



D08 - Formation Ruby on Rails

English version coming soon!

Summary: Un peu de notions d'administration système et de shell-scripting vont être nécessaire à l'apparition de votre application sur le web. Vous allez apprendre à configurer nginx, puma, et capistrano.// Une application rails et un serveur se configurent différemment en fonction de l'environnement. Vous allez tout savoir sur la fameuse -Mise en prod-.

Contents

I	Préambule	2
II	Consignes	4
III	Règles spécifiques de la journée	6
IV	Exercice 00: DevOps Skillz	7
V	Exercice 01: Ruby DevOps Skillz	10

Chapter I

Préambule

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

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes:
 - Ruby `>= 2.3.0`
 - pour d09 Rails `> 5`
 - mais pour tous les autres jours Rails `4.2.7`
 - HTML `5`
 - CSS `3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while`, `for`, `redo`, `break`, `retry`, `loop` et `until` dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices: seul le travail présent sur votre dépôt GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet /`
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...

- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapter III


Règles spécifiques de la journée

Vous n'avez pas le droit d'installer de "gem 'nginx'".

Aucun script ne devra être uploadé et/ou fetché au sein du script rendu, seule l'app "foubarre" est uploadée sur Bitbucket et est synchronisée durant le déploiement par capistrano dans l'ex01

Chapter IV

Exercice 00: DevOps Skillz

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

En ce debut de journée sur les serveurs et la mise en production nous devez rendre, pour l'ex00, un script qui, executé au sein d'une vm toute fraiche, va installer et mettre en service un serveur pour l'application "foubarre" .

Pour ce faire, ce script executera les commandes suivantes dans l'ordre :

- l'installation de git, curl et votre editeur de texte préféré
- l'installation de rvm
- l'installation de ruby et rails via rvm
- l'installation de toutes les dépendances nécessaires
- la bonne configuration du fichier `etc/hosts`
- la création de l'application témoin "foubarre", avec creation des DBs, migration des models et leur peuplement respectifs.
- la pré-compilation des assets.
- la configuration de la `'SECRET_KEY_BASE'`
- la mise en route du serveur **puma** avec toute la configuration de l'environnement de production.

Quelques tips sur la page suivante.

Le script à rendre doit intégrer le code suivant pour la creation de l'application temoin.

```
$> cat create_server.sh
[...]
mkdir /home/vagrant/site
cd /home/vagrant/site
rails new foubarre -d postgresql
cd foubarre
rails g scaffold component great_data
echo "Component.create(great_data: 'foo_bar_name')" >> db/seeds.rb
bundle install
sed -i -e "s/username: foubarre/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
[...]
$>
```

Tout d'abord, vérifiez que vous disposez de **virtualbox** et de **vagrant**. Ensuite, créez un dossier nommé "ex00". Lancez à sa racine la commande:

```
vagrant init hashicorp/jessie64
```

Remplacez-y le fichier "Vagrantfile" initial par:

```
# -*- 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
```

Vous disposerez ainsi d'une vm symbolisant votre serveur. Ne mettez pas à jour la distribution !

Vous pouvez dès à présent monter la VM à l'aide de la commande **vagrant up** et vous y connecter a l'aide de **vagrant ssh**.



Pour des soucis de place sur vos homes, nous vous recommandons CHAQUEMENT de placer les dossiers VirtualBox VMs et .vagrant.d autre part que dans votre home. Placez dans le goinfre et faites les liens symboliques nécessaires, par exemple.

Vous devez assigner un password root pour l'user vagrant , il est fortement conseillé d' utiliser 'vagrant'.

Sur votre VM, vous devez changer la ligne du fichier "/etc/hosts"

```
127.0.0.1 localhost
```

en:

```
0.0.0.0 localhost
```


Décommentez la ligne " config.vm.provision "file"... " dans Vagrantfile pour copier votre script au sein de la vm avec la commande:

```
vagrant provision
```

Si tout est bien fait, vous devez voir votre application desservie par puma à l'adresse "http://0.0.0.0:3031/" de votre navigateur .

Chapter V

Exercice 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 :	

Dans cet exercice nous allons utiliser Capistrano, une librairie d'automatisation de déploiement sur serveur distant.

C'est à dire que depuis notre environnement de **developpement** en **local**, vous allez vous connecter en [ssh](#) sur votre serveur, synchroniser vos données depuis un SVN de votre choix, en conservant les 5 dernières versions et symlinks des fichiers de configuration communs dans un souci d'optimisation d'utilisation du disque.

Ainsi **déployer** sur une machine **distante** (ici une vm) une application dans un environnement de **production**, et ce, si tout fonctionne, en une seule commande.

Vous allez devoir aussi utiliser **nginx** en "reverse proxy", ce qui nous permettra d'avoir une base prête à recevoir des configurations de "load balancing", en bénéficiant d'une sécurité accrue.

Dans cet exercice, vous **devez** rendre **deux** scripts.

Le PREMIER , "create_app.sh":

Ce script vous servira à recréer l'application et ainsi insérer la configuration de Nginx lors de la correction. Ce script doit commencer comme suit :

```
$> cat create_app.sh
rails new foubarre2 -d postgresql
cd foubarre
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 "_votre_configuration_ICI_" > config/deploy.rb
# Votre fichier de configuration nginx DOIT etre dans votre
# dossier de config au sein de votre app
touch config/nginx.conf
echo "_votre_configuration_ICI_">config/nginx.conf
[...]
```

Le script de creation doit aussi initialiser Git à la racine de votre projet et ajouter comme "remote branch", un repository privé BitBucket sur lequel vous aurez mis votre application Rails. Au passage, si vous avez pas de compte sur BitBucket, c'est le moment les gens. Allez on est sympa :

```
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
```

Voilà, votre dossier est synchronisé avec le dépôt online. Mais pour des soucis de justesse et d'équité, vous devrez détruire le dépôt et recommencer à chaque correction.

Le DEUXIEME, "create_server_2.sh":

Créez une nouvelle vm à la racine de l'application :

```
vagrant init hashicorp/jessie64
```

Remplacez le fichier "Vagrantfile" initial par:

```
# -*- 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
```

Ainsi, avec la commande "vagrant up", votre vm sera provisionnée avec le script "create_server_2.sh", plutôt simple non? .

Vous devez adapter ce script en vous servant de "create_server.sh" pour qu'il opère:

- la création de l'user 'deploy' avec le password 'deploy_password'
- l'installation de git, curl, nginx et votre editeur de texte préféré
- l'installation de rvm
- l'installation de ruby et rails, via rvm
- l'installation de toutes les dépendances nécessaires
- la création d'un script nommé "post_deploy_symlink.sh"
- l'écrasement du fichier "/etc/hosts" par :

```
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
```



Choisissez bien quel user doit installer quoi. En sachant que l'utilisateur `deploy` ne doit pas figurer dans les `sudoers` et que `Capistrano` doit se connecter et faire ses opérations en tant que `deploy`, lequel doit avoir sa clé `ssh` ajoutée à `Bitbucket`

Le fichier `hosts` devra contenir un reroutage de l'adresse `foubarre.com` sur l'IP de la VM qui est en `loopback`. On pourra ainsi déclarer dans la configuration de `nginx` :

```
server_name foubarre.com;
```

`Nginx` a besoin que `/etc/nginx/sites-available/foubarre` soit un lien symbolique vers `/home/deploy/apps/foubarre/current/config/nginx.conf`. C'est le fichier qui va contenir votre configuration pour que `nginx` fonctionne en [reverse proxy](#) et serve `Puma`. C'est à cet effet que le script `post_deploy_symlink.sh` doit être créé et exécuté après le premier déploiement.

Si tout se passe correctement vous pouvez faire votre premier déploiement :

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

Pour que l'exercice soit valide, sur le système `host` :

- On exécute le script `"create_app.sh"`
- On initialise la vm via le `Vagrantfile` du sujet
- On vérifie que l'application soit accessible via `"foubarre.com"` sur le navigateur après déploiement et exécution dans la vm du script `"post_deploy_symlink.sh"`
- on peut modifier l'app locale et la re-déployer :

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

Et vous devez vérifier qu'au sein de la VM:

- l'app est dans `"/home/deploy/apps"`
- le dossier de l'app est bien `"own"` par le user `deploy`
- le user `deploy` n'est ni `root` ni dans les `sudoers`



Ne partez pas tête baissée, et lisez vos logs : c'est la clé.