



D08 - Ruby on Rails Training

Servers

Summary: You will need a little knowledge about system administration and shell-scripting when your application goes live. You will learn how to configure nginx, puma and capistrano.// According to the environment, a Rails application and a server will be configured differently. You're about to learn everything about the infamous production phase.

Contents

I	Preamble	2
II	Ocaml piscine, general rules	4
III	Today's specific instructions	6
IV	Exercise 00: DevOps Skillz	7
V	Exercise 01: Ruby DevOps Skillz	10

Chapter I

Preamble

Why Are So Many Websites Hosted On Linux?

tl;dr: Because of Windows's windowy things ,and Linux's unixy things

1. Stability

Linux systems are well known for their ability to run for years without failure; in fact, many Linux users have never seen a crash. That's great for users of every kind, but it's particularly valuable for small and medium-sized businesses, for which downtime can have disastrous consequences.

Linux also handles a large number of processes running at once much better than Windows does—that's something, in fact, that tends to degrade Windows' stability quickly.

Then there's the need for rebooting. Whereas in Windows configuration changes typically require a reboot—causing inevitable downtime—there's generally no need to restart Linux. Almost all Linux configuration changes can be done while the system is running and without affecting unrelated services.

Similarly, whereas Windows servers must often be defragmented frequently, that's all but eliminated on Linux. Let your competitors endure the plentiful downtime that inevitably goes hand-in-hand with Windows; trusty Linux will keep you up and running and serving your customers around the clock.

2. Security

Linux is also innately more secure than Windows is, whether on the server, the desktop or in an embedded environment. That's due largely to the fact that Linux, which is based on Unix, was designed from the start to be a multiuser operating system. Only the administrator, or root user, has administrative privileges, and fewer users and applications have permission to access the kernel or each other. That keeps everything modular and protected.

Of course, Linux also gets attacked less frequently by viruses and malware, and vulnerabilities tend to be found and fixed more quickly by its legions of developers and users. Even the six-year-old kernel bug that was recently fixed, for instance—an extremely rare instance in the Linux world—had never been exploited.

Internally, meanwhile, users of a Windows system can sometimes hide files from the system administrator. On Linux, however, the sys admin always has a clear view of the file system and is always in control.

3. Hardware

Whereas Windows typically requires frequent hardware upgrades to accommodate its ever-increasing resource demands, Linux is slim, trim, flexible and scalable, and it performs admirably on just about any computer, regardless of processor or machine architecture.

Linux can also be easily reconfigured to include only the services needed for your business's purposes, thus further reducing memory requirements, improving performance and keeping things even simpler.

4. TCO

There's no beating Linux's total cost of ownership, since the software is generally free. Even an enterprise version purchased with corporate support will be cheaper overall than Windows or other proprietary software, which generally involve user-based licensing and a host of expensive add-ons, especially for security.

Same goes for most of the tools and applications that might be used on a Linux server. The overall TCO simply can't be beat.

5. Freedom

With Linux, there is no commercial vendor trying to lock you into certain products or protocols. Instead, you're free to mix and match and choose what works best for your business.

In short, with all the many advantages Linux provides in the server realm, it's no wonder governments, organizations and major companies around the world—including Amazon and Google—rely on the open source operating system in their own production systems.

If you're looking for a Linux distribution to run on your business's servers, you'd do well to consider CentOS (or RHEL, the paid version from Red Hat that CentOS is based on), Slackware, Debian and Gentoo.

Chapter II

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter III


Today's specific instructions

You cannot install the "nginx" gem.

No script will be uploaded or fetched within the turned in script. The "Fubar" app is the only element uploaded on Bitbucket. It is synchronized during the deployment with capistrano in the ex01

Chapter IV

Exercise 00: DevOps Skillz

	Exercise 00
Exercise 00: Shell_flavored	
Turn-in directory : <i>ex00/</i>	
Files to turn in : create_server.sh	
Allowed functions :	

To start this day about servers and production phase, you will have to turn-in a script that, when executed within a brand new vm, will install and set up a server for the "fubar" application. This will be ex00.

To do so, the script will execute the following commands in this specific order:

- the installation of git, curl and your favorite text editor
- the installation of rvm
- the installation of ruby and rails via rvm
- the installation of all necessary dependancies
- the correct configuration of the **etc/hosts** file
- the creation of the "fubar" sample application with the creation of DBs, models migration and their respective populating.
- the configuration of the 'SECRET_KEY_BASE'
- starting the **puma** server with all the production environment configuration.

A few tips about the following page.

The script you will turn-in must include the following code for the creation of the sample application.

```
$> cat create_server.sh
[...]
mkdir /home/vagrant/site
cd /home/vagrant/site
rails new fubarre -d postgresql
cd fubar
rails g scaffold component great_data
echo "Component.create(great_data: 'foo_bar_name')" >> db/seeds.rb
bundle install
sed -i -e "s/username: fubar/username: vagrant/g" config/database.yml
RAILS_ENV=production rake db:create
RAILS_ENV=production rake db:migrate
RAILS_ENV=production rake db:seed
sed -i "2iroot to: 'components#index'" config/routes.rb
echo "<h1><%=Rails.env%></h1>">app/views/components/index.html.erb
[...]
$>
```

First, make sure you have `virtualbox` and `vagrant`. Then, create a folder named "ex00". Run the following command to its roots:

```
vagrant init hashicorp/jessie64
```

Replace the initial "Vagrantfile" file with:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # Use Debian Jessie 64-bit stable as our operating system
  config.vm.box = "debian/jessie64"

  # Configure the virtual machine to use 1GB of RAM
  config.vm.provider :virtualbox do |vb|
    vb.customize ["modifyvm", :id, "--memory", "1024"]
  end

  ## Un-comment this line in order to copy your script into the VM
  ## with the "vagrant provision" command :
  #config.vm.provision "file", source: "create_server.sh", destination: "~/create_server.sh"

  # Forward the Rails server default port to the host
  config.vm.network :forwarded_port, guest: 3000, host: 3031
  config.vm.network :forwarded_port, guest: 80, host: 8090
end
```

You will then possess a vm symbolizing your server. Don't update the distribution!

Now you can mount the VM thanks to the `vagrant up` command and log to it thanks to `vagrant ssh`.



To manage the space on your homes, we HIGHLY recommend you place the VirtualBox VMs and `.vagrant.d` folders anywhere BUT in your home. Place it in the "Pig" and make the necessary symbolic links for instance.

You **must** assign a root password for the vagrant user. You should really use 'vagrant'.

On your VM, you **must** change the "/etc/hosts" file line

```
127.0.0.1 localhost
```

en:

```
0.0.0.0 localhost
```


Uncomment the " config.vm.provision "file"... " line in Vagrantfile to copy your script within the vm with the command:

```
vagrant provision
```

If you've done everything right, you must see your application served by **puma** at the following address: "http://0.0.0.0:3031/", in your browser.

Chapter V

Exercise 01: Ruby DevOps Skillz

	Exercise 01
Exercise 01: Ruby_flavored	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>create_server_2.sh</code> , <code>create_app.sh</code>	
Allowed functions :	

In this exercise, we will use Capistrano, a library for deployment automation on a distant server.

This means that **locally**, from your **developpement** environment, you will connect in **ssh** on your server, synchronize your data from a SVN of your choice, keeping the last 5 versions and symlinks of the common configuration files in order to optimize the use of the disk.

Thus, **deploying** on a **distant** machine (that is a vm, here), an application in a **production** environment. And if everything works as it should, in one single command.

You will also have to use **nginx** in "reverse proxy". This will help you get a base ready to receive the "load balancing" configurations while benefiting from increased security.

In this exercise, you **must** turn-in **two** scripts.

The FIRST ONE, "create_app.sh":

You will use this script to recreate the application and thus, insert the Nginx configuration during the evaluation. This script will start as follows:

```
$> cat create_app.sh
rails new foubarre2 -d postgresql
cd fubar
rails g scaffold component great_data
echo "Component.create(great_data: 'foo_bar_name')" >> db/seeds.rb
sed -i "2iroot to: 'components#index'" config/routes.rb
echo "<h1><%=Rails.env%></h1>">app/views/components/index.html.erb
echo "group :development do" >> Gemfile
echo "  gem 'capistrano',          require: false" >> Gemfile
echo "  gem 'capistrano-rvm',      require: false" >> Gemfile
echo "  gem 'capistrano-rails',    require: false" >> Gemfile
echo "  gem 'capistrano-bundler', require: false" >> Gemfile
echo "  gem 'capistrano3-puma',    require: false" >> Gemfile
echo "end" >> Gemfile
bundle install
cap install
echo "_your_configuration_HERE_" > config/deploy.rb
# Your nginx configuration file MUST bien in your
# config folder within you app
touch config/nginx.conf
echo "_your_configuration_HERE_">config/nginx.conf
[...]
```

The creation script must also initialize Git at the root of your project and add as a "remote branch" a BitBucket private repository that will contain your Rails application. By the way, if you don't own a BitBucket account, it might be a good time to get one. And since we're nice fellas:

```
git init
git add .
git commit -m "first commit"
git remote add origin git@bitbucket.org:user_name/repo_name
git push -u origin master
```

Now, your folder is synchronized with the online repo. But for the sake of accuracy and equity, you will destroy the repo and start again for each evaluation.

The SECOND ONE, "create_server_2.sh":

Create a new vm at the root of the application:

```
vagrant init hashicorp/jessie64
```

Replace the initial "Vagrantfile" file with:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # Use Debian Jessie 64-bit stable as our operating system
  config.vm.box = "debian/jessie64"

  # Configure the virtual machine to use 1GB of RAM
  config.vm.provider :virtualbox do |vb|
    vb.customize ["modifyvm", :id, "--memory", "1024"]
  end

  config.vm.provision :shell, path: "create_server_2.sh"

  # Forward the Rails server default port to the host
  config.vm.network :forwarded_port, guest: 80, host: 8090
  config.vm.network :forwarded_port, guest: 22, host: 2222, id: "ssh", disabled: true
  config.vm.network :forwarded_port, guest: 22, host: 64673, auto_correct: true
end
```

Thus, with the "vagrant up" command, your vm will be earmarked with the "create_server_2.sh" script. Rather simple, isn't it?.

You must adapt this script using "create_server.sh" so it executes:

- the creation of the 'deploy' user with the password 'deploy_password'
- the installation of git, curl, nginx and your favorite text editor
- the installation of rvm
- the installation of ruby and rails, via rvm
- the installation of all the necessary dependencies
- the creation of a script named "post_deploy_symlink.sh"
- the crushing of the file "/etc/hosts" by:

```
127.0.0.1    foubarre.com
127.0.1.1    jessie.raw    jessie

::1         localhost ip6-localhost ip6-loopback
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```



Choose which user should install what. Knowing that the deploy user must not be among the subdoers and that Capistrano must log in and operate as deploy, which must have its ssh key added to BitBucket

The hosts file will have to include a rerouting of the `foubarre.com` address on the VM's IP in loopback. Thus we will be able to declare in the nginx configuration that:

```
server_name foubarre.com;
```

Nginx needs `/etc/nginx/sites-available/foobarre` to be a symbolic link to `/home-/deploy/apps/foobarre/current/config/nginx.conf`. This is the file that will contain your configuration so that `nginx` works in [reverse proxy](#) and it serves Puma. This is why the script `post_deploy_symlink.sh` must be created and executed after the first deployment.

If it all works well, you will be able to execute your first deployment:

```
bundle exec cap production deploy:initial
```

For the exercise to be validated, on the host system:

- We execute the `"create_app.sh"` script
- We set up the vm via the subject's Vagrantfile
- We make sure that the application is accessible via `"foubarre.com"` on the browser after deployment and execution in the vm of the script `"post_deploy_symlink.sh"`
- We can modify the local app and redeploy it:

```
git add modified_file
git commit -m "message"
git push origin master
bundle exec cap production deploy
```

AND you must make sure that within the VM:

- the app is in `"/home/deploy/apps"`
- the app's folder is `"own"` by the `deploy` user
- the `deploy` user is neither root nor in the subdoers



Don't rush into it. Read you logs: `this is the key.`