

AUTOMAÇÃO DE CONFIGURAÇÕES

Introdução ao Ansible com Vagrant



INFORMAÇÕES GERAIS

OBJETIVO

Cobrir os conhecimentos básicos de Ansible e Vagrant visando dar liberdade para testa e praticar novas técnicas de forma rápida e segura em ambiente local.

PÚBLICO ALVO

Profissionais de TI das diversas áreas e disciplinas que precisam executar tarefas administrativas, bem como aqueles que estão iniciando e tendo o primeiro contato com automação de infraestrutura.

SOBRE

Conteúdo criado para incentivar o estudo e o compartilhamento de conhecimento dentro e fora da empresa, despertar o interesse em procurar ferramentas e processos mais eficientes que possam melhorar a forma como trabalhamos. Com isso iremos gastar mais tempo planejando e menos tempo executando tarefas manuais repetitivas suscetíveis a falhas, diminuindo o risco e aumentando qualidade nos serviços e entregas.

REQUISITOS

- Sistema Operacional Windows 10 (Nesse exemplo utilizaremos Windows, presente como única alternativa no dia a dia de muitas corporações).
- Computador com acesso à internet.
- 4 GB de memória RAM.
- Editor de texto (Visual Studio Code, Atom, Notepad ++ ...)
- Código exemplo do projeto, disponível em: <https://github.com/guilhermepozo/config-ansible-vagrant>

SUMÁRIO

CENÁRIO PROPOSTO	
Desenhando ambiente	4
VAGRANT	
Informações Gerais	5
VAGRANT	
Instalando em Windows	6
VIRTUALBOX	
Informações Gerais	7
VIRTUALBOX	
Instalando em Windows	8
VAGRANT	
Primeiros Passos	9
VAGRANT	
Conexão Remota	10
VAGRANT	
Customização	11
VAGRANT	
Interagindo	12
PLANEJAMENTO	
Montando Ambiente Virtual	13
VAGRANTFILE	
Preparação e Variáveis	14
VAGRANTFILE	
Laço, Nome e Imagem Base	15
VAGRANTFILE	
Hostname, Redirecionamento e Rede	16
VAGRANTFILE	
Provedor e Sincronismo	17
VAGRANTFILE	
Provisionamento	18
VAGRANTFILE	
Provisionamento	19
ANSIBLE	
Conceitos	20
ANSIBLE	
Inventory	21
ANSIBLE	
Comandos Ad-hoc	22
ANSIBLE	
Playbook Hosts e Users	23
ANSIBLE	
Playbook Tasks e Modules	24
ANSIBLE	
Playbook Tasks e Modules	25
ANSIBLE	
Playbook Handlers	26
ANSIBLE	
Variáveis	27
ANSIBLE	
Inventory Variables	28
ANSIBLE	
Acessando Variáveis e Templating	29
PLANEJAMENTO	
Etapas Detalhadas	30
ANSIBLE	
Automação Banco de Dados	31
ANSIBLE	
Automação Servidor Web	32
ANSIBLE	
Preparando Ambiente de Desenvolvimento	33
ANSIBLE	
Conectado e Verificando o Workspace	34
ANSIBLE	
Verificando Workspace	35
ANSIBLE	
Executando o Playbook	36
VALIDAÇÃO	
Verificando Execução	37
CONCLUSÃO	
Considerações	38

CENÁRIO PROPOSTO

Desenhando ambiente

Solução

Para dar liberdade e segurança ao usuário poder testar e praticar novas tecnologias, iremos utilizar o Vagrant, provisionando 3 máquinas virtuais, sendo uma estação de trabalho com Ansible e duas parte da arquitetura: MySQL, PHP, Apache e Wordpress, todas Ubuntu 18.04.

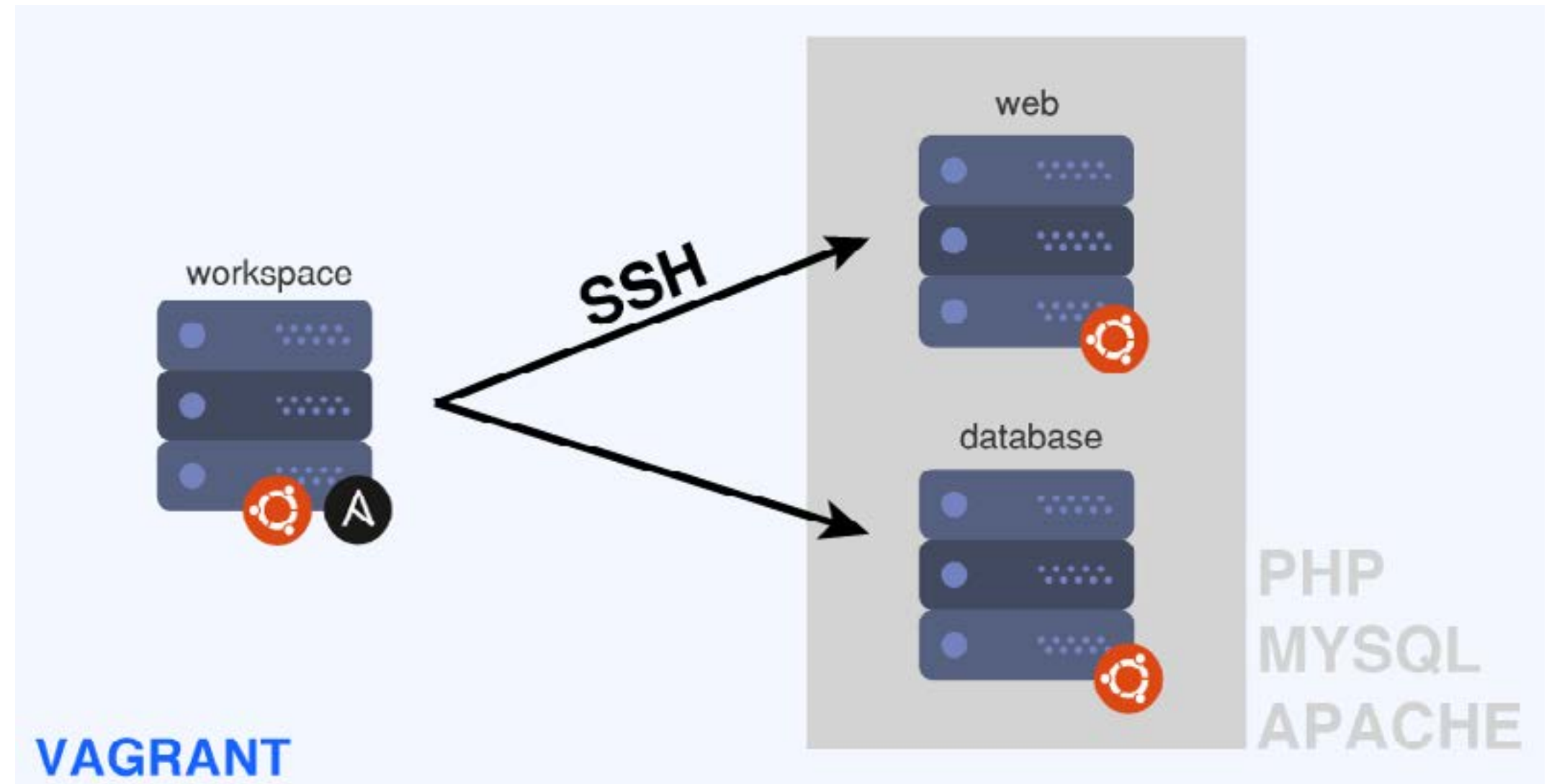
Tarefas

- Criar 3 máquinas virtuais com Vagrant.
- Instalar Ansible e ferramentas de trabalho (workspace).
- Criar automação para instalar PHP, Apache e Wordpress (web).
- Criar automação para instalar MySQL (database).

A resolução do problema e o código fonte está disponível em:

<https://github.com/guilhermepozo/config-ansible-vagrant>

Claro que detalharemos todos os arquivos.



VAGRANT

Informações Gerais

Download

<https://www.vagrantup.com/downloads.html>

Compatibilidade

- Debian
- Windows
- Centos
- macOS
- Arch Linux
- Linux

O que é?

Ferramenta para automatizar a criação de máquinas virtuais com:

- VMWare (Pago).
- VirtualBox*.
- Hyper-V.

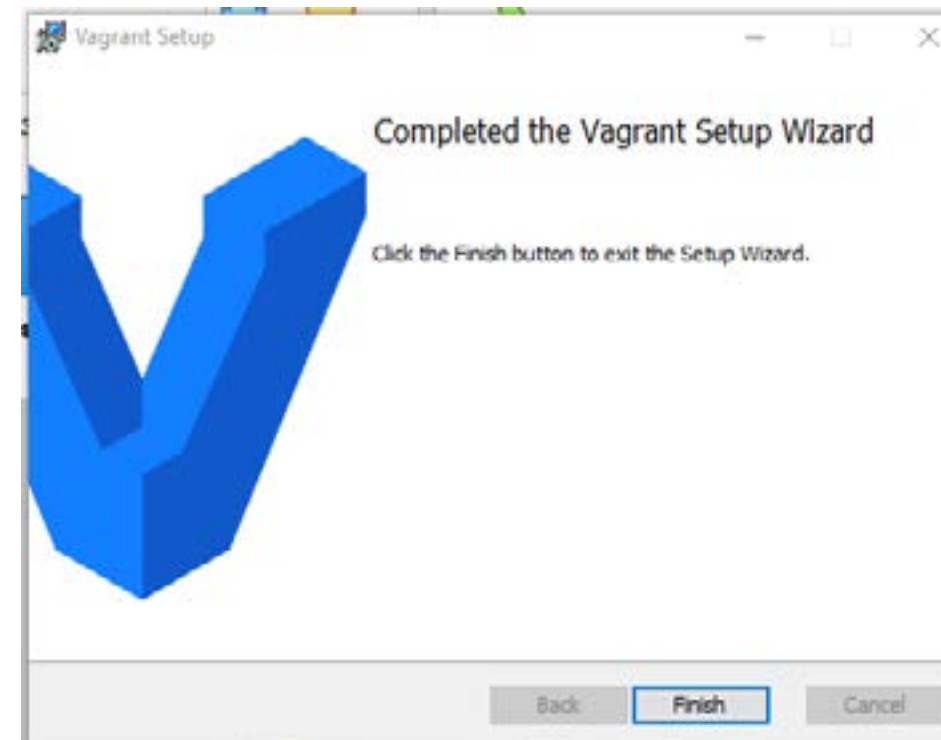
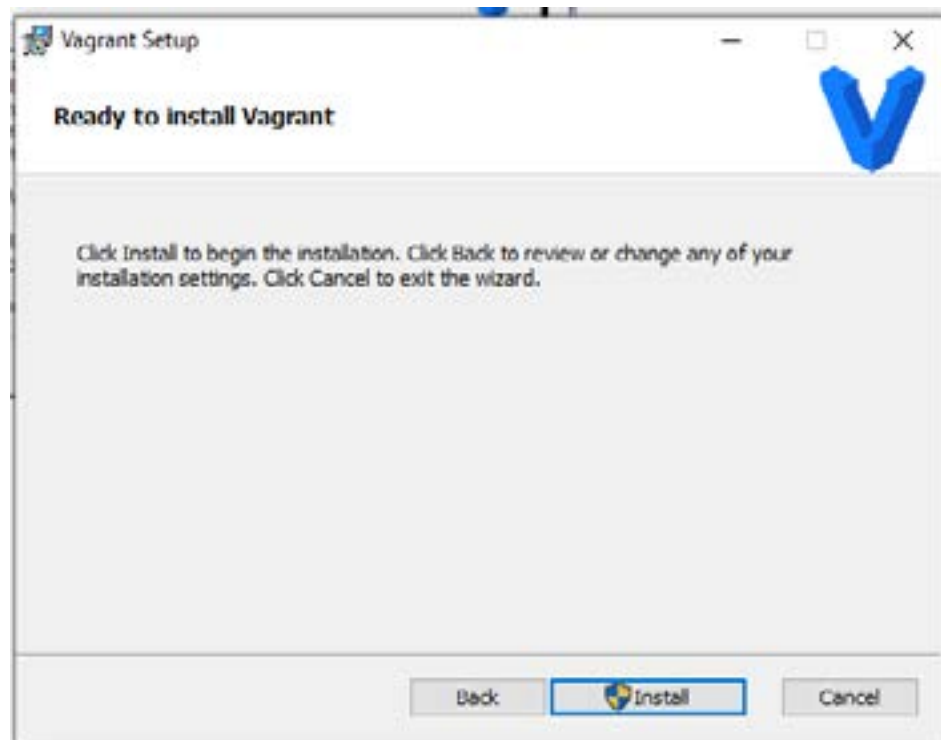
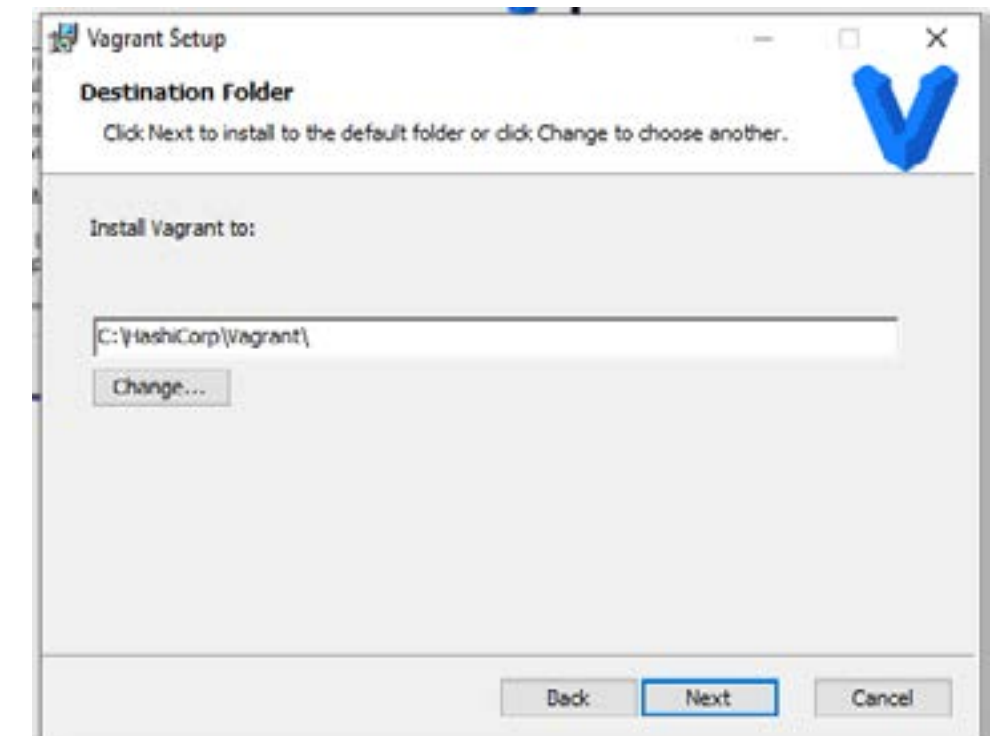
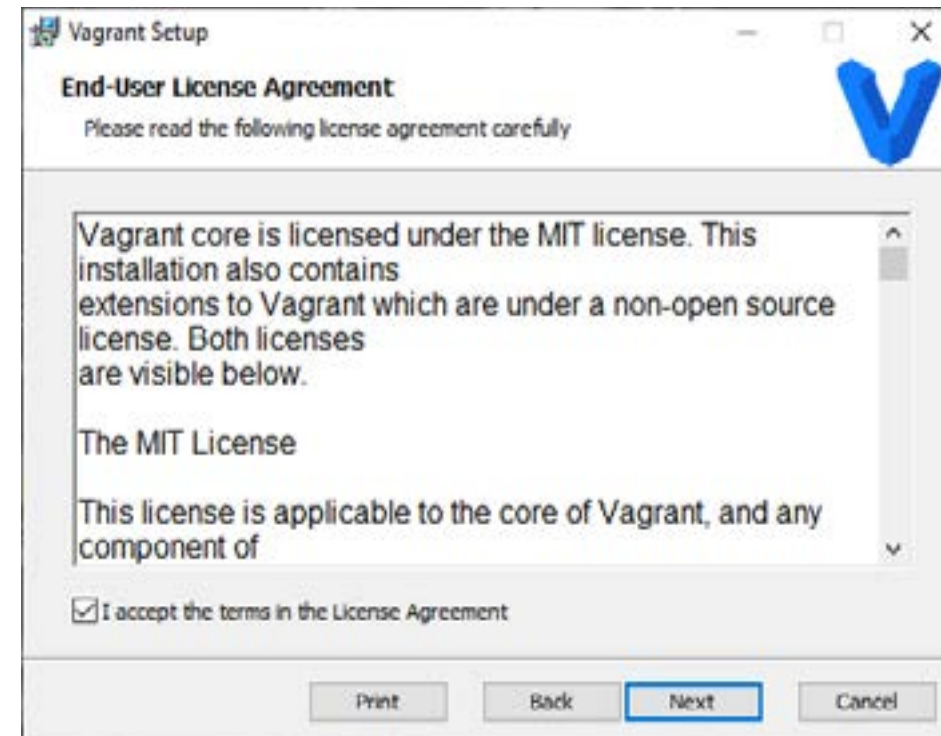
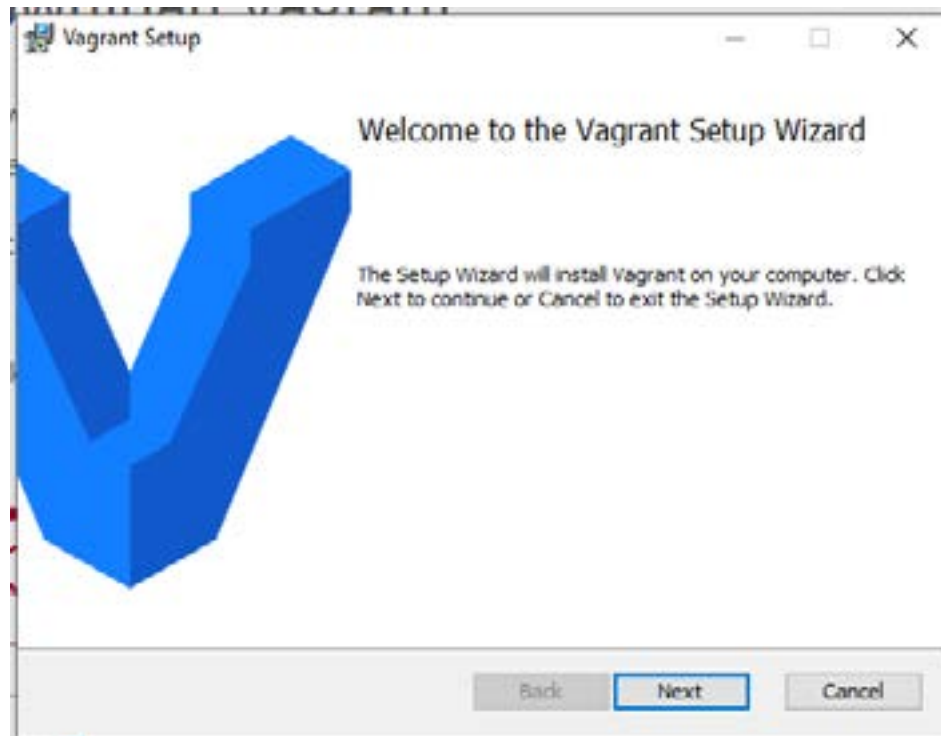
Porque?

- Simples e fácil.
- Reproducibilidade.
- Portabilidade.
- Comandos (Terminal).

*Iremos utilizar o VirtualBox como provedor de virtualização local.

VAGRANT

Instalando em Windows



VIRTUALBOX

Informações Gerais

Download

<https://www.virtualbox.org/wiki/Downloads>

Compatibilidade

- Windows hosts.
- OS X hosts.
- Linux distributions.
- Solaris hosts.

O que é?

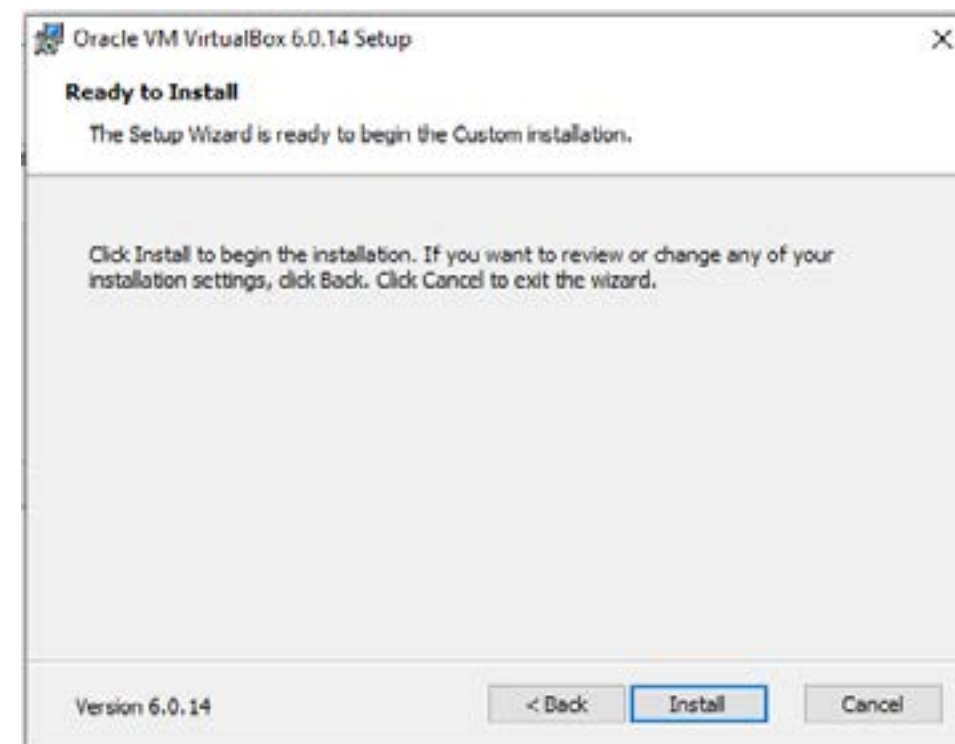
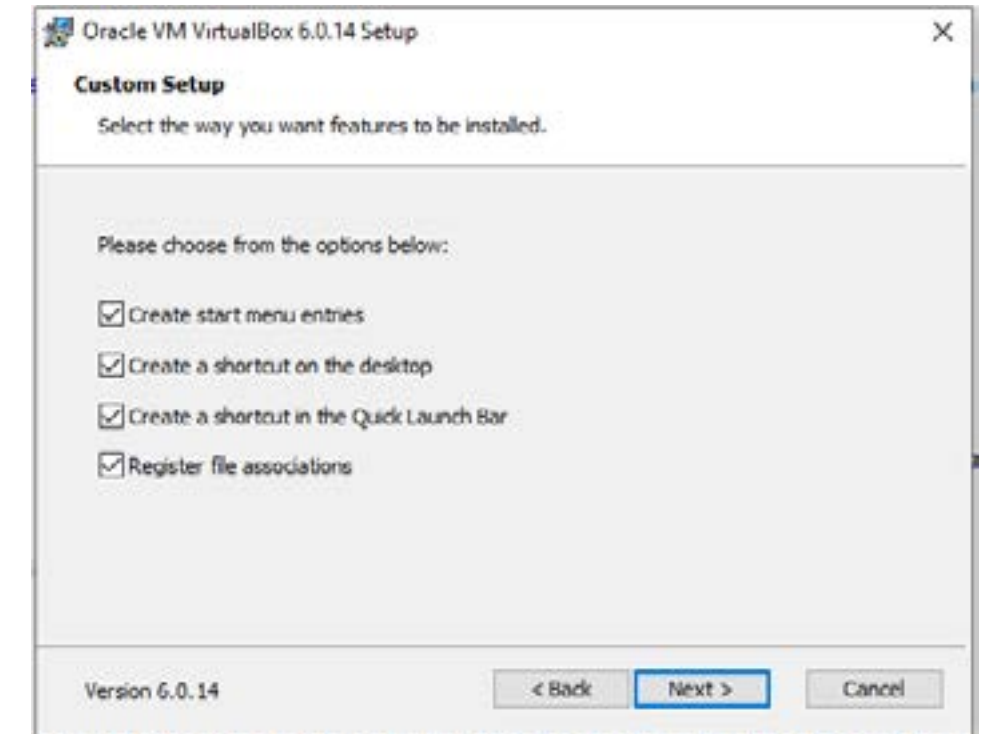
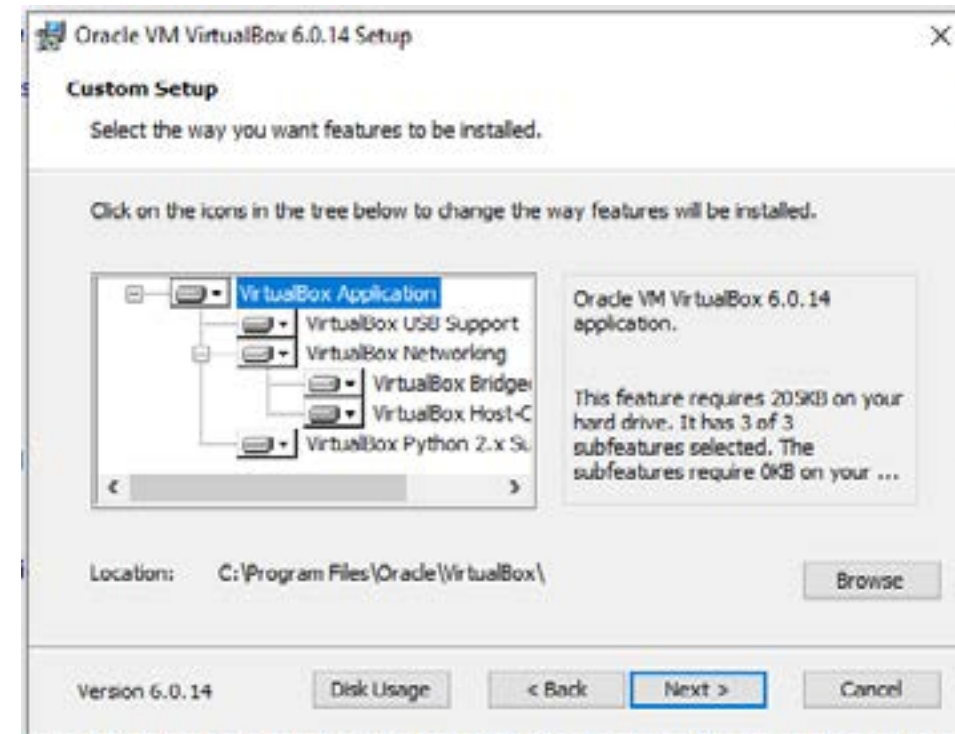
Ferramenta para virtualização que permite instalar e executar diferentes sistemas operacionais em um único computador sem complicações.

Porque?

- Sem custo.
- Intuitivo.
- Console simples.

VIRTUALBOX

Instalando em Windows



VAGRANT

Primeiros Passos

Verifique se a instalação ocorreu com sucesso.

```
vagrant --version
```

```
C:\vagrant>vagrant --version  
Vagrant 2.2.6
```

Crie um diretório no local de sua preferência e nele crie um template de máquina virtual a partir de uma box. (<https://app.vagrantup.com/boxes/search>)

```
vagrant init ubuntu/bionic64
```

Inicie uma máquina virtual de acordo com o template criado anteriormente, lembrando que o comando deve ser executado no mesmo local que o passo anterior.

```
vagrant up
```

```
C:\Windows\system32\cmd.exe  
C:\vagrant>vagrant init ubuntu/bionic64  
A 'Vagrantfile' has been placed in this directory. You are now  
ready to 'vagrant up' your first virtual environment! Please read  
the comments in the Vagrantfile as well as documentation on  
'vagrantup.com' for more information on using Vagrant.  
  
C:\vagrant>vagrant up  
Bringing machine 'default' up with 'virtualbox' provider...  
=> default: Box 'ubuntu/bionic64' could not be found. Attempting to find and install...  
default: Box Provider: virtualbox  
default: Box Version: >= 0  
=> default: Loading metadata for box 'ubuntu/bionic64'  
default: URL: https://vagrantcloud.com/ubuntu/bionic64  
=> default: Adding box 'ubuntu/bionic64' (v20191018.0.0) for provider: virtualbox  
default: Downloading: https://vagrantcloud.com/ubuntu/boxes/bionic64/versions/20191018.0.0/provi  
ders/virtualbox.box  
default: Download redirected to host: cloud-images.ubuntu.com  
default:  
=> default: Successfully added box 'ubuntu/bionic64' (v20191018.0.0) for 'virtualbox'!  
=> default: Importing base box 'ubuntu/bionic64'...  
=> default: Matching MAC address for NAT networking...  
=> default: Checking if box 'ubuntu/bionic64' version '20191018.0.0' is up to date...  
=> default: Setting the name of the VM: vagrant_default_1571450150566_71415  
=> default: Clearing any previously set network interfaces...  
=> default: Preparing network interfaces based on configuration...  
default: Adapter 1: nat  
=> default: Forwarding ports...  
default: 22 (guest) => 2222 (host) (adapter 1)  
=> default: Running 'pre-boot' VM customizations...  
=> default: Booting VM...  
=> default: Waiting for machine to boot. This may take a few minutes...  
default: SSH address: 127.0.0.1:2222  
default: SSH username: vagrant  
default: SSH auth method: private key  
default: Warning: Connection reset. Retrying...  
default: Warning: Connection aborted. Retrying...  
default:  
default: Vagrant insecure key detected. Vagrant will automatically replace  
default: this with a newly generated keypair for better security.  
default:  
default: Inserting generated public key within guest...  
default: Removing insecure key from the guest if it's present...  
default: Key inserted! Disconnecting and reconnecting using new SSH key...  
=> default: Machine booted and ready!  
=> default: Checking for guest additions in VM...  
default: The guest additions on this VM do not match the installed version of  
default: VirtualBox! In most cases this is fine, but in rare cases it can  
default: prevent things such as shared folders from working properly. If you see  
default: shared folder errors, please make sure the guest additions within the  
default: virtual machine match the version of VirtualBox you have installed on  
default: your host and reload your VM.  
default:  
default: Guest Additions Version: 5.2.32  
default: VirtualBox Version: 6.0  
=> default: Mounting shared folders...  
default: /vagrant => C:/vagrant  
  
C:\vagrant>
```

VAGRANT

Conexão Remota

Após aguardar a inicialização, conecte-se na máquina virtual criada.

`vagrant ssh`

```
vagrant@ubuntu-bionic: ~$ vagrant ssh
C:\vagrant>vagrant ssh
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

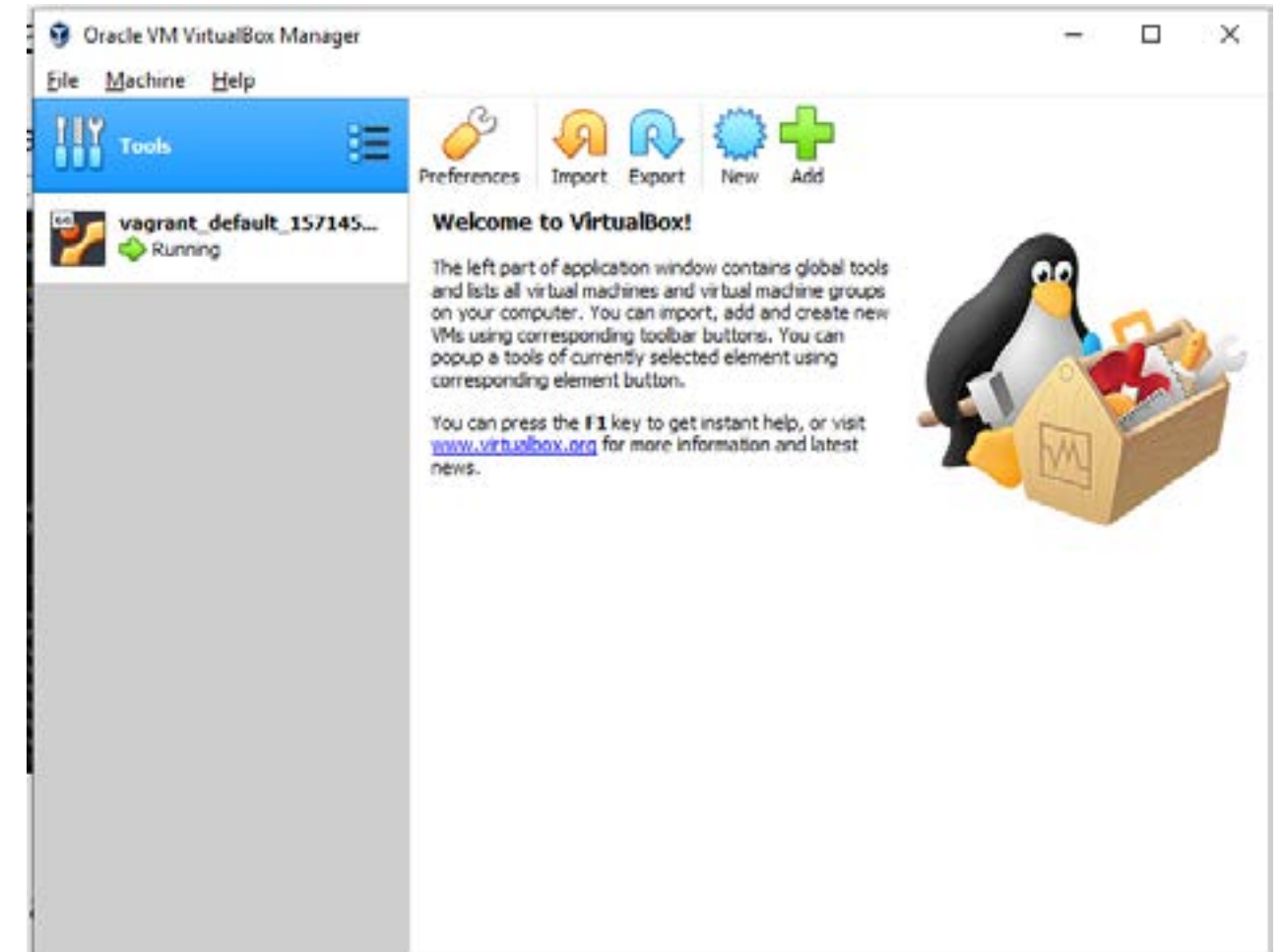
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Oct 19 04:11:19 UTC 2019

System load:  0.34               Processes:    98
Usage of /:   10.0% of 9.63GB    Users logged in: 0
Memory usage: 12%               IP address for enp0s3: 10.0.2.15
Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

vagrant@ubuntu-bionic:~$ ll
total 32
drwxr-xr-x 5 vagrant vagrant 4096 Oct 19 04:09 ./
drwxr-xr-x 4 root root 4096 Oct 19 04:09 ../
-rw-r--r-- 1 vagrant vagrant 220 Oct 18 16:15 .bash_logout
-rw-r--r-- 1 vagrant vagrant 3771 Oct 18 16:15 .bashrc
drwx----- 2 vagrant vagrant 4096 Oct 19 04:09 .cache/
drwx----- 3 vagrant vagrant 4096 Oct 19 04:09 .gnupg/
-rw-r--r-- 1 vagrant vagrant 807 Oct 18 16:15 .profile
drwx----- 2 vagrant vagrant 4096 Oct 19 04:09 .ssh/
vagrant@ubuntu-bionic:~$
```



O comando irá conectar na única máquina criada, caso existam outras, passe nome ou id da máquina (nome do console do VirtualBox), mais informações: <https://www.vagrantup.com/docs/cli/ssh.html>

Com isso já conectamos na máquina virtual Ubuntu 18.04.3, mas o processo é o mesmo independente do sistema operacional, procure um outro template em: <https://app.vagrantup.com/boxes/search>.

VAGRANT

Customização

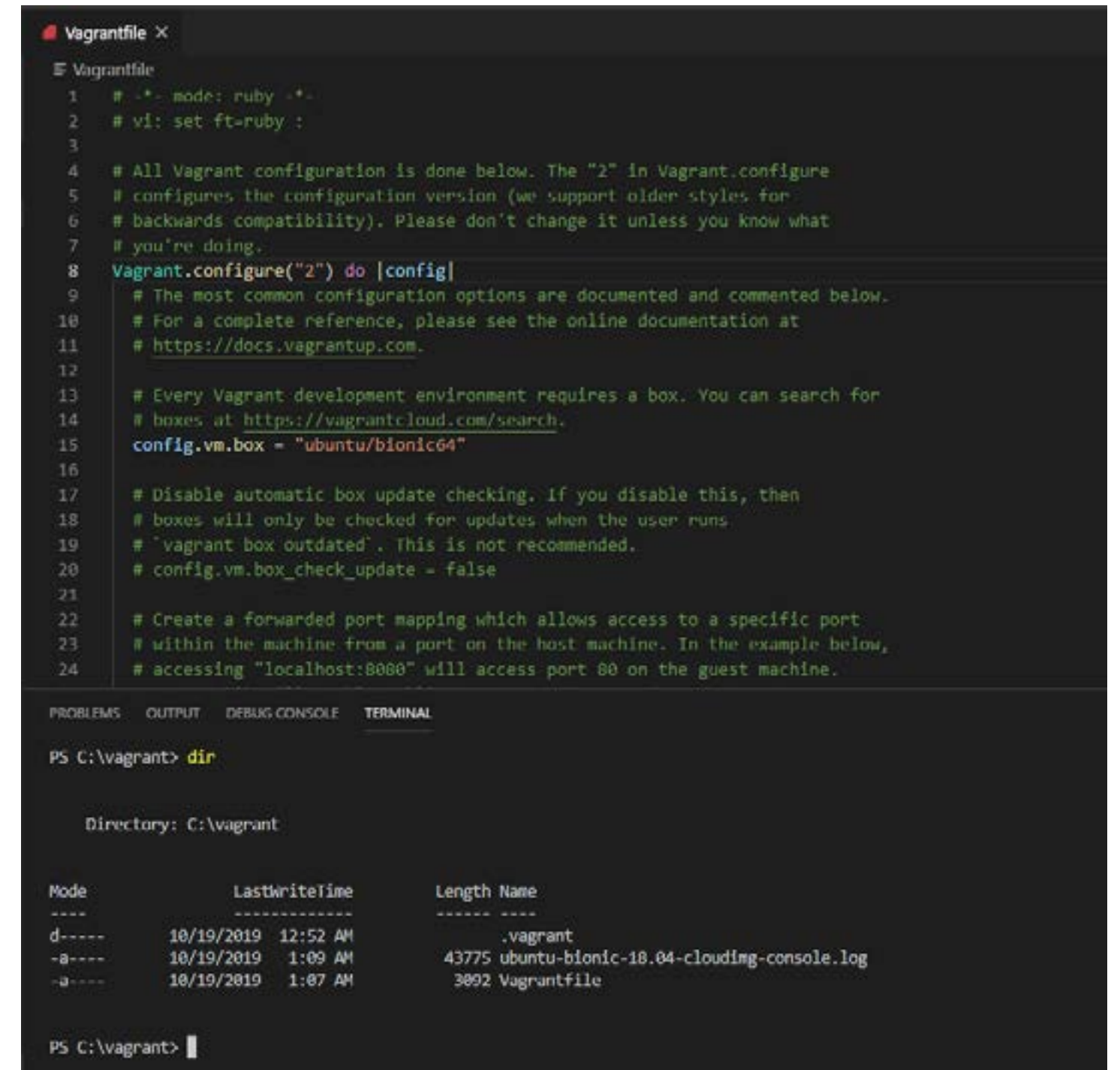
Vagrantfile são arquivos em Ruby usados para configurar o Vagrant, sendo um arquivo por projeto/diretório.

Não é necessário conhecer Ruby, somente os parâmetros usados para as customizações, mais informações: <https://www.vagrantup.com/docs/vagrantfile>.

Existem diversas opções e parâmetros usados para customizar o Vagrantfile, dentre eles:

- Qual template (box) usar.
- Compartilhar diretórios.
- Configurações rede e redirecionamentos de portas.
- Memória, CPU.
- SSH e segurança.

Em breve iremos cobrir com detalhes o uso do Vagrantfile.



The screenshot shows a code editor with a file named 'Vagrantfile'. The code is in Ruby and includes comments explaining the configuration. The terminal window at the bottom shows the command 'dir' being executed in the directory 'C:\vagrant', displaying a list of files and directories.

```
Vagrantfile
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 # All Vagrant configuration is done below. The "2" in Vagrant.configure
5 # configures the configuration version (we support older styles for
6 # backwards compatibility). Please don't change it unless you know what
7 # you're doing.
8 Vagrant.configure("2") do |config|
9   # The most common configuration options are documented and commented below.
10  # For a complete reference, please see the online documentation at
11  # https://docs.vagrantup.com.
12
13  # Every Vagrant development environment requires a box. You can search for
14  # boxes at https://vagrantcloud.com/search.
15  config.vm.box = "ubuntu/bionic64"
16
17  # Disable automatic box update checking. If you disable this, then
18  # boxes will only be checked for updates when the user runs
19  # 'vagrant box outdated'. This is not recommended.
20  # config.vm.box_check_update = false
21
22  # Create a forwarded port mapping which allows access to a specific port
23  # within the machine from a port on the host machine. In the example below,
24  # accessing "localhost:8080" will access port 80 on the guest machine.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\vagrant> dir

Directory: C:\vagrant

Mode	LastWriteTime	Length	Name
d----	10/19/2019 12:52 AM		.vagrant
-a----	10/19/2019 1:09 AM	43775	ubuntu-bionic-18.04-cloudimg-console.log
-a----	10/19/2019 1:07 AM	3092	Vagrantfile

PS C:\vagrant> █

VAGRANT

Interagindo

Todas as manipulações realizadas com Vagrant ocorrem através de comandos no terminal (cli - command line interface), extensão instalada automaticamente com o Vagrant, mais informações: <https://www.vagrantup.com/docs/cli>.

Comandos Básicos

`vagrant init user/box` - Cria um Vagrant file de acordo com o template/box especificado.

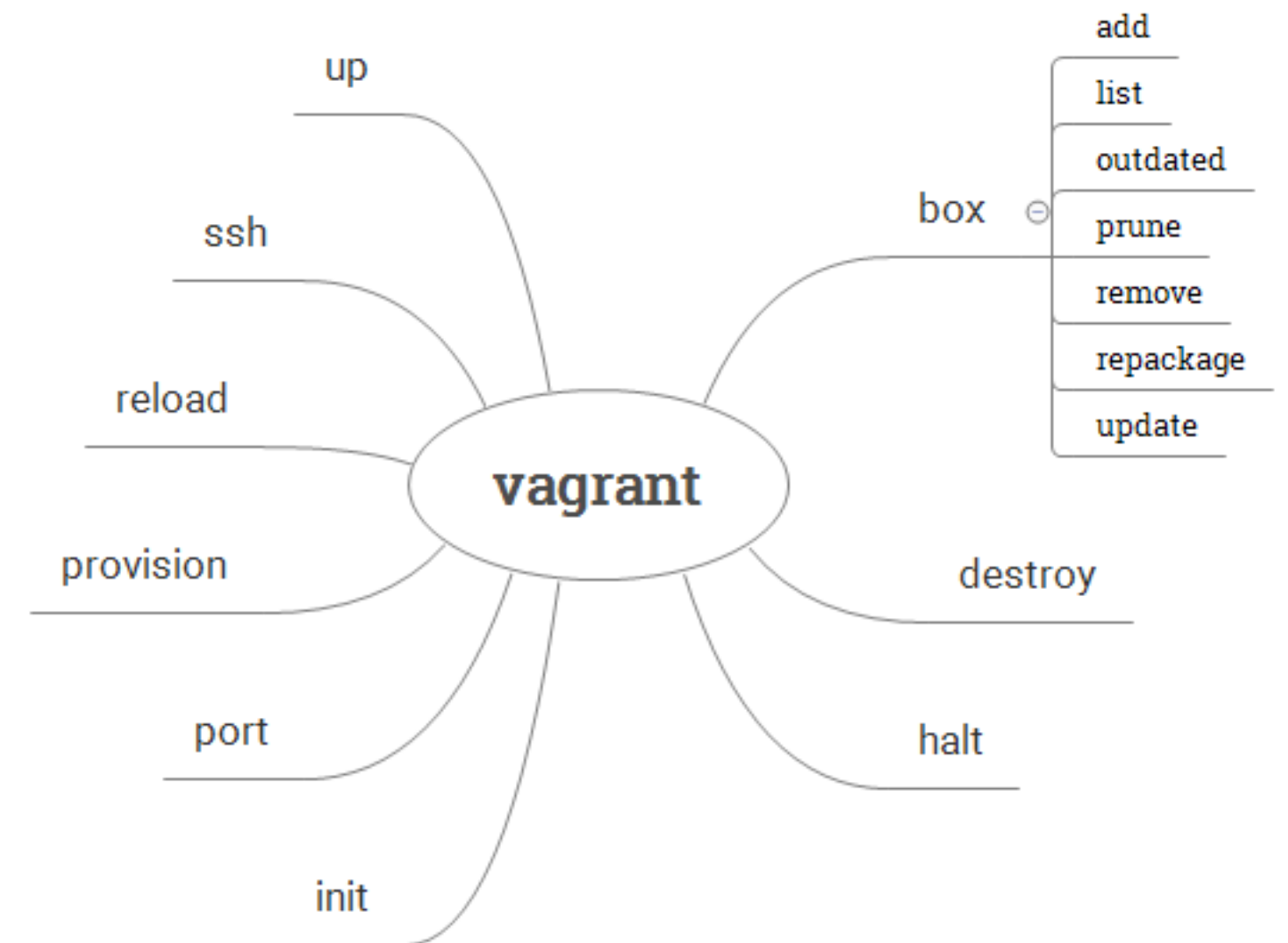
`vagrant up` - Cria ou inicializa as Máquinas Virtuais (de acordo com o Vagrantfile do local da execução do comando).

`vagrant reload` - Atualiza a Máquina Virtual caso Vagrantfile seja modificado.

`vagrant halt` - Desliga a Máquina Virtual.

`vagrant destroy` - Remove a Máquina Virtual.

`vagrant ssh` - Conecta à Máquina Virtual.



PLANEJAMENTO

Montando Ambiente Virtual

Para criar 3 as máquinas virtuais, conforme solução proposta, usaremos o **Vagrantfile**, nele iremos:

- Definir Nome e IP de cada máquina virtual.
- Redirecionar porta 80 da máquina virtual web para porta 7070 do seu computador.
- Criar um diretório compartilhado entre a máquina virtual workspace e seu computador.
- Inserir chave pública nas máquinas virtuais criadas para acesso remoto (ssh).
- Copiar chave privada para máquina virtual workspace.
- Instalar e configurar Ansible na máquina virtual workspace.

Lembrando que todas essas configurações serão feitas com código, visando reproducibilidade e agilidade, pois assim poderemos destruir e subir o ambiente todo com poucos comandos.

VAGRANTFILE

Preparação e Variáveis

O Vagrantfile presente na raiz do projeto está comentado linha a linha, mas para melhor entendimento, declaramos um array de objetos (notação em Ruby) com os atributos: Nome e IP, onde serão utilizados em breve.

```
Vagrantfile X
Vagrantfile
1 | # DECLARAR ARRAY DE OBJETOS COM INFORMAÇÕES DAS MÁQUINAS VIRTUAIS QUE SERÃO CRIADAS
2   hosts = [{name: "workspace", ip:"192.168.56.65"},{name: "web", ip:"192.168.56.66"},{name: "database",ip:"192.168.56.67"}]
3
```

Após isso declaramos uma variável que irá receber o caminho que está a chave pública usada para as conexões do Ansible. O projeto contém um par de chaves, pública e privada, mas sintá-se a vontade de utilizar outro par.

Em Windows acrescente uma \ a mais quando for passar um caminho no Vagrantfile.

```
4   # DECLARAR CAMINHO DA CHAVE PÚBLICA
5   pubkeypath = "shared\\ssh\\public"
```

VAGRANTFILE

Laço, Nome e Imagem Base

Começaremos as configurações das máquinas virtuais, para isso usaremos um loop passando pelo array de objetos declaramos anteriormente.

Para cada objeto do array iremos definir todas as propriedades da máquina virtual (Nome, Hostname, IP) e aplicar as configurações necessárias (compartilhar diretório, redirecionamento de porta...).

`.vm.define` é usado para definir uma máquina virtual em um ambiente Multi-Machine (<https://www.vagrantup.com/docs/multi-machine>), assim iremos passar o parâmetro `name`, do objeto atual do loop e um bloco (callback em Ruby) que será chamado.

```
8      # LOOP PARA CRIAR MÁQUINAS DECLARADAS NO OBJETO "hosts" (LINHA 2)
9      hosts.each do |host|
10         config.vm.define host[:name] do |node|
```

Uma variável auxiliar será declarada com o nome da máquina para realizar as condições (Instalar Ansible somente da máquina workspace).

Também indicamos que as máquinas virtuais usarão como base a imagem “ubuntu/bionic64”, disponível gratuitamente na plataforma de “boxes” do Vagrant (<https://app.vagrantup.com/boxes/search>).

```
11         # VARIÁVEL AUXILIAR COM O NOME, SERÁ USADA PARA REALIZ
12         name = host[:name]
13
14         # BOX USADA COMO BASE - https://app.vagrantup.com/boxes/search
```

VAGRANTFILE

Hostname, Redirecionamento e Rede

Vamos definir o hostname da máquina virtual com `.vm.hostname`.

Com `.vm.network` iremos criar o redirecionamento de porta e também definiremos tanto o modo de rede como IP.

```
17 # CONFIGURANDO HOSTNAME COM NOME DECLARADO ANTERIORMENTE (OBJETO hosts - LINHA 2)
18 node.vm.hostname = host[:name]
19
20 # REDIRECT DE PORTAS CASO NOME DA MÁQUINA SEJA "web"
21 node.vm.network "forwarded_port", guest: 80, host: 7070 if name == "web"
22
23 # CRIANDO REDE PRIVADA E ATRIBUINDO IP DECLARADO ANTERIORMENTE (ARRAY DE OBJETOS "hosts" - LINHA 2)
24 node.vm.network "private_network"
```

Para que seja possível acessar o apache direto do seu computador, será necessário redirecionar a porta 80 da máquina virtual para a porta 7070 do seu computador, lembrando que essa configuração só deve ser feita caso seja a máquina virtual de nome web.

Nesse caso utilizamos o modo de rede “private_network” para possibilitar a comunicação entre as máquinas virtuais de forma isolada e segura, existem outros tipos de rede, sintá-se a vontade de testar outros modos, mais informações: <https://www.vagrantup.com/docs/networking>.

Definimos IPs do tipo estático, sintá-se a vontade de testar outros tipos, mais informações: https://www.vagrantup.com/docs/networking/private_network.html.

VAGRANTFILE

Provedor e Sincronismo

Agora definimos qual provedor virtual será utilizado, no caso Virtualbox, também definindo o nome da máquina que será apresentado no console.

```
26 # CONFIGURANDO NOME COM NOME DECLARADO ANTERIORMENTE (ARRAY DE OBJETOS "hosts" - LINHA 2)
27 node.vm.provider :virtualbox do |vb|
28   vb.name = host[:name]
29 end
30
31 # CRIANDO PASTA COMPARTILHADA ENTRE SUA MÁQUINA E MÁQUINA VIRTUAL CASO NOME DA MÁQUINA SEJA "workspace"
32 config.vm.synced_folder "shared/", "/home/vagrant/shared" if name == "workspace"
```

Também criamos o diretório compartilhado entre seu computador e a máquina virtual workspace.

Com `.vm.synced_folder` montamos um compartilhamento com o caminho “shared/”, diretório do projeto que contém os playbooks e chaves, no caminho “/home/vagrant/shared”. O caminho alvo, do sistema operacional da máquina virtual, deve ser absoluto.

Lembrando que essa configuração só deve ser feita caso seja a máquina virtual workspace.

Existem outras opções para sincronizar diretórios, sintá-se a vontade de testar e customizar e acordo com sua necessidade, mais informações:
https://www.vagrantup.com/docs/synced-folders/basic_usage.html.

VAGRANTFILE

Provisionamento

Cada demanda tem sua característica, muitas vezes necessário customizar o sistema operacional base. Vagrant também é capaz de automaticamente instalar softwares e alterar configurações direto sistema operacional com `.vm.provision`.

```
34 # INSERINDO CHAVE PÚBLICA NO SISTEMA OPERACIONAL PARA SSH
35 config.vm.provision "shell" do |s|
36   # LER ARQUIVO DE CHAVE PUBLICA (RUBY)
37   ssh_pub_key = File.readlines(pubkeypath).first.strip
38   s.inline = <<-SHELL
39     echo #{ssh_pub_key} >> /home/vagrant/.ssh/authorized_keys
40     echo #{ssh_pub_key} >> /root/.ssh/authorized_keys
41   SHELL
42 end
```

Iremos utilizar uma função Ruby para ler o arquivo da chave pública, passando a variável: `pubkeypath`, com o caminho do arquivo.

Após isso iremos utilizar o método `.inline` para executar alguns comandos shell direto no sistema operacional da máquina virtual.

Por estar dentro do laço, estaremos inserindo nossa chave pública dentro dos sistemas operacionais de todas as máquinas virtuais.

Existem outras opções de provisionamento além de shell, mais informações: <https://www.vagrantup.com/docs/provisioning>, como também outras maneiras de executar scripts shell, mais informações: <https://www.vagrantup.com/docs/provisioning/shell.html>.

VAGRANTFILE

Provisionamento

Agora faremos o mesmo processo de provisionamento mas nesse caso será específico para a máquina workspace. Iremos atualizar o sistema operacional, instalar o Ansible, declarar a variável: `ANSIBLE_HOST_KEY_CHECKING=0` para ignorar o processo de confiança/autenticidade de chaves do SSH (conexões do Ansible), copiar a chave privada para o destino: `~/.ssh`, copiar o arquivos de hosts do Ansible e alterar a permissionamento da chave privada usada para as conexões SSH.

Com isso já podemos subir as 3 máquinas virtuais que serão usadas no exercício, vá até o local que está o Vagrantfile e de o comando `vagrant up`.

Poderíamos provisionar diretamente as máquinas usando Ansible, mas para o exercício iremos subir uma máquina ubuntu para ser workspace e facilitar a adesão de usuários Windows ao Linux.

```
45 # EXECUTAR AÇÕES DE PROVISIONAMENTO NA MÁQUINA "workspace"
46 config.vm.define "workspace" do |node|
47   # EXECUTAR AÇÕES DE PROVISIONAMENTO DO TIPO "shell"
48   node.vm.provision "shell" do |s|
49     s.inline = <<-SHELL
50     # ATUALIZANDO PACOTES DO SISTEMA OPERACIONAL
51     apt update
52     # INSTALAÇÃO DO ANSIBLE
53     apt install software-properties-common
54     apt-add-repository --yes --update ppa:ansible/ansible
55     apt install ansible -y
56     # IGNORAR CONFIRMAÇÃO MANUAL NO TERMINAL DE CONFIANÇA DE CHAVES NO MOMENTO DO SSH
57     echo "export ANSIBLE_HOST_KEY_CHECKING=0" >> /home/vagrant/.bashrc
58     # COPIAR CHAVE PRIVADA PARA LOCAL PADRÃO SSH
59     cp /home/vagrant/shared/ssh/private /home/vagrant/.ssh/
60     # COPIAR ARQUIVO DE HOSTS PARA LOCAL PADRÃO DO ANSIBLE
61     cat /home/vagrant/shared/ansible/inventory/hosts >> /etc/ansible/hosts
62     # ALTERAR PERMISSIONAMENTO DA CHAVE PRIVADA PARA SSH
63     chmod 600 /home/vagrant/.ssh/private
64   SHELL
65   end
66 end
```

ANSIBLE

Conceitos

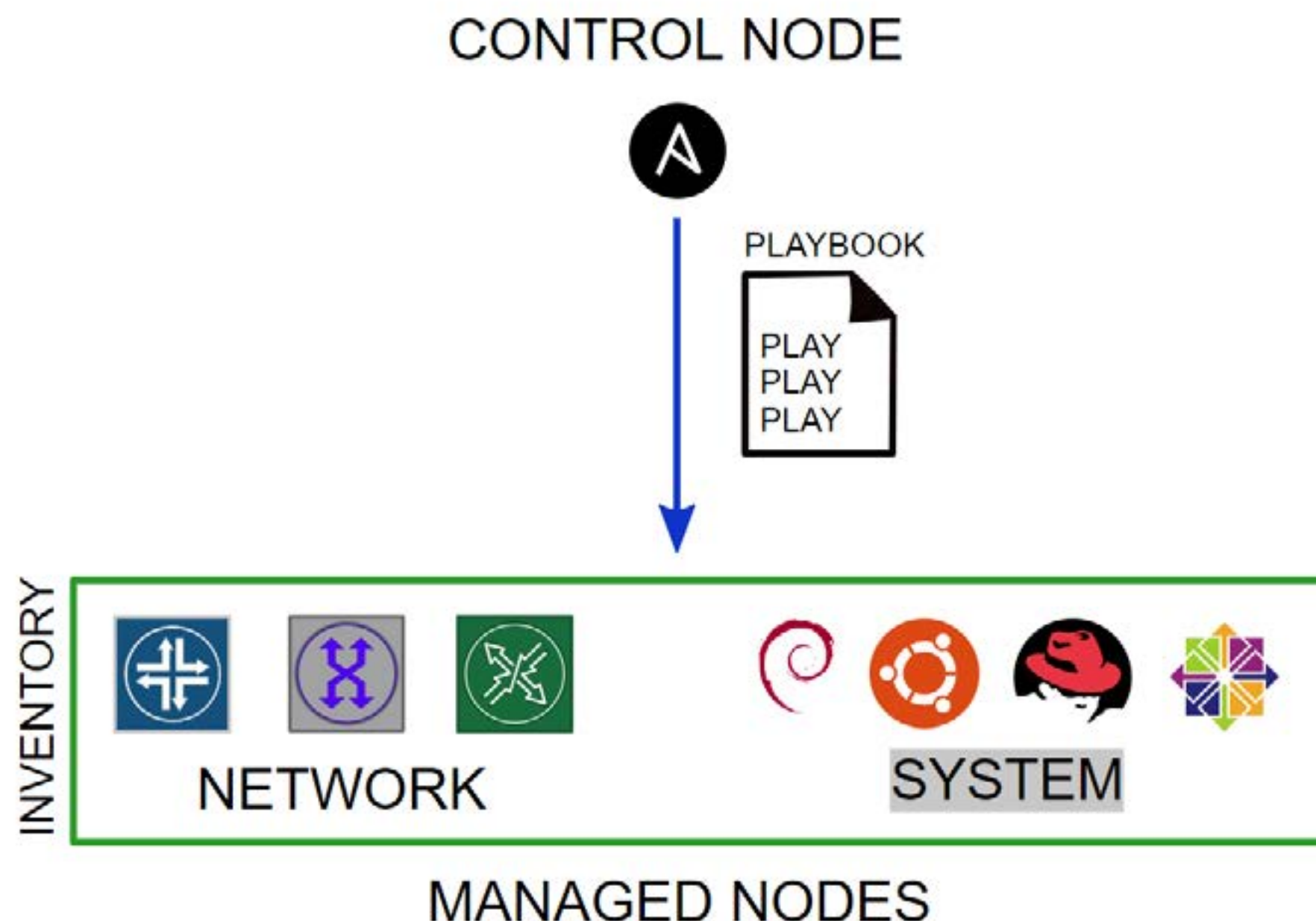
Ansible é uma plataforma de gerenciamento de configurações, ocupando o mesmo espaço que Puppet, Chef, e SaltStack no mundo DEVOPS. É considerado uma ferramenta radicalmente simples e fácil de aprender sem nenhuma experiência com programação.

Escrito em Python, Ansible não utiliza agentes para realizar suas configurações, ao invés disso envia os programas e módulos para serem executados diretamente no alvo, via conexão remota.

Esse é um dos pontos fundamentais para sua adoção, poupando esforços de administração de agentes, consumo de rede.

Podemos dividi-lo em 4 componentes básicos:

- **Control Node:** Ponto central de gerenciamento, de onde irá partir as conexões e onde estará presente seus códigos.
- **Managed Node:** Máquinas alvo gerenciados pelo Control Node (necessário Python).
- **Inventory:** Mapeamento das informações dos Managed Nodes.
- **Plabooks:** Lista de ações que serão executadas nos Managed Nodes, configuração como código.



Inventory

Inventory é um arquivo que informa ao Ansible como e onde ele vai atuar, quais máquinas ele tem disponível para conectar e armazena outras informações como: hostname, ip, variáveis, usuário de conexão, interpretador python.

Pode listar máquinas individuais ou grupos, definidos pelo usuário, facilitando a gestão de máquinas semelhantes.

Também pode incluir variáveis para serem utilizadas em hosts individuais ou grupos sendo parâmetros em seus playbooks, tornando o Ansible ainda mais profissional, maleável e idempotente.

Existem diversas maneiras de organizar o inventory, sua forma mais simples é uma lista em um único arquivo no caminho padrão: `/etc/ansible/hosts`.

Também podemos criar inventories dinâmicos em python (mapear máquinas na AWS, OpenStack, Beyondtrust), separar e organizar por grupos, hosts, diretórios, ou até alterar o caminho do arquivo padrão, mas são assuntos que serão discutidos em outro momento.

Mais informações: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#

ANSIBLE

Comandos Ad-hoc

Ansible pode também executar milhares de comandos simultâneos diretamente nos servidores com comandos ad-hoc, mais informações: https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html .

Administradores de sistemas executam diversas tarefas repetidas no dia a dia:

- Aplicar patches, atualizar e instalar pacotes.
- Verificar recursos (CPU, memória, disco).
- Checar arquivo de log.
- Gerenciar usuários e grupos.
- Copiar e mover arquivos.
- Deploy de aplicações.
- Gerenciar servidores e serviços (start, stop, restart).
- Gerenciar rotinas cron (jobs).

```
hakase@ansible-node:~$  
hakase@ansible-node:~$  
hakase@ansible-node:~$ ansible all -m ping  
provision | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}  
provision2 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}  
hakase@ansible-node:~$  
hakase@ansible-node:~$ █
```

Todas essas rotinas podem ser facilmente executadas de forma inteligente e simultânea com Ansible. Claro que alguns casos exigirão ações humanas, pois normalmente o legado não é padronizado e algum comando não funcionará em 100% das máquinas.

Lembrando que comandos ad-hoc não substituem os playbooks, são normalmente usados em casos que são raramente repetidos, como ações de horário de verão.

ANSIBLE

Playbook Hosts e Users

Para descrever as ações e tarefas que serão executadas em arquivos, Ansible usa a metáfora, a abstração chamada de **Playbook**. São basicamente uma lista de tarefas que serão executadas em grupos os hosts individuais. São escritos em YAML, facilitando o entendimento dos mais distantes do mundo do desenvolvimento, uma forma estruturada, simples e mais human-readable que outras abordagens como shell, chef, puppet...

Hosts e Users

Normalmente começamos a escrever um playbook declarando onde será executado e por quem, ou seja, qual o host ou grupo e qual o usuário remoto que executará a ação. Como o Ansible é baseado em SSH, podemos indicar qual usuário da máquina executara as ações.

```
- hosts: exemplo  
  remote_user: root
```

Aqui indicamos que o playbook será executado no grupo webservers pelo usuário root.

Também podemos conectar com outro usuário e dentro do sistema operacional mudar o privilégio, escolher o usuário e escolher o método de “mudança de privilégio”.

```
- hosts: exemplo  
  remote_user: usuario  
  become: yes  
  become_user: postgres  
  become_method: su
```

Become força a mudança e caso o usuário exija senha, o playbook deve ser executado com a opção -K ou -ask-become-pass.

ANSIBLE

Playbook Tasks e Modules

Tasks

Toda execução de um playbook contém uma lista de tarefas à serem cumpridas, em ordem, uma por vez, em todas as máquinas declaradas no hosts (inventory). **Task** é uma ação que será executada e deve conter um nome (name) descrevendo o que será realizado, e modulo que será usado.

Caso apresente algum erro, o host que apresentou falha é retirado da execução e nenhuma outra task subsequente é iniciada.

Modules

Ansible já vem por padrão com diversos “programas” prontos para serem usados de acordo com sua necessidade. **Modules** são desenvolvimentos reutilizáveis da maioria das tarefas o dia a dia (Files, Packaging, Commands, Database) com nível de maturidade suficiente para serem utilizados em produção.

Para usá-los é fácil, encontre um que atenda sua necessidade, entenda seus parâmetros e inclua-a na sua lista de tarefas, mais informações: https://docs.ansible.com/ansible/latest/modules/list_of_files_modules.html.

tasks:

- **name:** make sure apache is running

service:

- name:** httpd

- state:** started

Aqui temos uma lista de tarefas com um único passo, contendo seu nome e modulo usado.

O modulo service é usado para gerenciar os serviços da máquina e nesse caso são passados como argumentos o nome do serviço que será verificado e o estado desejado (started), garantindo que o apache está executando, mais informações: https://docs.ansible.com/ansible/latest/modules/service_module.html.

ANSIBLE

Playbook Tasks e Modules

tasks:

- **name:** ensure postgresql is at the latest version

yum:

name: postgresql

state: latest

- **name:** ensure that postgresql is started

service:

name: postgresql

state: started

Agora 2 tarefas, a primeira realiza a instalação do PostgreSQL com o módulo **yum** e segunda usa o módulo **service** para garantir que está serviço do PostgreSQL está executando, mais informações: https://docs.ansible.com/ansible/latest/modules/yum_module.html.

Mapeie suas necessidades, procure modulos que possam atende-las (dificilmente não existirá algo pronto) e caso não encontre, modulos podem ser construídos, assunto para outro momento.

ANSIBLE

Playbook Handlers

Algumas ações devem ser executadas a partir de eventos, após alguma mudança, condicionalmente. Para isso existe o **Notify**.

Notify aciona um **Handler**: uma lista de ações (igual a uma lista de tarefas) que é declarado de forma global no seu playbook e não pode conter nomes repetidos.

Então lembre-se que **Notify** aciona um **Handler** pelo nome (nomes não podem repetir) e é executado uma única vez no final do playbook.

Existem outras maneiras de executar handlers entre as tasks, mas não é o padrão, mais informações: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#handlers-running-operations-on-change.

tasks:

- **name:** restart everything
 - command:** echo "this task will restart the web services"
 - notify:** restart apache

handlers:

- **name:** restart apache
 - service:**
 - name:** apache
 - state:** restarted

ANSIBLE

Variáveis

Como em outros sistemas e linguagens, Ansible também trabalha com variáveis e existem diversas maneiras de acessar, declarar.

Variáveis devem sempre começar com letras ([a-zA-Z]) e podem conter números ([0-9]) e sublinhado (_).

Podemos declarar variáveis em diferentes lugares e diferentes maneiras.

Playbook Variables

A opção mais simples e declará-las no início do playbook, após indicar qual host será executado.

```
---  
- hosts: example  
vars:  
    chave: valor  
tasks:  
    ...
```

A sintaxe para declarar variáveis em um playbook, consequentemente em um arquivo YAML, é usando os dois pontos, chave: valor.

A opção mais simples e declará-las no início do playbook, após indicar qual host será executado.

```
# playbook.yml  
---  
- hosts: example  
vars_files:  
    - vars.yml  
tasks:  
    ...
```

```
# vars.yml  
---  
var: testando  
var2: testando2  
var3: testando3
```

ANSIBLE

Inventory Variables

Outra opção é declará-las no inventory, para um host específico ou em um grupo.

Hosts, ou Inventory, é um arquivo do tipo .ini e a forma para declarar variáveis é: chave=valor, usando o sinal de igual.

Host

[exemplos]

exemplo01 ansible_host=10.16.10.11 **ansible_network_os**=vyos **ansible_user**=my_vyos_user

exemplo02 ansible_host=10.16.10.12 **ansible_network_os**=vyos **ansible_user**=my_vyos_user

Group

[exemplos]

leaf01 ansible_host=10.16.10.11

leaf02 ansible_host=10.16.10.12

[exemplos:vars]

ansible_network_os=vyos

ansible_user=my_vyos_user

A documentação oficial elenca as melhores formas de utilizar variáveis.

Algumas alternativas entram em assuntos ainda não discutidos (roles, facts, registred...), mas vale a leitura e o entendimento, mais informações: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

ANSIBLE

Acessando Variáveis e Templating

Acessando Variáveis

No geral para acessar variáveis usamos a notação: `{{ variavel }}`, em um playbook (arquivo YAML) é necessário encapsular com aspas duplas `"{{ variavel }}"`. Sem as aspas o YAML irá entender que é um dicionário/objeto (`{{nome: guilherme}}`).

Templating (Jinja2)

Muitas das ações do dia a dia envolvem alterar arquivos, uma alternativa é usar templates no formato .j2 (Jinja2).

São arquivos moldes e podem acessar variáveis, aplicar filtros, realizar condições, loops, expressões regulares - https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html.

```
[client]
user={{ root.login }}
password={{ root.password }}
```

O exemplo acima é uma das maneiras de criar o arquivo `my.cnf` do MySQL, que configura a autenticação do client do banco.

PLANEJAMENTO

Etapas Detalhadas

Conforme as necessidades forem surgindo consequentemente a complexidade das automações aumentarão e mais funcionalidades do Ansible serão usadas para atender os diferentes cenários.

Até aqui já cobrimos os conhecimento básicos de Ansible para o cenário proposto da forma mais simples e direta.

Para escrever um playbook, pense em todas as etapas necessárias e onde devem ser executadas:

1. Instalar MySQL server e dependências - database.
2. Atualizar arquivo de autenticação client - database.
3. Atualizar senhas do usuário Root do MySQL - database.
4. Modificar arquivo de configuração MySQL, habilitando conexões remotas - database.
5. Reiniciar MySQL - database.
6. Garantir que o MySQL está rodando - database.
7. Criar banco de dados Wordpress - database.
8. Criar usuário Wordpress no MySQL - database.
9. Instalar Apache2 e dependências - web.
10. Iniciar e habilitar o serviço do Apache2 - web.
11. Download pacote Wordpress - web.
12. Extrair pacote - web.
13. Atualizar site padrão do Apache2 - web.
14. Atualizar arquivo de configuração Wordpress - web.
15. Reiniciar Apache2 - web.

ANSIBLE

Automação Banco de Dados

Comece pelo básico e faça tudo em um arquivo só, depois iremos avançar e descobrir como podemos organizar e separar de uma forma mais reusável o projeto e os arquivos.

O arquivo “/shared/ansible/playbook-implementation/config.yml” implementa todas as etapas descritas no planejamento.

As etapas de 1 à 8 serão executadas nos servidores do grupo db, como sudo e com usuário root.

Algumas etapas exigem definição de parâmetros como: usuários, senhas, hosts e serão declaradas direto no arquivo.

Cada etapa faz uso de módulos prontos que realizam a maioria das funções principais.

A primeira etapa deve instalar os pacotes: **mysql-server** e **python3-mysqldb**.

O módulo de instalação de pacotes do Ubuntu é o apt, onde passo os nomes dos pacotes em forma de lista no parâmetro **name**, também indico a última versão com os parâmetros: **state: latest** e parâmetro **update_cache: true** para atualizar os pacotes do sistema operacional.

```
! config.yml X
shared > ansible > playbook-implementation > ! config.yml
1  ---
2  # Etapas do servidor database
3  - hosts: db
4    vars:
5      # Habilitar conexão entre máquinas virtuais para possível troubleshooting
6      con_hosts:
7        - 192.168.56.67
8        - 192.168.56.66
9        - 192.168.56.65
10     db: wordpress
11     root: ...
15    wp: ...
19    become: yes
20    become_user: root
21    tasks:
22      # Etapa 1 - Instalar MySQL server e dependências
23      - name: install mysql
24        apt:
25          name:
26            - mysql-server
27            - python3-mysqldb
28          update_cache: true
29          state: latest
30      # Etapa 2 - Atualizar arquivo de autenticação client
31      - name: copy the root credentials as .my.cnf file...
38      # Etapa 3 - Atualizar senhas do usuário Root do MySQL
39      - name: update root password for all accounts...
```

ANSIBLE

Automação Servidor Web

Após a configuração do banco de dados, configure o servidor web e instale o WordPress.

O mesmo arquivo “/shared/ansible/playbook-implementation/config.yml”, implementa todas as etapas descritas no planejamento.

As etapas de 9 à 15 serão executadas nos servidores do grupo web, como sudo.

Algumas delas exigem definição de parâmetros como: usuários, senhas e hosts e serão declaradas direto no arquivo.

```
! config.yml X
shared > ansible > playbook-implementation > ! config.yml

92  # Etapas do servidor web
93  - hosts: server
94    become: yes
95    vars:
96      db:
97        name: wordpress
98        host: 192.168.56.67
99      wp:
100        name: wordpress
101        login: wordpress
102        password: wordpress
103    tasks:
104      # Etapa 9 - Instalar Apache2 e dependências
105      > - name: install apache2 and php...
119      # Etapa 10 - Iniciar e habilitar o serviço do Apache2
120      > - name: start apache service...
125      # Etapa 11 - Download pacote Wordpress
126      > - name: download wordpress...
131      # Etapa 12 - Extrair pacote
132      > - name: extract wordpress...
137      # Etapa 13 - Atualizar site padrão do Apache2
138      > - name: update default apache site...
143      # Etapa 14 - Atualizar arquivo de configuração Wordpress
144      > - name: copy wordpress config file...
149      # Etapa 15 - Reiniciar Apache2
150      > - name: restart apache2...
```

ANSIBLE

Preparando Ambiente de Desenvolvimento

Com os playbooks prontos, vamos executá-los na workspace e ver se tudo ocorre conforme o planejado.

Garanta que o ambiente de desenvolvimento Vagrant está criado e funcional com:

`vagrant status`

```
PS C:\Users\guilh\config-ansible-vagrant> vagrant status
Current machine states:

workspace      not created (virtualbox)
web            not created (virtualbox)
database       not created (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Lembre-se que caso não estejam criadas, inicie as máquinas virtuais com:

`vagrant up`

```
PS C:\Users\guilh\config-ansible-vagrant> vagrant status
Current machine states:

workspace      running (virtualbox)
web            running (virtualbox)
database       running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```


ANSIBLE

Conectado e Verificando o Workspace

Após isso conecte-se na máquina virtual workspace com:

```
vagrant ssh workspace
```

Verifique se o ansible foi instalado corretamente:

```
ansible --version
```

```
PS C:\Users\guilh\config-ansible-vagrant> vagrant ssh workspace
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Nov 11 18:21:28 UTC 2019

System load: 0.06               Processes:            98
Usage of /:  13.2% of 9.63GB    Users logged in:     0
Memory usage: 16%              IP address for enp0s3: 10.0.2.15
Swap usage:  0%                IP address for enp0s8: 192.168.56.65

33 packages can be updated.
17 updates are security updates.

vagrant@workspace:~$ ansible --version
ansible 2.9.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/vagrant/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.15+ (default, Oct 7 2019, 17:39:04) [GCC 7.4.0]
```


ANSIBLE

Verificando Workspace

Verifique se o arquivo hosts do Ansible está correto, contendo as conexões das máquinas virtuais web e database.

```
cat /etc/ansible/hosts
```

```
vagrant@workspace:~$ cat /etc/ansible/hosts
[server]
web ansible_ssh_private_key_file=~/.ssh/private ansible_host=192.168.56.66 ansible_user=root ansible_python_interpreter=/usr/bin/python3
(db)
database ansible_ssh_private_key_file=~/.ssh/private ansible_host=192.168.56.67 ansible_user=root ansible_python_interpreter=/usr/bin/python3
```

Estamos usando o arquivo hosts no caminho padrão de forma global, outras formas podem ser utilizadas para organizar às máquinas que serao administradas pelo Ansible.

Arquivo com dois grupos: server e db. Cada um pode conter uma listá de máquinas (no exemplo só uma) sendo cada linha contendo:

- Nome;
- Chave privada para conexão;
- IP/Host;
- Usuário que realizará a conexão;
- Caminho do interpretador Python (requisito Ansible);

Mais informações sobre as opções de parâmetros no Hosts:

ANSIBLE

Executando o Playbook

Agora execute o playbook que está no caminho criado entre seu computador e a máquina virtual. Para executar o playbook use o comando `ansible-playbook` passando o caminho do seu arquivo YAML contendo sua automação.

```
ansible-playbook shared/ansible/playbook-implementation/config.yml -v
```

O argumento `-v` é usado para obter mais detalhes da execução do playbook, podendo ser usado até 4x, ou seja: `-vvvv`.

Com isso serão executados todos os passos nos dois alvos: `web` e `database`.

```
vagrant@workspace:~$ ansible-playbook shared/ansible/playbook-implementation/config.yml -v
Using /etc/ansible/ansible.cfg as config file

PLAY [db] *****

TASK [Gathering Facts] *****
ok: [database]
```

Ao final será exibido um resumo sobre todas as alterações e ações listadas em seu playbook, com os status de tudo que aconteceu na execução como falhas e erros de comunicação.

```
PLAY RECAP *****
database      : ok=9    changed=7    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
web           : ok=8    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```


VALIDAÇÃO

Verificando Execução

Com tudo executado, sem falhas, abra um navegador no endereço: <http://localhost:7070> (porta redirecionada via Vagrantfile).

A página de configuração do usuário administrador do Wordpress irá carregar indicando que a automação ocorreu com sucesso.

← → ↻ ⓘ localhost:7070/wp-admin/install.php



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password
Strong
Important: You will need this password to log in. Please store it in a secure location.

Your Email
Double-check your email address before continuing.

Search Engine Visibility ☐ Discourage search engines from indexing this site
It is up to search engines to honor this request.

CONCLUSÃO

Considerações

Os assuntos cobertos são suficientes pra dar introdução aos princípios básicos de Ansible: automações simples, diretas e não reutilizáveis.

Assuntos como: Roles, Condições, Vault, Lookups, Async, Facts serão tratados em outro momento.

Sinta-se livre para contrubuir com o repositório, implementar melhorias e submeter requests.

Alguns pontos podem ser melhorados sem utilizar assuntos não cobertos, o diretório playbook-implementation mostra a forma mais simples possível de criar um playbook em um único arquivo.

Outras versões desse mesmo cenário devem ser criadas seguindo melhoras práticas, como por exemplo: roles-implementation.

Agradeço á Sonda por fomentar a transformação digital olhando pra dentro de casa, incentivando o estudo e o compartilhamento de conhecimento, entendendo que a mudança cultural focada em pessoas é o ponto chave.

Dúvidas, sugestões, críticas, comentários:

 <https://twitter.com/guilhermepozo>

 <https://linkedin.com/in/guilherme-pozo-037b41a5>

 <https://github.com/guilhermepozo>

