01

Do hardware ao servidor web

Transcrição

Olá, meu nome é Guilherme!

Juntos, discutiremos sobre um sistema Java e como podemos inseri-lo em um ambiente de produção, de homologação, desenvolvimento e como replica-lo de diversas maneiras. Aprenderemos como criar o passo a passo de operações, colocar um sistema em produção, monitorá-lo e, por fim, automatizá-lo.

Agilidade é uma questão importante nos projetos atuais. Queremos adicionar mudanças em softwares o tempo inteiro e realizar diversos deployies. É claro, quanto mais deployies fazemos, maior o risco de erros. Então, temos de conseguir fazer análises temporais e replicar conteúdos.

Em um primeiro momento, achamos simples criar uma máquina nova: instalar o Linux, usar o `apt-get` e instalar o software de que você precisa. Depois, basta editar alguns arquivos de configuração utilizando VI. Por fim, fazemos uma cópia de alguns arquivos locais para um arquivo remoto com SCP. Essa é uma ideia bastante difundida.

Mas na prática há algumas complexidades: temos de conhecer exatamente o servidor do banco de dados usado, quais as configurações dele, a versão do Java, configurações no servidor de aplicação, do Servlet Conteiner. Precisamos saber, ainda, quais são os diretórios de uma máquina que são importantes para, por exemplo, se precisarmos realizar backups ao deletarmos uma máquina e criarmos outra. Quais são os usuários que eu tenho que criar? Quais são as portas que devo fechar e as portas que devo deixar abertas? Quais são os usuários, as senhas e as chaves que devo manter? Quais são os serviços que têm que rodar na minha máquina? Tudo isso deve estar contido no backup.

Todo o processo está sujeito a falhas, principalmente se é um ser humano que está o executando. O Mais importante é toda a vez que executarmos um deploy, ele se dê sem falhas e sempre da mesma forma, isto é, automatizado. Não devemos precisar de força humana para resolver problemas de *provisioning*.

Precisamos conseguir recriar uma máquina com agilidade caso aconteça algum problema que demande a destruição de uma máquina anterior, ou mesmo realizar atualizações de software com agilidade. Sabendo lidar com problemas de forma prática e rápida, conseguimos aproveitar projetos de um cliente para usar o mesmo modelo com clientes diferentes. Não podemos correr riscos ou perder tempo.

Neste curso, criaremos uma máquina nova para o desenvolvedor u para homologação e até mesmo criar uma máquina para produção no *cloud*. Instalaremos o Java, um servidor web, um banco de dados e configurar todos eles. Todo esse processo se dará de uma maneira automatizada, e não importa quantas vezes ele se repita, sempre teremos um mesmo modelo de máquina. Poderemos duplicar e recriar máquinas tranquilamente, esse é o processo de Devops.

O projeto em que trabalharemos é o **Music Jungle**.

Para criarmos uma máquia de produção ou homologação,precisamos primeiramente de um hardware. Para ter um hardware configurado, se eu for fazer uma máquina virtual dentro da minha máquina, eu posso usar diversos softwares. Por exemplo, o **VirtualBox** que permite que criemos máquinas virtuais dentro do meu sistema operacional. No futuro, usaremos uma máquina virtual em nossa própria máquina, que será executada na Amazon.

Com o hardware configurado, precisamos de um sistema operacional e neste projeto usaremos o Ubuntu. Legal, vou usar o Ubuntu. Eu quero instalar o Java, e o Tomcat, um servidor web dentro dessa máquina. Eu quero também ter um ip pra essa máquina, pra poder acessá-la via web. Tudo isso a gente vai conseguir fazer daqui a pouquinho.

Para criarmos nossa máquina virtual, usaremos Vagrant (http://www.vagrantbox.es/), que é capaz de criar e gerenciar essas máquinas virtuais através do VirtualBox, por exemplo.

O Vagrant será configurado na sua versão 2, e especificares que a que a imagem da máquina é Ubuntu, que por sua vez é chamado de `hashicorp/precise32`, de 32 bits.

```
Vagrant.configure("2") do |config|
    config.vm.box = "hashicorp/precise32"
end
```

Existem diversas imagens que podemos baixar na internet, mas podemos até mesmo criar nossa própria imagem padrão, em um sistema operacional já instalado e configurado. Ainda não baixamos essa imagem do sistema operacional, mas o Vagrant cuidará disso para nós.

Quando utilizamos `config.vm.box = "hashicorp/precise32"`, signifca que **todas** as minhas máquinas virtuais dentro desse arquivo de configuração utilizarão o Ubuntu 32 bits. Criaremos uma máquina chamada `web`:

```
Vagrant.configure("2") do |config|
    config.vm.box = "hashicorp/precise32"

    config.vm.define :web do |web_config|
    end
end
```

Não realizamos nenhuma configuração, apenas definimo uma única nova máquina. Como é que eu executo esse Vagrant file no terminal? – isto é, eu criei um arquivo chamado Vagrant file, configurei o Vagrant para usar o Ubuntu 32, criar uma máquina virtual com o VirtualBox, que é o padrão dele – e então acionamos um `vagrant up` na linha de comando.

O `vagrant up` vai criar a máquina virtual, instalar o Ubuntu e baixar aquela imagem correspondente. Com o Ubuntu instalado, podemos fazer um SSH: `vagrant ssh`, então conseguimos nos conectar com máquina. O Vagrant está sendo executado, e se acionar os comandos `ls`, `whoami`, algo do gênero, perceberemos a lógica Ubuntu. O usuário padrão do `vagrant ssh` e é `vagrant`.

A próxima etapa é instalar o Tomcat. No `ssh`, escreveremos `sudo apt-get update` pra atualizar todos os pacotes do Ubuntu. Assim feito, escreveremos `sudo apt-get install openjdk-7-jre tomcat7`. Estou instalando tanto o jdk quanto o Tomcat, através de `apt-get`. Ao executarmos, notaremos que o Tomcat já está instalado, presente em `/etc/init-d`. Por meio do comando `ps -ef | grep java`, eu vejo que o Tomcat está rodando com o Java 7.

Contudo, precisamos de um ip, caso contrário não conseguiremos realizar testes, pediremos um ip Vagrant, solicitando uma configuração de rede. A rede será fechada (`private_network`). Para nossa máquina web teremos um ip fixo, que será: `192.168.50.10`.

```
Vagrant.configure("2") do |config|
    config.vm.box = "hashicorp/precise32"
    config.vm.define :web do |web_config|
        web_config.vm.network "private_network", ip: "192.168.50.10"
    end
end
```

Estamos em uma rede com a nossa máquina e ela pode ser acessada. Contudo, a máquina já estava rodando e todo o hardware dela já foi inicializado, e não havia esse ip anteriormente. Então eu preciso reiniciar a máquina para que o VirtualBox possa configurar, com o auxílio do Vagrant, o ip dessa máquina. Para tanto, usaremos o comando `vagrant reload`.

O `vagrant reload` para a máquina e a reinicia já com as configurações que solicitamos. Em seguida, o Tomcat é inicializado, porque o pacote `tomcat7` é automático. Dessa forma, poderemos acessar no navegador http://192.168.50.10:8080 (http://192.168.50.10:8080).

Mas todas as vezes que criarmos uma nova máquina, devemos fazer o processo `sudo apt-get` novamente? Temos que decorar tudo isso? Não podemos depender da memória humana para realizar uma configuração básica da máquina, isso deve ser executado automaticamente e não utilizar sempre a linha de comando.

O primeiro passo agora é automatizar o `apt-get update` e `apt-get install openjdk-7-jre tomcat7`.

Para isso, não é o Vagrant que vai nos ajudar. Quem vai configurar e provisionar o software necessário é o Puppet. Depois de destruir a minha máquina, usaremos o comando `vagrant destroy` e aí eu destruo completamente a minha máquina – acabou, a máquina virtual não existe mais no VirtualBox.

Criaremos um arquivo chamado `web.pp`. Esse arquivo `web.pp` vai dentro de um diretório chamado `manifests`. Lá dentro eu crio esse arquivo `web.pp`. Por que `web.pp`? "pp" de Puppet, "web" para ficar o mesmo nome da nossa máquina, pra gente saber que o `web.pp` é da máquina `web`.

Agoar, queremos executar o comando `apt-get update`:

```
exec { "apt-update":
    command => "/usr/bin/apt-get update"
}
```

Daremos um nome para esse `exec`, será `apt-update`. Esse arquivo `web.pp`, quando ele for executado pelo Puppet, irá acionar o comando `apt-get update` para nós. Como executamos o Puppet? Devemos estar como *root*. Portanto: `sudo puppet apply /vagrant/manifests/web.pp`. Mas esse arquivo `web.pp`, eu criei na minha máquina, no meu host.

Onde temos `vagrant file` criaremos um diretório chamado `manifests` e, dentro desse diretório, `web.pp`. Como a nossa máquina virtua,l que está no `vagrant ssh`, acessa esse arquivo?

O Vagrant cria uma *shared folder*m um arquivo compartilhado entre a sua máquina, que é o host, e a máquina virtual. Esse diretório que está compartilhado é o diretório em que está o seu `vagrant file`. Então, se eu quero acessar o arquivo que está em `manifests/web.pp`, declaramos `/vagrant/manifests/web.pp`, por isso `sudo puppet apply /vagrant/manifests/web.pp`.

Quando acionamos o `apply`, eu vejo lá "Opa, qual que foi a mensagem que você deu aqui?" Você executou com sucesso o quê? O `apt update`. Repara que ele referencia para gente um nome: `exec apt-update`, e não o comando em si.

Já que eu salvei esse arquivo, os meus desenvolvedores, sempre que eles quiserem criar máquinas novas, basta que executem `sudo apply` que a máquina vai estar configurada, certo? Ainda não, falta instalar o `openjdk` e o Tomcat. Vamos instalá-los? Então, eu vou falar pro Puppet a instalação do *package* `openjdk-7-jre` e `tomcat7`. Não só isso: gostaríamos também de ter certeza de que eles estão instalados: `ensure => installed`.

Antes de instalar os pacotes, queremos que o comando o comando `Exec["apt-update"]` tenha sido executado:

```
exec { "apt-update":
    command => "/usr/bin/apt-get update"
}
package { ["openjdk-7-jre", "tomcat7"]:
    ensure => installed,
    require => Exec["apt-update"]
}
```

Temos essas duas tarefas: a primeira: `exec { "apt-update"` , e a segunda: `package { ["openjdk-7-jre", "tomcat7"]` . Para executar o arquivo escreveremos: `sudo puppet apply /vagrant/manifests/web.pp` .

Está tudo instalado. Toda vez que quisermos uma máquina nova, basta eu dar `sudo puppet apply /vagrant/manifests/web.pp` , e assim ela é criada e todo o software é instalado para nós, mas como podemos ter certeza disso?

No navegador acessaremos http://192.168.50.10:8080 (http://192.168.50.10:8080) e assim verificamos que o Tomcat está rodando. Mais adiante no curso, aprenderemos a instalar o banco de dados, como configurar sua aplicação e assim por diante.