner.org/
- https://github.com/presidentbeef/brakeman
- http://brakemanscanner.org/docs/options/

Deployment

- Capistrano / Capistrano-Multistage (RVM & Bundler)
- Git
- Rake

Configuration

- Environment Variables
- Config Files
- Chef DataBags

Database

- SQL
  - MySQL (RDMS)
  - PostgreSQL (ORDMS)
  - Sqlite
  - MSSQL, Oracle, DB2
- NoSQL
  - MongoDB
  - CouchDB
  - Hadoop

Database Comparisons

- http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems#Operating_system_support
- http://www.slideshare.net/octplane/mongodb-vs-mysql-a-devops-point-of-view

ACID Compliance

- ACID model:
  - A: atomicity.
  - C: consistency.
  - I: isolation.
  - D: durability.

Application Servers

- Rails Application Processing
- Passenger
- Thin (EventMachine)
- Unicorn (Good with Low Latency)

Web Servers

- Nginx
- Apache

Scaling

- Horizontal (Moar Stuff!)
- Vertical (Moar Power!)

- Code School Videos Scaling Rails

DB Scaling

- http://blog.sphereinc.com/2010/07/5-ways-to-boost-performance-of-your-rails-applications/

Load Balancing

- HA Proxy
- HeartBeat
- Nginx (SSL Off Loading)
- Apache Round Robin

Load Balancing / Scale

- http://www.slideshare.net/salizzar/porting-rails-apps-to-high-availability-systems (slide 49)

Rails Server Needs

- unprivl user (deploy) with this in the path
- ruby (rvm)
- ruby gems
- Webserver (nginx, apache) (typically user running is in the same group as deploy (like www))
- - environment vars set by the web server
- - points to the public directory
- Application server (passenger, thin, unicorn)
- App Directory (public)
- Assets compiled
- Database (MySQL, Mongo, PostGresQL, etc...)
- Database configuration
- Configuration fies
- (passenger and rvm - configure load paths...)

Different Configurations

- Talk Through Options