Jose Luiz Bento Da Silva / luizbsilva@gmail.com

4LINUX
OPEN SOFTWARE SPECIALISTS

CURSO 525

INFRAESTRUTURA ÁGIL COM PRÁTICAS DEVOPS

4

Jose Luiz Bento Da Silva / luizbsilva@gmail.com

Desenvolvimento de ambientes virtuais com Hashicorp Vagrant

Conceitos de Virtualização

Anotações

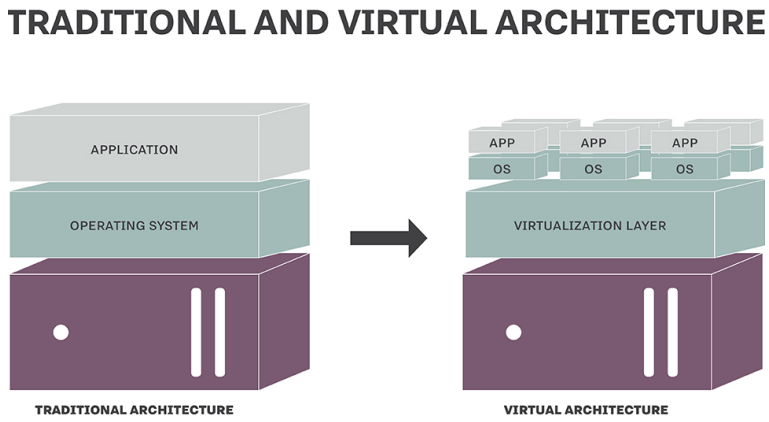
Ementa do treinamento

1. O que é virtualização
2. Entendendo o Vagrant
3. Instalando o Vagrant

Anotações

O que é Virtualização

Virtualização é o processo de criar uma versão virtual de algo, por exemplo uma rede de servidores, sistemas operacionais, dispositivos de armazenamento, entre outros.



Antigamente, existia muito o costume de comprar servidores para hospedagem de algum serviço, e ás vezes o serviço não precisava de muito recurso. Porém, por conta dele estar instalado nesse servidor, você não pode mais instalar outros serviços que usem a mesma porta, você perdeu um espaço no datacenter para um serviço importante, porém que pouco usa recurso e ainda precisa pagar altíssimas contas de energia, somente para manter o serviço funcionando.

A virtualização veio para solucionar esse problema: com ela, podemos repartir os recursos de nossa máquina física em diversas máquinas virtuais, cada uma com a quantidade necessária para o serviço que possui funcionar. Por conta de sua natureza, máquinas virtuais são isoladas, só precisando se comunicar com o virtualizador, sobre o qual elas foram criadas.

Benefícios da Virtualização

Gerenciamento de Recurso

Agora, podemos gerenciar o quanto de recurso computacional cada serviço irá consumir, considerando que cada máquina virtual manterá um serviço rodando.

Segurança

Máquinas virtuais são seguras devido a isolação delas: não conversam em nenhum momento com a máquina hospedeira, exceto para operações do hardware virtual.

Automação

Atualmente, existem diversas ferramentas que permitem criar infraestruturas virtualizadas em questão de segundos.

Anotações

O que é o Vagrant

Vagrant é uma ferramenta de automação escrita em Ruby e mantida pela HashiCorp, a mesma empresa responsável por outras ferramentas como o Terraform, Vault e o Nomad.
Através de um script, é possível subir uma infraestrutura inteira e até mesmo configurar suas máquinas.

Vagrant possui uma linguagem própria para os scripts baseados em Ruby, e permite descrever todos os aspectos de uma máquina virtual, como hostname, ip, cpu, memória, disco, sistema operacional, quantidade de máquinas que serão criadas, script que deve ser executado após o término do boot, entre outros aspectos.
Toda máquina criada pelo Vagrant é feita através de uma Vagrantbox, que é uma imagem base com todos os pacotes que uma máquina virtual deve sempre ter quando for criada. Muita atenção na box que você possui: ela só funciona no virtualizador no qual ela foi criado, e nenhum outro mais.

Provedor x Provisionador

O Vagrant trabalha com dois papéis distintos:

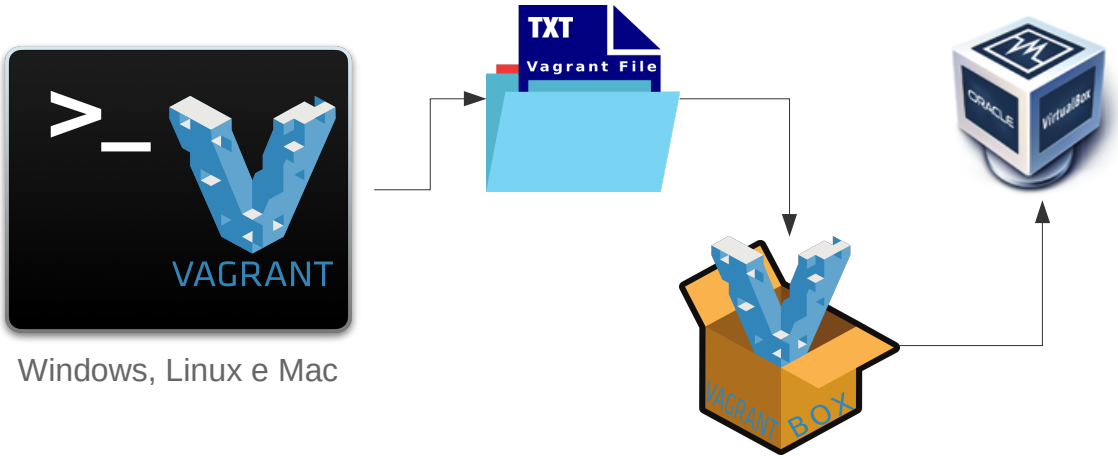
Provedor	O provedor **(provider)** é o responsável pela criação da instancia dos ambientes, geralmente uma máquina virtual, podendo também ser uma instância em alguma Cloud.
Provisionador	O provisionador **(provisioner)** é o responsável pelas tarefas a serem executadas de forma automatizada, como por exemplo instalação de pacotes e configuração do sistema.

Exemplos de Providers: Virtualbox, Hyper-V, VMWare e Docker;
Exemplos de Provisioners: Ansible, Chef, CFEngine, Puppet, Shell, Salt e Docker.

O Docker entra tanto no papel de Provisioner quanto de Provider. Como provisioner, ele pode instalar o Docker, fazer o download de contêineres e configurar os mesmos. Como Provider, o mesmo pode efetuar a implantação da sua infraestrutura em contêineres.

Entendendo o Vagrant

Fluxo de criação de uma máquina virtual:



Windows, Linux e Mac

A construção de uma máquina virtual do Vagrant, se inicia baixando uma imagem base de dentro do Vagrant Cloud, um repositório de box do Vagrant. Uma vez a imagem baixada, será acessado o virtualizador em questão para começar a construção dessa máquina virtual, informando que o disco se trata da imagem base que baixamos agora pouco. Por padrão, o Vagrant possui suporte a diversos virtualizadores, como o Virtualbox, a AWS, Vmware Workstation, entre outros. Porém, é necessário ficar de olho se a imagem que você deseja usar possui suporte aos demais virtualizadores.

Com a máquina criada, o ultimo passo é começar a executar o script de configuração, caso você tenha informado algum, caso contrário, basta se logar e começar a utilizar.

Instalação do Vagrant no Linux

O processo de instalação do Vagrant é bem simples: basta baixar o instalador correspondente ao seu sistema operacional, instalar com seu gerenciador de pacotes, e então validar se tudo está funcionando.

1. Baixe o instalador do Vagrant:

```
# wget https://releases.hashicorp.com/vagrant/2.2.5/vagrant_2.2.5_x86_64.deb
```

2. Instale o Vagrant através do dpkg:

```
# dpkg -i vagrant_2.2.5_x86_64.deb
```

3. Valide se o processo de instalação ocorreu com sucesso:

```
# vagrant --version
```

Anotações

Recapitulando

1. O que é virtualização
2. Entendendo o Vagrant
3. Instalando o Vagrant

Anotações

Desenvolvimento de ambientes virtuais com Hashicorp vagrant

Definição de máquinas virtuais no Vagrant

Anotações

Objetivos da aula

1. Entendendo a estrutura de um Vagrantfile
2. Criando uma Instância
3. Criando Múltiplas Instâncias
4. Configurando o endereço IP

Anotações

Entendendo o Vagrantfile

Vagrantfile é o nome do arquivo que contém instruções do que o Vagrant deve fazer para criar uma máquina virtual.

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provider "virtualbox" do |vb|
       vb.memory = "1024"
       vb.cpus = "2"
     end
end
```

O Vagrantfile é um arquivo que irá conter todas as instruções de criação de sua instância, como a quantidade de memória, cpu, qual rede ela estará, imagem a ser utilizada, entre outras opções.

O arquivo deverá se chamar realmente Vagrantfile, pois esse é o arquivo que o comando vagrant tentará localizar no diretório atual, para que possa saber o que deve construir e como deve construir.

No exemplo acima, criamos uma instância usando nossa imagem do centos importada anteriormente, com 1024MB de memória e 2 vCPUs.

Entendendo o Vagrantfile

Toda estrutura de um Vagrantfile começa informando a versão que vai usar do configfile e termina com a palavra end.

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provider "virtualbox" do |vb|
       vb.memory = "1024"
       vb.cpus = "2"
     end
end
```

O Vagrantfile é baseado em Ruby, por isso segue as mesmas regras de sintaxe dessa linguagem de programação.

Uma dessas regras é fechar uma estrutura com a palavra **end.** Dessa forma, ele informa a linguagem de programação, de que tudo que for digitado fora desse **end,** não faz parte da estrutura que você criou.

Entendendo o Vagrantfile

O próximo passo é informar qual a imagem que iremos utilizar nessa instância, nesse caso usaremos a imagem chamada **centos/7**.

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provider "virtualboxv do |vb|
       vb.memory = "1024"
       vb.cpus = "2"
     end
end
```

Anotações

Entendendo o Vagrantfile

Por ultimo, criamos uma estrutura onde informamos qual é o provedor de nossa instância e suas configurações (cpu,mem,hostname…).

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provider "virtualbox" do |vb|
        vb.memory = "1024"
        vb.cpus = "2"
      end
end
```

Anotações

Criando uma Instância

Vamos criar uma instância com 1GB de memória e 2 vCPUs, igual no exemplo anterior. O Vagrant permite que você gere o arquivo usando a opção init, porém você também pode criar o arquivo, contanto que respeite o nome.

1. Gere um arquivo do Vagrantfile com o mínimo de configuração:
```
# vagrant init -m centos/7
```

2. Edite o arquivo para ficar igual ao exemplo anterior:
```
# vim Vagrantfile
```

3. Construa seu ambiente de acordo com o que se encontra no Vagrantfile:
```
# vagrant up
```

Somente de executar o comando vagrant init, será criado um arquivo com diversas opções já preenchidas, porém comentadas.

Para quem está trabalhando pela primeira vez com ela, talvez seja a melhor opção, caso contrário use a opção -m ou –minimal para remover todas essas linhas e deixar somente o necessário.

Durante a criação do Vagrantfile, é possível informar qual imagem que será utilizada, assim o arquivo já é automaticamente construído com esse valor preenchido.

Por padrão, a variável **config.vm.box** vem com o valor “base”, que significa nada, sendo necessário alterar para o nome da imagem que deseja utilizar, por exemplo “ubuntu/xenial64”.

Criando uma Instância

Para validar se a criação da instância ocorreu com sucesso, basta executar a opção ssh para acessá-la.

1 Acessando sua instância via ssh:

```
# vagrant ssh
# exit
```

Também podemos ver todas as máquinas que temos, através da opção global-status.

2 Listar as máquinas, sendo utilizadas pelo vagrant:

```
# vagrant global-status
```

Para desligar a instância, basta executar a opção halt.

3 Desligue a instância criada anteriormente:

```
# vagrant halt
```

vagrant ssh – Efetua a conexão ssh com a máquina descrita no Vagrantfile, é possível especificar o nome da máquina ou o id da mesma, que pode ser verificado com o comando vagrant global-status;

vagrant halt – Efetua o desligamento das máquinas descritas no Vagrantfile, é possível especificar o nome da máquina ou o id da mesma, que pode ser verificado com o comando vagrant global-status.

Criando uma Instância

Para deletar a instância, use a opção "destroy".

Atenção: A opção destroy é irreversível, e tudo relacionado aquele ambiente que você criou será excluído, com exceção do conteúdo que está dentro do diretório do seu Vagrantfile.

1 Destrua a instância que criou anteriormente:

```
# vagrant destroy
```

Anotações

Criando Múltiplas Instâncias

Quando trabalhamos com Vagrant, não criamos um arquivo por instância, e sim descrevemos um ambiente todo dentro dele, assim iremos adicionar múltiplas instâncias no nosso Vagrantfile.

Anotações

Obtendo arquivos do Curso

Vamos baixar do Git da 4Linux os arquivos do curso Infraestrutura Ágil com práticas DevOps. Em sistemas operacionais Linux e Mac é só utilizar o comando git. Se você utiliza Windows baixe e instale o Notepad ++ e o Git nos seguintes endereços:
https://notepad-plus-plus.org/downloads/
https://git-scm.com/download/win

1. Crie uma pasta de nome "infraagil" e acesse a mesma:

```
# mkdir infraagil
# cd infraagil
```

2. Em seguida baixe os arquivo do curso através do seguinte comando:

```
# git clone https://github.com/4linux/4525.git
```

Anotações

Criando Múltiplas instancias

Vamos editar nosso Vagrantfile para adicionar mais uma instância ao mesmo:

1

Gere o arquivo Vagrantfile utilizando o modelo Vagrantfile_11 da pasta Aula 2.2:

```
# cp 4525/arquivos/Aula\ 2.2/Vagrantfile_11 Vagrantfile
# cat Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.define "server1" do |server1|
    server1.vm.box = "centos/7"
  end
  config.vm.define "server2" do |server2|
  server2.vm.box = "ubuntu/bionic64"
  end
end
```

Anotações

Se você utiliza sistema Windows substitua o comando:

cp 4525/arquivos/Aula\ 2.2/Vagrantfile_11 Vagrantfile

Por

PS C:\> cp '.\4525\arquivos\Aula 2.2\Vagrantfile_11' Vagrantfile

Criando Múltiplas Instâncias

Podemos agora subir nossa instância:

1 Construa o ambiente de acordo com o Vagrantfile:

```
# vagrant up
```

Também podemos ver todas as máquinas que temos, através da opção global-status.

2 Listar as máquinas, sendo utilizadas pelo vagrant:

```
# vagrant global-status
```

Podemos nos conectar as instâncias, passando o nome que foi definido no campo config.vm.define:

3 Conecte-se a instância server1:

```
# vagrant ssh server1
```

Anotações

Criando Múltiplas Instâncias

1 Verifique que o sistema operacional da instancia server1 é um CentOS:

```
# cat /etc/*release
```

Desconecte da máquina server1 e conecte-se à máquina server2:

2

```
# exit
# vagrant ssh server2
```

3 Verifique que o sistema operacional da instancia server2 é um Ubuntu:

```
# cat /etc/*release
```

Desconecte da máquina server2 e destrua o ambiente:

4

```
# exit
# vagrant destroy
```

Anotações

Configurando endereço IP

Podemos também definir o endereço IP de cada instância no Vagrantfile, utilizando o parâmetro **vm.network** para configurar uma rede privada entre as máquinas e informar o endereço IP.

Anotações

Configurando endereço IP

Para adicionarmos o endereço IP as instâncias, basta editarmos o arquivo Vagrantfile para incluir as linhas abaixo. Gere o arquivo Vagrantfile utilizando o modelo Vagrantfile_15 da pasta Aula 2.2

```
Vagrant.configure("2") do |config|
  config.vm.define "server1" do |server1|
    server1.vm.box = "centos/7"
    server1.vm.network "private_network", ip: "10.5.25.10"
  end
 config.vm.define "server2" do |server2|
    server2.vm.box = "ubuntu/bionic64"
    server2.vm.network "private_network", ip: "10.5.25.20"
  end
end
```

O parâmetro **.vm.network** configura a rede da máquina descrita, podemos configurar por exemplo encaminhamento de portas, redes privadas, redes públicas, IP estático ou DHCP, dentre várias outras funcionalidades.

Configurando endereço IP

Execute o script para verificar se as instâncias foram criadas:

1 Copie o arquivo modelo e inicie suas instâncias do Vagrant:

```
# cp 4525/arquivos/Aula\ 2.2/Vagrantfile_15 Vagrantfile
# vagrant up
```

Conecte-se na server1 e execute o comando ping para validar a configuração de rede:

2

```
# vagrant ssh server1
# ping -c4 10.5.25.20
```

Por ultimo, desconecte e destrua as instâncias para que possamos começar a construção de nosso ambiente:

3 Force a destruição da instâncias através da flag "-f":

```
# exit
# vagrant destroy -f
```

Anotações

Recapitulando

1. Entendendo a estrutura de um Vagrantfile
2. Criando uma Instância
3. Criando Múltiplas Instâncias
4. Configurando o endereço IP

Anotações

Desenvolvimento de ambientes virtuais com Hashicorp vagrant

Variáveis, Loops e Condições no Vagrant.

Anotações

Objetivos da aula

1. Trabalhando com variáveis
2. Trabalhando com laços de repetição (loops)
3. Trabalhando com Condições

Anotações

Trabalhando com Variáveis

Podemos utilizar variáveis no nosso Vagrantfile para parametrizar nosso ambiente, isso faz com que o código possa ser reaproveitado para outros ambientes, sendo possível mudar apenas algumas linhas e ter uma infraestrutura completamente nova criada.

Anotações

Trabalhando com Variáveis

Vamos editar nosso Vagrantfile para adicionar algumas variáveis:

1 Acesse a pasta infraagil e abra o Vagrantfile para edição:

```
# cd infraagil
# vim Vagrantfile
```

2 Adicione as linhas no topo do arquivo para definir as variáveis que iremos utilizar:

```
NETWORK = "10.5.25."
BOX_SRV1 = "centos/7"
BOX_SRV2 = "ubuntu/bionic64"
```

Anotações

Trabalhando com Variáveis

Edite os seguintes campos no Vagrant file para utilizar as variáveis criadas:

```
Vagrant.configure("2") do |config|
  config.vm.define "server1" do |server1|
✔   server1.vm.box = BOX_SRV1
    server1.vm.network "private_network", ip: NETWORK+"10"
  end
 config.vm.define "server2" do |server2|
✔   server2.vm.box = BOX_SRV2
    server2.vm.network "private_network", ip: NETWORK+"20"
  end
end
```

Para utilização da variável, basta inseri-la no código, caso seja necessário uma concatenação da variável como é no caso do ip, é preciso que a variável seja informada seguida de um sinal de soma e o valor que deve ser concatenado entre aspas, como por exemplo:

NETWORK+"20" = "10.5.25.20" , uma vez que NETWORK = 10.5.25.

Trabalhando com Variáveis

Vamos agora verificar as configurações:

1 Iniciando as instâncias:

```
# vagrant up
```

Conecte-se na máquina server1 e verifique sua versão do S.O. e seu endereço IP:

2

```
# vagrant ssh server1
# cat /etc/*release
# ip -c a
```

Desconecte da instância:

3

```
# exit
```

Anotações

Trabalhando com Variáveis

Repita os mesmos passos para o server2:

1
```
# vagrant ssh server2
# cat /etc/*release
# ip -c a
```

Desconecte da instância e destrua o ambiente:

2
```
# exit
# vagrant destroy -f
```

Anotações

Trabalhando com Laços de Repetição

Quando trabalhamos com Vagrant, não criamos um arquivo por instância, e sim descrevemos um ambiente todo dentro dele. Para esse fim, usamos laços de repetição e criamos todas as máquinas que foram especificadas dentro do arquivo.

Anotações

Trabalhando com Laços de Repetição

Vamos primeiramente criar nosso arquivo Vagrantfile e colocar as linhas abaixo para definir nossas máquinas virtuais:

1

Gere o arquivo Vagrantfile utilizando o modelo Vagrantfile_9 da pasta Aula 2.3:

```
# cp 4525/arquivos/Aula\ 2.3/Vagrantfile_9 Vagrantfile
# cat Vagrantfile

   machines = {
   "compliance"=> { "image" => "ubuntu/bionic64" },
   "container"=> { "image" => "fedora/29-atomic-host" },
   "scm"=> { "image" => "debian/buster64" },
   "log"=> { "image" => "ubuntu/bionic64" },
   "automation"=> { "image" => "centos/7" },
   }
```

Anotações

Trabalhando com Laços de Repetição

Para usarmos esse bloco **machines**, basta editarmos o arquivo Vagrantfile para incluir as linhas abaixo:

```
Vagrant.configure("2") do |config|
  machines.each do |name,conf|
    config.vm.define "#{name}" do |machine|
       machine.vm.box = "#{conf["image"]}"
       machine.vm.hostname = "#{name}.4labs.example"
     end
  end
end
```

Para acessar as variáveis em um array, é necessário que ela esteja no formato **#{VARIAVEL}**, onde a hash identifica que trata-se de um array e as chaves delimitam o campo da variável.

A definição das máquinas é feito na linha **machines.each do |name,conf|**, onde é realizada a leitura do bloco machines, sendo o primeiro campo informado como nome e os demais campos são formados com o array chave e valor, sendo chamado como **#{conf[valor]}**.

O laço de repetição ocorre quando acessamos o método each, passando o conteúdo de cada iteração para a variável **machine**. Basicamente, a cada repetição um item de sua lista será salvo dentro da variável machine, e então um trecho de código será repetido. No nosso caso, todas as instruções de configuração de servidor ficarão dentro do laço, assim iremos criar nossas instâncias baseadas nesse conteúdo.

Por se tratar de um array de chave e valor, o conteúdo da variável machine para termos acesso ao valor, precisamos informar uma chave, no caso informamos image como nossa chave para que então tenhamos acesso ao valor “**centos**” , “**ubuntu**” , etc…

O hostname de cada máquina também é configurado através do parametro **machine.vm.hostname**, onde o ele concatena a chave **name** com **.4labs.example** para configurar o hostname.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

machines = {
  "compliance" => {"memory" => "2048", "cpu" => "2", "ip" => "20", "image" => "ubuntu/bionic64"},
  "container" => {"memory" => "2048", "cpu" => "1", "ip" => "30", "image" => "fedora/29-atomic-host"},
  "scm" => {"memory" => "256", "cpu" => "1", "ip" => "40", "image" => "debian/buster64"},
  "log" => {"memory" => "1024", "cpu" => "1", "ip" => "50", "image" => "ubuntu/bionic64"},
  "automation" => {"memory" => "3072", "cpu" => "2", "ip" => "10", "image" => "centos/7"}
}

Vagrant.configure("2") do |config|

  config.vm.box_check_update = false
  machines.each do |name, conf|
    config.vm.define "#{name}" do |machine|
      machine.vm.box = "#{conf["image"]}"
      machine.vm.hostname = "#{name}.4labs.example"
      machine.vm.network "private_network", ip: "10.5.25.#{conf["ip"]}"
      machine.vm.provider "virtualbox" do |vb|
        vb.name = "#{name}"
        vb.memory = conf["memory"]
        vb.cpus = conf["cpu"]
        vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
    end
  end
end
```

Trabalhando com Laços de Repetição

Por ultimo, execute o script para vermos se a instância foi criada com sucesso.

1 Inicie suas instâncias do Vagrant:

```
# vagrant up
```

Valide se todas elas iniciaram corretamente.

2 Verifique o status das máquinas:

```
# vagrant status
```

Por ultimo, destrua as instâncias para começar a construção de nosso ambiente.

3 Destrua as instâncias:

```
# vagrant destroy -f
```

Anotações

Trabalhando com Condições

Como o Vagrantfile é escrito em ruby, também é possível que sejam configuradas condições utilizando um bloco de **IF** para realizar um teste e executar uma ação.

Anotações

Trabalhando com Condições

Vamos editar nosso Vagrantfile para adicionar uma condição no nosso loop, se a máquina for a automation, iremos adicionar mais um disco de 20GiB, para isto precisaremos criar o disco e verificar se o mesmo já não existe, a fim de não duplicar o mesmo.

1 Gere o arquivo Vagrantfile utilizando o modelo Vagrantfile_15 da pasta Aula 2.3:

```
# cp 4525/arquivos/Aula\ 2.3/Vagrantfile_15 Vagrantfile
# cat Vagrantfile
```

Anotações

Trabalhando com Condições

Edite o arquivo, adicionando as seguintes linhas após **vb.customize....**

```
vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
  if name == "automation" and not File.file?('iscsi525.vdi')
    vb.customize ['createhd', '--filename', 'iscsi525.vdi',
'--size', 20 * 1024]
     (...)
  end
end
```

==========

```
if name == "automation" and not File.file?('iscsi525.vdi')
  vb.customize ['createhd', '--filename', 'iscsi525.vdi', '--size', 20 * 1024]
  vb.customize ['storagectl', :id, '--name', 'SATA Controller', '--add', 'sata']
   vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1, '--device', 0, '--type',
'hdd', '--medium', 'iscsi525.vdi']
end
```

==========

A declaração if verifica se o arquivo iscsi525.vdi existe, se o mesmo não existir o arquivo é criado, em seguida é adicionada uma controladora SATA à máquina e o disco é anexado na mesma.

Este tipo de declaração é necessária, uma vez que se o Vagrantfile for executado diversas vezes, o arquivo seria reescrito, apagando assim seu conteúdo interno.

Trabalhando com Condições

Por último, execute o script para vermos se a instância foi criada com sucesso.

1 Inicie suas instâncias do Vagrant:

```
# vagrant up
```

Valide se todas elas iniciaram corretamente.

2 Verifique o status das máquinas:

```
# vagrant status
```

Por ultimo, destrua as instâncias para começar a construção de nosso ambiente.

3 Destrua as instâncias:

```
# vagrant destroy -f
```

Anotações

Recapitulando

1. Trabalhando com variáveis
2. Trabalhando com laços de repetição (loops)
3. Trabalhando com Condições

Anotações

Desenvolvimento de ambientes virtuais com Hashicorp vagrant

Provisionadores

Anotações

Objetivos da aula

1. O que são provisionadores
2. Tipos de Provisionadores
3. Utilizando Provisionadores
4. Construindo o Ambiente

Anotações

O que são Provisionadores

Provisionadores no vagrant possibilitam a instalação automática de software, alteração de configuração e arquivos na máquina como parte do processo do vagrant up.

Anotações

Tipos de Provisionadores

Existem vários tipos de provisionadores para o Vagrant, dentre eles podemos citar:

File Shell
Ansible CFEngine
Chef Docker
Puppet Salt

Anotações

Utilizando Provisionadores

Para a utilização dos provisionadores, efetuamos a chamada do mesmo através do parametro **.vm.provision**, seguido do nome do provisionador:

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provision "shell",
   (...)
   end
end
```

Anotações

Utilizando Provisionadores

Para utilização do provisionador file, podemos utilizar como parâmetros a origem e o destino dos arquivos:

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provision "file",
    source: "~/files/hosts",
    destination: "/etc/hosts"
  end
end
```

Para a utilização do módulo file, devemos informar a origem na máquina hospedeira e o destino na máquina criada com o vagrant.

Utilizando Provisionadores

Para utilização do provisionador shell, podemos informar qual será o comando executado através do parâmetro **inline**:

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
  config.vm.provision "shell",
     inline: "mkdir -p /srv/app"
   end
end
```

Através do parâmetro inline, podemos adicionar os comandos diretamente no vagrantfile, como por exemplo executar a criação da estrutura de diretórios /srv/app.

Utilizando Provisionadores

Também podemos declarar um script no topo do Vagrantfile e chamá-lo via **inline**:

```
$script = <<-SCRIPT
mkdir -p /srv/app
touch /srv/app/file1
SCRIPT

Vagrant.configure("2") do |config|
  config.vm.box = "centos"
  config.vm.provision "shell", inline: $script
   end
end
```

A definição do script no topo do Vagrantfile faz com que a manutenção e edição do arquivo seja feita de forma rápida, deixando o arquivo mais organizado.

Como um script em shell, as tarefas são executadas linha a linha.

Utilizando Provisionadores

Outra maneira de chamar um script é através do parametro **path**, isto fará com que seja executado o script que se encontra na máquina local:

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos"
  config.vm.provision "shell",
    path: "script.sh"
   end
end
```

Podemos também informar a origem na máquina hospedeira de um script, para ser executado através do parâmetro path.

Utilizando Provisionadores

O parâmetro **path** aceita também o script, através de uma URL:

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos"
  config.vm.provision "shell",
    path: "https://www.4labs.example/provision.sh"
   end
end
```

Anotações

Utilizando Provisionadores

Podemos também descrever nossa automação, utilizando Ansible e Puppet que são assuntos deste curso, para tal precisamos primeiramente conhecer a ferramenta. Mais informações sobre estes módulos podem ser encontrados em:

https://www.vagrantup.com/docs/provisioning/

Anotações

Construindo o Ambiente

Vamos começar a construir nosso ambiente. Para as aulas seguintes, utilizaremos 5 máquinas – compliance, container, scm, log e automation. Para isso, iremos alterar o Vagrantfile para descrever todo nosso ambiente, como CPU, memória, imagem, hostname, rede, entre outros.

Anotações

Construindo o Ambiente

Abra o arquivo Vagrantfile e crie diversas variáveis para descrever os aspectos de cada uma dessas instâncias (rede, imagem, cpu, memória e hostname). Altere também como o Vagrantfile irá trabalhar com essas informações.

1 Acesse a pasta infraagil para iniciar a aula:

```
# cd infraagil
```

2 Gere o arquivo Vagrantfile utilizando o modelo da pasta Aula 2.4:

```
# cp 4525/arquivos/Aula\ 2.4/Vagrantfile Vagrantfile
# cat Vagrantfile
```

Anotações

=== Vagrantfile ===

```
# -*- mode: ruby -*-
# vi: set ft=ruby  :

machines = {
  "compliance" => {"memory" => "2048", "cpu" => "2", "ip" => "20", "image" => "ubuntu/bionic64"},
  "container" => {"memory" => "2048", "cpu" => "1", "ip" => "30", "image" => "centos/7"},
  "scm" => {"memory" => "512", "cpu" => "1", "ip" => "40", "image" => "ubuntu/bionic64"},
  "log" => {"memory" => "2048", "cpu" => "1", "ip" => "50", "image" => "ubuntu/bionic64"},
  "automation" => {"memory" => "3072", "cpu" => "2", "ip" => "10", "image" => "centos/7"}
}

Vagrant.configure("2") do |config|

  config.vm.box_check_update = false
  machines.each do |name, conf|
    config.vm.define "#{name}" do |machine|
      machine.vm.box = "#{conf["image"]}"
      machine.vm.hostname = "#{name}.4labs.example"
      machine.vm.network "private_network", ip: "10.5.25.#{conf["ip"]}"
      machine.vm.provider "virtualbox" do |vb|
        vb.name = "#{name}"
        vb.memory = conf["memory"]
        vb.cpus = conf["cpu"]
        vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
        if name == "automation" and not File.file?('iscsi525.vdi')
          vb.customize ['createhd', '--filename', 'iscsi525.vdi', '--size', 20 * 1024]
          vb.customize ['storagectl', :id, '--name', 'SATA Controller', '--add', 'sata']
           vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1, '--device', 0, '--type', 'hdd',
     '--medium', 'iscsi525.vdi']
        end
      end
        if "#{conf["image"]}" == "ubuntu/bionic64"
          machine.vm.provision "shell", inline: "apt install python -y"
        end
    end
  end
  config.vm.provision "shell", path: "script.sh"
end
```

Construindo o Ambiente

Crie também o script.sh que será responsável pela garantia das chaves de acesso e população do arquivo */etc/hosts:*

1 Gere o arquivo script.sh utilizando o modelo da pasta Aula 2.4:

```
# cp 4525/arquivos/Aula\ 2.4/script.sh script.sh
# cat script.sh
```

Anotações

==== script.sh ====

```
# Garantindo as chaves
KEY_PATH='/vagrant/files'

mkdir -p /root/.ssh
cp $KEY_PATH/key /root/.ssh/id_rsa
cp $KEY_PATH/key.pub /root/.ssh/id_rsa.pub
cp $KEY_PATH/key.pub /root/.ssh/authorized_keys
chmod 400 /root/.ssh/id_rsa*
cat /root/.ssh/id_rsa.pub >> /home/vagrant/.ssh/authorized_keys
cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys

# Garantindo os hosts
HOSTS=$(head -n7 /etc/hosts)
echo -e "$HOSTS" > /etc/hosts
echo '10.5.25.10 automation.4labs.example' >> /etc/hosts
echo '10.5.25.20 compliance.4labs.example' >> /etc/hosts
echo '10.5.25.30 container.4labs.example' >> /etc/hosts
echo '10.5.25.40 scm.4labs.example' >> /etc/hosts
echo '10.5.25.50 log.4labs.example' >> /etc/hosts
```

Trabalhando com Laços de Repetição

Agora, vamos criar o diretório para guardar as chaves de acesso:

1 Criando o diretório:

```
# mkdir files
```

Criaremos as chaves de acesso e armazenaremos dentro do diretório criado, para que as mesmas sejam enviadas para as máquinas virtuais.

2 Gerando o par de chaves de acesso:

```
# ssh-keygen -f $PWD/files/key
    <ENTER>
    <ENTER>
```

Anotações

Trabalhando com Laços de Repetição

Por último, execute o script para vermos se a instância foi criada com sucesso.

1 Inicie suas instâncias do Vagrant:

```
# vagrant up
```

Por termos múltiplas instâncias do Vagrant iniciadas, será necessário informar o nome dela para que possamos acessar via ssh. Execute novamente o vagrant ssh, só que dessa vez informe o nome de uma delas para que possamos conferir se a mesma foi criada com sucesso.

2 Conectando ssh em uma máquina vagrant:

```
# vagrant ssh compliance
# exit
```

Anotações

Recapitulando

1. O que são provisionadores
2. Tipos de Provisionadores
3. Utilizando Provisionadores
4. Construindo o Ambiente

Anotações