03

Rodando no cloud

Transcrição

Temos muitos conteúdos prontos, tanto o manifest `vagrantfile`, quanto os manifestos do Puppet, que é o `web.pp`, mas qualquer outro arquivo .pp que precisar de configuração pode se "commitado" no Github, dessa forma, outro desenvolvedor pode acessar os conteúdos para criar a máquina, como em produção.

Estamos criando máquinas virtuais na máquina do desenvolvedor, e não exatamente isso que queremos. É interessante pensarmos em criar uma máquina na nuvem - na Amazon, por exemplo – que tenha todas as características essenciais.

Claro, pra criar uma máquina na Amazon ou num outro cloud qualquer, primeiramente precisamos ter uma conta nestes clouds. O Vagrant possui um plugin para Amazon. Vejamos como podemos criar uma máquinha Amazon EC2 de forma manual.

Teremos de escolher qual será o sistema operacional a ser utilizado. Depois, deveremos definir qual o tamanho da máquina que eu quero: micro, pequena, grande, gigante e assim por diante. Mais adiante, devemos definir qual região do mundo essa máquina poderá ficar disponível. Posteriormente, definiremos qual o nível de segurança que teremos, isto é, quais as portas do *firewall* da Amazon iremos liberar ou não.

Temos um *security_group* `devops-vagrant` que nos comunica que a porta 8080 está aberta assim como a do MySQL. É claro, em produção, você não quer essa porta do MySQL aberta. Adiante, precisaremos de uma chave para acessar o usuário padrão. No Ubuntu, na Amazon, o usuário padrão chama Ubuntu. Então, precisamos de uma chave ssh privada/pública, para poder acessar o usuário Ubuntu na Amazon.

Assim que a chave foi escolhida, a máquina será criada, mas precisaremos que passar as configurações básicas da máquina, assim como a gente fez com o Virtual Box.

Um tempo adiante, a máquina já deve estar rodando, nesse ponto podemos fazer um ssh Primeiramente eu copiaremos o arquivo `.pem`, a chave de segurança do ssh, e podemos gerá-lo: `ssh -i palestra-devops.pem` + o ip da minha máquina na Amazon.

Estamos utilizando o sistema operacional Ubuntu em uma máquina da Amazon, mas poderíamos estar trabalhando com qualquer servidor. No caso da Amazon, especificamente, buscaremos outras metodologias, de forma que não seja necessário ue ficar executando `apt-get update`, `apt-get install`. Usaremos aqueles arquivos do Vegrant.

Uma vez que a máquina na Amazon foi criada,usaremos o plugin do Vagrant para se conectar com a Amazon *webservices*. Para instalar esse plugin, usaremos o comando:

```
vagrant plugin install vagrant-aws
```

O Vagrant possui diversos plugins para utilizarmos, o que usaremos agora é o plugin `aws`, que serve para mudar o `provider` das nossas máquinas. Em vez de usar o provider do Virtual Box, que era a máquina virtual local, agora usaremos provider do `aws`, o provider da Amazon webservices. Instalar o plugin não quer dizer que você está usando-o, ele precisa ser minimamente configurado levando em conta as características do sistema operacional e da máquina,por exemplo: teremos de especificar que o conjunto de chaves que estamos usando são quelas do Devops.

Escolhemos o Ubunto para trabalhar, trata-se de um `ami`, uma imagem de um Ubuntu, e eu tenho que falar essa mesma imagem. Então eu copiei lá do site da Amazon o `ami` no qual eu queria me basear. Tivemos, ainda, que escolher o `security_group` de Devops. Pois queríamos deixar a porta 8080 aberta assim como a do MySQL. Como já foi dito, em produção, provavelmente você vai fechar a porta do MySQL pra ninguém acessar de fora.

Configuraremos o `username`, afinal por padrão o usuário do ssh no Vagrant se chama "vagrant". Não é o caso. Na Amazon, o usuário do ssh se chama "ubuntu". Então, sobrescrevemos este usuário para "ubuntu".

Precisaremos declarar onde está a chave privada, para gerar um ssh nesse endereço. Então, disponibilizaremos a chave privada em `palestra-devops.pem`, usaremos essa chave para acessar a Amazon.

```
config.vm.provider :aws do |aws, override|
    aws.keypair_name = "palestra-devops"
    aws.ami = "ami-358c955c"
    aws.security_groups = ['devops-vagrant']

    override.ssh.username = "ubuntu"
    override.ssh.private_key_path = "palestra-devops.pem"
end
```

Precisamos ainda configurar usuário e senha. Coletaremos os `access_key_id` e `secret_access_key` do site da Amazon e o inseriremos me nosso código. Claro, cada usuário vai ter seu próprio `access_key_id` e seu próprio `secret_access_key`.

Temos de configurar o box uilizado, Você se lembra que a gente estava usando o Ubuntu 32 bits? Aquele Ubuntu 32bits só funcionava pro Virtual Box. Então tremos que trocar. Usaremos um box que não instala nenhum recurso, mas serve muito bem pra Amazon: o `ami` que a gente tinha escolhido Há algumas diferenças neste box. A box que Vegrant define como padrão, `precise32`, já vem com o Puppet instalado. Já a que estamos utilizando agora não vem com nada além do Ubuntu padrão.

Daremos um nome para a nossa máquina para os ajudar na organização. Dentro da definição da maquina web inseriremos uma tag `<name>`, cujo valor será `MusicJungle (vagrant)`:

```
web_config.vm.provider :aws do |aws|
            aws.tags = { 'Name' => 'MusicJungle (vagrant)'}
        end
```

O nome, na verdade, é uma tag que o Amazon `aws` usa para mostrar o nome da sua máquina, no nosso caso, ela chama-se `MusicJungle (vagrant)`. Assim você sabe que máquina é essa, quando ela foi criada e se de fato ela ainda está em uso. Dessa forma organizada, você pode controlar melhor seus gastos, é importante deletar as máquinas do cloud depois que elas não possuem mais uso.

Como nessa nova box não temos o Puppet, devemos instala-lo usando um script em `sh`. Podemos utilizar scripts mais complexos para instar o Puppet, mas faremos esse procedimento no curso, aproveitando a versão compatível com o `deb` atual. É esse script `bash` que faz isso. Aí você fala:

Estamos instalando o Puppet com `bash`, um arquivo de script. Não podemos depender do Puppet para instalar ele mesmo, obviamente. Depois que o Puppet está instalado, iremos adicionar todos os outros recursos que precisamos: Tomcat, instala Java, o que você precisar.

Vamos configurar o Puppet. Acasseremos o local em que usávamos o `provision` do Puppet e antes de fazer o provision do Puppet, faremos um provision do shell, isto é, o arquivo bash que acabamos de utilizar.

Em outras palavras estamos solicitando o provision do tipo `shell`, que é um bash simples, e o `path` desse arquivo deverá ser `manifests/bootstrap.sh`. Esse é o nome que demos para o arquivo, `bootstrap`, porque ele configura o `boot`. Agora, dois `provision`. O primeiro provision, do tipo shell, chama o arquivo `bootstrap.sh`. O segundo provision chama o `puppet` e instala o `web.pp`. Ele vai executar nessa ordem: primeiro o `bootstrap.sh`, depois o `puppet` com o `web.pp`.

```
config.vm.define :web do |web_config|
        web_config.vm.network "private_network", ip: "192.168.50.10"
        web_config.vm.provision "shell", path: "manifests/bootstrap.sh"
        web_config.vm.provision "puppet" do |puppet|
            puppet.manifest_file = "web.pp"
        end
        web_config.vm.provider :aws do |aws|
            aws.tags = { 'Name' => 'MusicJungle (vagrant)'}
        end
    end
```

Temos a questão do idempotente, mencionada anteriormente. Utilizando o script de `bootstrap.sh`, veremos que ele tem a condicional if: se o Puppet já está instalado, ele não o será novamente, apenas será executado. Trata-se de um script padrão que já está preparado para essa circunstância. Existem outros provisions, outras ferramentas, que também servem, assim como o Puppet, ou assim mesmo como o `shell`, para instalar e configurar o seu ambiente.

Neste ponto, instalamos o Puppet, e configuramos a máquina da Amazon.

Se a máquina já está rodando, podemos apenas acionar o `vagrant provision`, que serve executar o provision. Existem outras opções disponíveis para realizar esse caso, mas o `vagrant provision` é capaz de forçar uma execução de provision.

O provision é executado tanto no `shell` quanto no `puppet` na máquina da Amazon.

Na Amazon a máquina está rodando, podemos copiar o ip da máquina no navegador na porta 8080, e acessar o servidor. Podemos, até mesmo, acessar o Music Jungle.

Podemos utilizar o `vagrant ssh` e ele vai fazer o ssh pra máquina remota, então, via MySQL, verificamos que o usuário foi adicionado na máquina da Amazon.

Até então para executar a máquina seguimos os seguintes procedimentos: fizemos a configuração de um box da Amazon, inserimos a configuração de `aws`, isto é, a `ami`, o nome da máquina e depois instalamos o Puppet com um provision.

Lembrando: o `vagrant up` "evanta" por padrão o provider do Virtual Box. Se eu quero forçar o provider do `aws`, isso é, se eu quero usar o plugin do `aws`, temos que utilizar `vagrant up --provider=aws`. Isto é, devemos específicar para o `vagrant up` qual é o provider que deve ser utilizado. Com o `vagrant provision` podemos executar o o `puppet apply`, ou o `shell`.Dessa forma, somoso capzes de destruir a máquina e subir uma máquina totalmente nova do zero com tudo configurado em 6 minutos.

Se for do seu interesse agilizar ainda mais todo o processo, podemos até mesmo criar a própria box, graças ao `vagrant package` um novo arquivo .box é criado. Não faremos desse modo, mas é possível explorar outras áreas: ao invés do Puppet você pode conhecer outras ferramentas, usar o Vagrant com outros providers e assim acelerar ainda mais o processo de criação da máquina, usando sua própria imagem.

O que somos capazes de fazer agora? Usar o Vagrant e usar o Puppet. A gente é capaz de dar *up, destroy, reload, provision* em máquinas virtuais, seja aqui, seja lá na Amazon, seja no provider que for.

Somos capazes, ainda, de usar o Puppet pra fazer *apply* e *reapply* diversas vezes dos meus scripts de `provision` ; fazer o *networking* dessas máquinas; criar *shared folders*; executar comandos aleatórios no Puppet com o `exec` ; instalar pacotes, com `package` , e configurar serviços com `service` .

Podemos também usar para provision tanto o `shell` quanto o `puppet` , e com o Puppet criar arquivos, definir nossas próprias funções recebendo variáveis como parâmetro. Com o vagrant, podemos usar o provider do Virtual Box e do `aws` .

Somos capazes de promover novas soluções desenvolvendo esse ambiente de produção na máquina e fazer testes *end-to-end* de forma local, com o ambiente igual ao de produção. Se tivermos algum problema ao longo do tempo, acessaremos oGithub, no seu arquivo `vagrantfile` e analisamos vê como a máquina estava, confere as diferenças, testa de novamente. Você consegue voltar no tempo à medida que você commita esses arquivos no Github.

Você consegue criar várias máquinas, a gente definiu só uma `define :web` , mas há mais possibilidades. Cada uma dessas máquinas é capaz de receber variáveis e de trabalhar com ambientes diferentes. O Puppet vai muito mais além na prática de gerenciar as configurações, isto é, *configuration management*.

A gente tem que automatizar todo esse processo de criação da máquina e, depois, além da criação, da monitoração, ou seja, aspectos de segurança ou de usabilidade, de carga, ou seja lá do que for. Fazemos isso por meio do processo de Devops, coletando operações e trabalhando com desenvolvimento.

Vamos aos exercícios?