

Infraestrutura como Código

Provisionamento

Michel Vasconcelos
michel.vasconcelos@gmail.com

Ferramentas

- Virtualização e Containers
 - Vagrant
 - Packer
 - Docker
- Provisionamento
 - Cloud Formation
 - Terraform

Vagrant

- Construção e gestão de máquinas virtuais
 - Baixando templates
 - Iniciando e Parando VMs
 - etc
- Facilita o fluxo de trabalho para desenvolvedores
 - Ex.: vários ambientes de desenvolvimento, um em cada VM instanciada

ONE DOES NOT SIMPLY SAY

"WELL, IT WORKED ON MY MACHINE"

Uso

- Compartilhamento de ambiente virtualizado em uma equipe
- Uma máquina física, vários ambientes de desenvolvimento
- Configuração de várias VMs simultaneamente

Conceitos

- Vagrantfiles
 - Boxes
- Cliente de linha de comando
- Providers
- Provisioning

Vagrantfile

- Descreve a máquina virtual
 - Boxes
- Versionável
- Sintaxe Ruby

Vagrantfile

- *Vagrantfile*: Nome padrão para o arquivo
 - 1 arquivo por projeto
 - `home/michel/projeto/Vagrantfile`
 - `home/michel/Vagrantfile`
 - `home/Vagrantfile`
 - `/Vagrantfile`
 - `VAGRANT_CWD` muda o diretório de início
- Ordem de carregamento
 - Arquivo empacotado com a VM
 - Arquivo padrão na sua home (`~/.vagrant.d`)
 - Arquivo do projeto

Boxes

- Forma de empacotamento padrão no Vagrant
 - Resumindo... uma VM compactada para seu uso
 - Template
- Há um repositório público e oficial para *boxes* mantido pela Hashicorp
 - <https://app.vagrantup.com/>
- Base Boxes
 - Criadas do zero

Packer

- Criar imagens de máquinas virtuais para diversas plataformas
 - AMIs
 - VMKD
 - OVF's
 - ISO
 - etc
- Uso?!

Configurações

- `vagrant.configure`
 - Retrocompatibilidade entre versões do vagrant
 - Duas versões disponíveis: 1 e 2
- Você pode mesclar duas versões, mas em seções diferentes

```
Vagrant.configure("1") do |config|  
  # v1 configs...  
end  
  
Vagrant.configure("2") do |config|  
  # v2 configs...  
end
```

Namespaces

- config
 - vm
 - ssh
 - winrm
 - winssh
 - vagrant

config.vm

- Define todas as configurações da VM
 - rede, SO, etc
 - Principal namespace
- Principais configurações
 - box
 - communicator (padrão: ssh)
 - guest (padrão: linux)
 - hostname
 - network
 - provider
 - provisioning

config.ssh

- Configura o acesso à máquina via SSH
 - Geralmente as configurações padrão funcionam bem
- Principais configurações
 - username (padrão: vagrant)
 - password
 - port (padrão: 22)
 - shell (padrão: bash)
 - extra_args

Vagrantfile

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

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

  config.vm.box = "debian-7.2.0-amd64"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  config.vm.network :private_network, ip: "10.10.10.2"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  config.vm.synced_folder ".", "/vagrant", nfs: true

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  config.vm.provider :virtualbox do |vb|

    # Use VBoxManage to customize the VM. For example to change memory:
    vb.customize ["modifyvm", :id, "--memory", "2048"]
    vb.customize ["modifyvm", :id, "--vram", "10"]
    vb.customize ["setextradata", :id, "VBoxInternal/Devices/ahci/0/LUN#[0]/Config/IgnoreFlu
  end

  config.vm.provision :chef_client do |chef|
    chef.chef_server_url = "https://api.opscode.com/organizations/ORGNAME"
    chef.validation_key_path = "ORGNAME-validator.pem"
  end
end
```

Cliente de linha de comando

```
michel ~ vagrant https://www.vagrantup.com/docs/cli
Usage: vagrant [options] <command> [<args>]

-v, --version      Print the version and exit.
-h, --help         Print this help.

Commands (CLI)
Common commands:
  box                manages boxes: installation, removal, etc.
  destroy            stops and deletes all traces of the vagrant machine
  global-status      outputs status Vagrant environments for this user
  halt               stops the vagrant machine
  help               shows the help for a subcommand
  init               initializes a new Vagrant environment by creating a Vagrantfile
  login               log in to HashiCorp's Vagrant Cloud
  package            packages a running vagrant environment into a box
  plugin             manages plugins: install, uninstall, update, etc.
  port               displays information about guest port mappings
  powershell         connects to machine via powershell remoting
  provision           provisions the vagrant machine
  push               deploys code in this environment to a configured destination
  rdp                 connects to machine via RDP
  reload             restarts vagrant machine, loads new Vagrantfile configuration
  resume             resume a suspended vagrant machine
  snapshot            manages snapshots: saving, restoring, etc.
  ssh                connects to machine via SSH
  ssh-config         outputs OpenSSH valid configuration to connect to the machine
  status             outputs status of the vagrant machine
  suspend            suspends the machine
  up                 starts and provisions the vagrant environment
  validate           validates the Vagrantfile
  version            prints current and latest Vagrant version

For help on any individual command run `vagrant COMMAND -h`
```


vagrant box

- Gerencia boxes
 - Adiciona, remove, etc boxes no repositório definido por um endereço
- `vagrant box <subcomando> endereço`
 - `add`
 - `list`
 - `outdated`
 - `prune`
 - `remove`
 - `repackage`
 - `update`

vagrant init

- Inicializa o diretório atual como um ambiente para o Vagrant e cria um arquivo de configuração
 - 1º argumento: vm.box
 - 2º argumento: vm.box_url
- `vagrant init [name [url]]`
 - `vagrant init hashicorp/precise64`

vagrant package

- Empacota uma VM existente como um box
 - Providers: VirtualBox e Hyper-V
- `vagrant package [name|id]`
 - `--output NAME`
 - `--vagrantfile FILE`

vagrant plugin

- Gerencia plugins
- Subcomandos
 - expunge
 - install
 - license
 - list
 - repair
 - uninstall
 - update

vagrant port

- Lista as portas que estão mapeadas entre o hospedeiro e hospedado

```
$ vagrant port
  22 (guest) => 2222 (host)
  80 (guest) => 8080 (host)
```

- Opções
 - `--guest PORT`

```
$ ssh -p $(vagrant port --guest 22)
```

vagrant ssh

- Conecta em uma máquina gerenciada pelo vagrant e permite acesso ao *shell*
- `vagrant ssh [name|id] [-- extra_ssh_args]`
 - `-c` COMANDO

vagrant up

- Cria e configura uma máquina virtual conforme descrito no arquivo de configuração
 - Comando mais importante e usado da ferramenta
- `vagrant up [name|id]`
 - `name`
 - `id`
 - `--provider`
 - `--[no-]provision`
 - `-- provision-with`

Outros comandos

- vagrant
 - snapshot
 - status
 - suspend
 - validate
 - reload
 - resume
 - etc

<https://www.vagrantup.com/docs/cli/>

Providers

- Hipervisores \ Gerenciadores de VM \ Container
- Providers “embutidos”
 - **Virtualbox**
 - Hyper-V
 - Docker
- Instalação
 - `vagrant plugin`

Providers

- Boxes são particulares para cada provider
 - Você pode ter uma box disponível para mais de um provedor (com o mesmo nome)

```
$ vagrant box list  
precise64 (virtualbox)  
precise64 (vmware_fusion)
```

- VAGRANT_DEFAULT_PROVIDER
 - Variável de ambiente que define o provider padrão

Provisioner

- Realiza a configuração das VMs criadas
 - instala software
 - altera arquivos e configurações
 - etc
- Torna o processo
 - Repetível
 - Escalável
 - Simples

Provisioner

- Realiza a configuração das VMs criadas
 - instala software
 - altera arquivos e configurações
 - etc
- Torna o processo
 - Repetível
 - Escalável
 - Simples

Bem melhor que conectar-se via ssh e instalar tudo “na mão”

Provisioning

- Acontece
 - 1º vagrant up
 - vagrant provisioning
 - opção *--provisioning*
- `vm.provision`
 - *run: "always"*
- Métodos
 - Arquivo
 - Shell script
 - Ansible, Chef, Salt, etc

```
Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline: "echo foo"

  config.vm.define "web" do |web|
    web.vm.provision "shell", inline: "echo bar"
  end

  config.vm.provision "shell", inline: "echo baz"
end
```

File Provisioner

- Forma simples de copiar arquivos ou diretórios para a máquina convidada (VM)
 - replicação de arquivos de configuração
 - Arquivos copiados não serão sincronizados após cópia

```
Vagrant.configure("2") do |config|  
  # ... other configuration  
  
  config.vm.provision "file", source: "~/.gitconfig", destination: ".gitconfig"  
end
```

Shell Provisioner

- Executa um shell script na VM criada
 - Ideal para quem já trabalha com scripts

- Inline
 - Comando inline no Vagrantfile

```
Vagrant.configure("2") do |config|  
  config.vm.provision "shell",  
    inline: "echo Hello, World"  
end
```

- Externo
 - script definido em arquivo

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

Shell Provisioner

- Outras opções
 - args: argumentos para o script
 - env: lista de tuplas chave-valor
 - privileged
 - upload_path
 - md5

Docker Provisioner

- Automaticamente instala e configura o Docker na VM

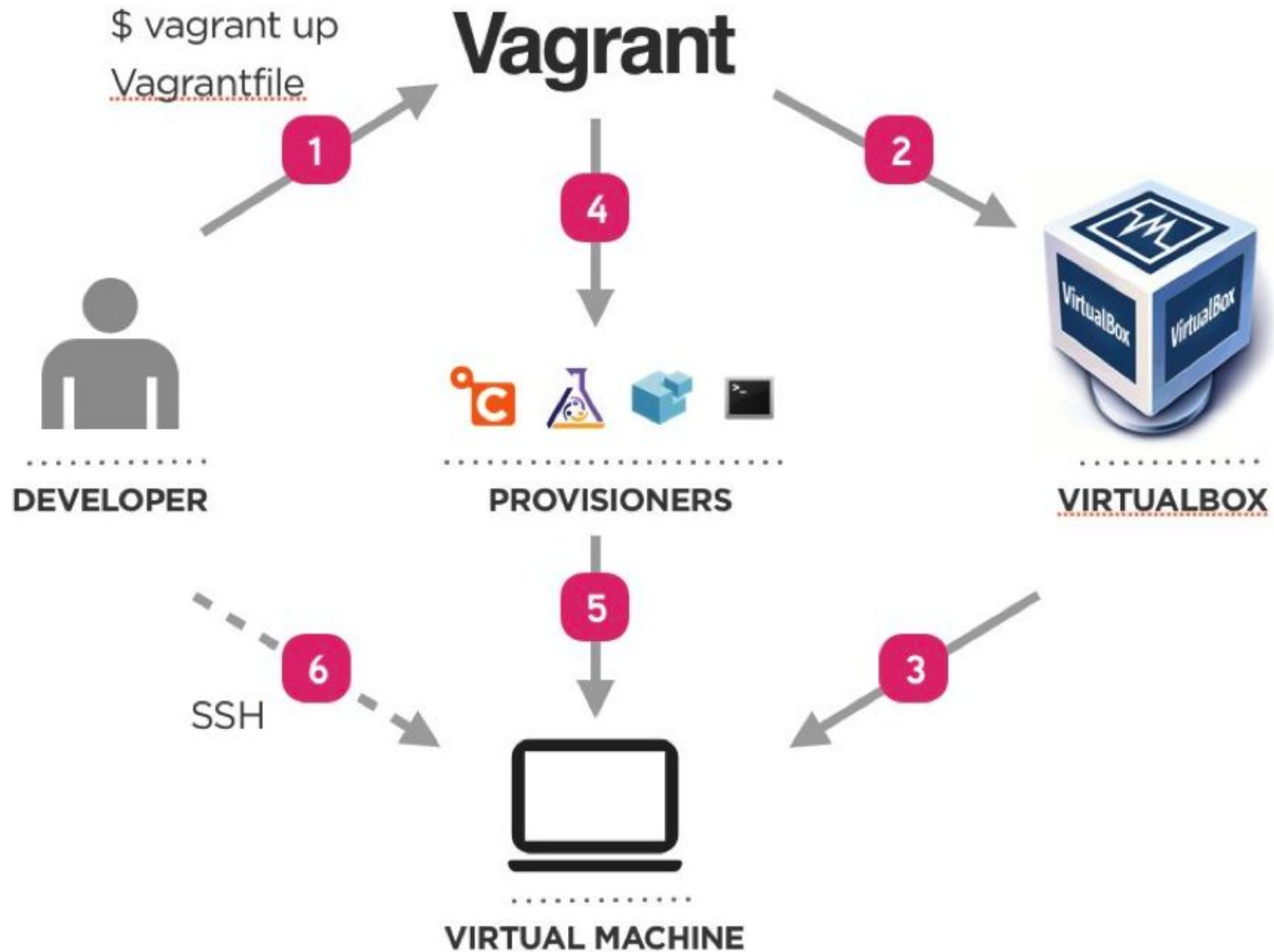
- Opções
 - images
 - build_image
 - pull_images
 - run
 - image
 - cmd
 - args
 - restart

```
Vagrant.configure("2") do |config|
  config.vm.provision "docker" do |d|
    d.run "ubuntu",
      cmd: "bash -l",
      args: "-v '/vagrant:/var/www'"
  end
end
```

Outros

- Chef
 - Solo, Zero ou Client
- Ansible
 - ansible ou local
- Puppet
 - Apply ou Agent
- Salt

Fluxo



Dicas

- Log detalhado
 - `VAGRANT_LOG`
- Você pode definir múltiplas máquinas em um único arquivo
- Use vagrant snapshot como cache de construção
- Você pode customizar o provisionamento, provedores e também criar plugins

Exercício 1

Crie uma máquina virtual contendo uma instalação do cliente de linha de comando da AWS e da ferramenta Terraform.

Você pode seguir as orientações contidas no repositório:

<https://github.com/michelav/espec-iac>

Cloud Formation

Características

- Serviço de provisionamento de infraestrutura da AWS
 - Permite focar na aplicação e não em infra
- Configuração de várias VMs simultaneamente
- Integração com mecanismos de CI \ CD da AWS
- Possui uma ferramenta de design

Conceitos

- Templates
- Stacks
- Change Sets

Templates

- Descreve uma coleção de recursos AWS e respectivas propriedades que serão utilizados em sua Infraestrutura
- Arquivo Texto (Json ou YAML)
 - `AWSTemplateFormatVersion`
 - `Metadata`
 - `Parameters`
 - `Resources`
 - `etc`
- Há [snippets de template](#) para diversos recursos na AWS

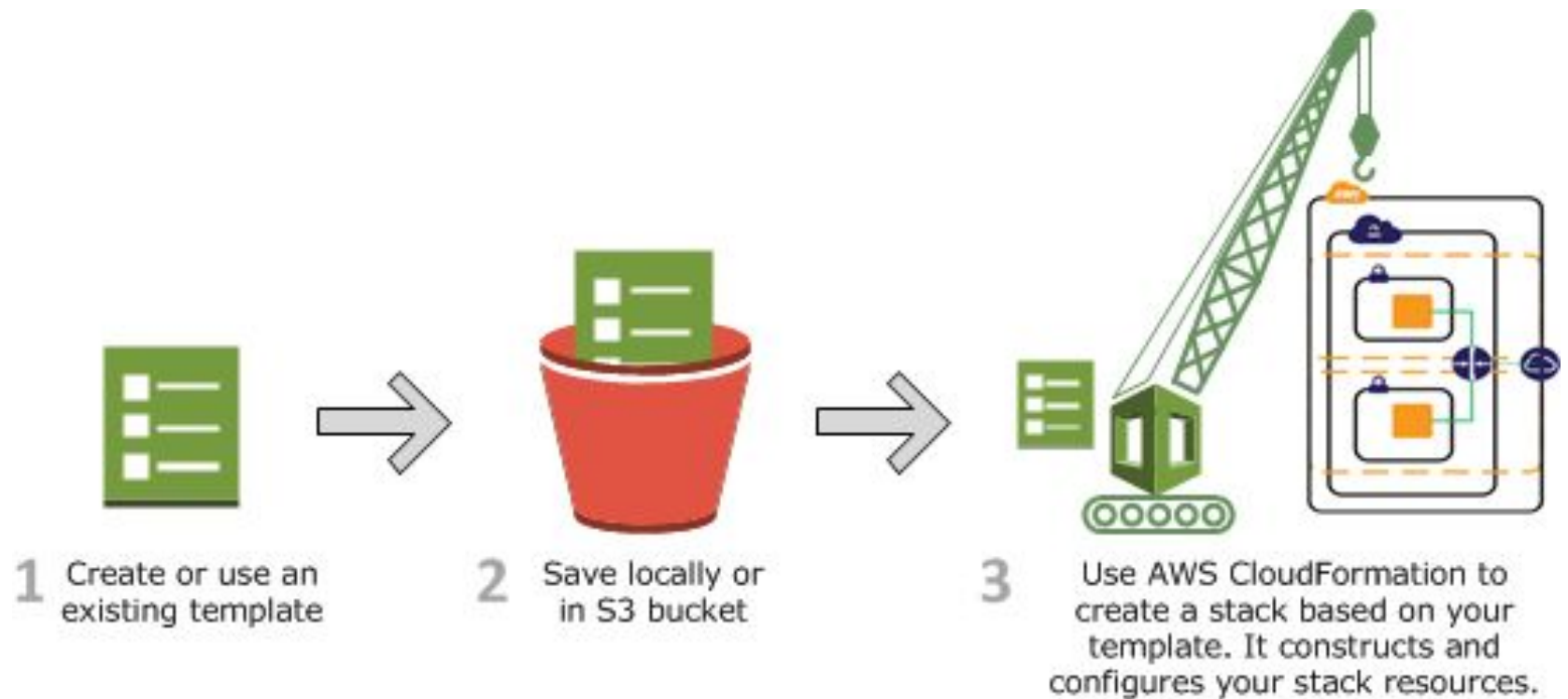
Stack

- Compreende uma coleção de recursos AWS, sendo descrita por um template
 - Recursos que serão implantados juntos, como um grupo
 - Ex: Uma aplicação Web utiliza diversos recursos (servidor web, BD, rotas, etc) em várias camadas. Você pode criar ou destruir todos esses recursos (em conjunto) por meio de uma *stack*
- Meio
 - AWS Console, API e AWS CLI
- Atualizações
 - Atualização direta vs Change Sets

Change Sets

- Forma de prever as alterações que serão realizadas nos recursos AWS gerenciados pelo Cloud Formation
- Também formado por arquivo Json ou YAML
- Utilize quando quiser avaliar como a AWS realizará as modificações necessárias em sua Infraestrutura

Criando Stacks



Definindo um Template

- AWSTemplateFormatVersion

- "AWSTemplateFormatVersion" : "2010-09-09"

- Parameters

- Parâmetros que devem ser informados na criação da *stack*

```
---
AWSTemplateFormatVersion: "2010-09-09"
Description: "A sample template"
Parameters:
  InstanceType:
    Description: "Tipo de instancia"
    Type: "String"
    Default: "t2.micro"
    AllowedValues:
      - t1.micro
      - t1.small
      - t2.micro
    ConstraintDescription: "Deve ser uma instancia valida"
  KeyName:
    Description: "EC2 Key Pair"
    Type: "AWS::EC2::KeyPair::KeyName"
    ConstraintDescription: "Deve ser uma Key Pair valido"
```

Definindo um Template

- Resources

- Onde são definidos efetivamente os recursos que serão utilizados
- O nome do recurso é um nome lógico que pode ser referenciado em outras partes do arquivo
- Um recurso pode ter diversas propriedades conforme ele necessitar ser configurado

```
Resources:  
  Logical ID:  
  Type: Resource type  
  Properties:  
    Set of properties
```

```
Resources:  
  MyWebServer:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: ami-3885d854  
      InstanceType:  
        - Ref: "InstanceType"  
      KeyName:  
        - Ref: "KeyName"
```

Definindo um Template

- Outputs
 - Valores que podem ser importados em outras stacks
 - Respostas ao processo de criação
 - Status no Console

```
Outputs:
  Logical ID:
    Description: Information about the value
    Value: Value to return
    Export:
      Name: Value to export
```

```
Outputs:
  MyServerURL:
    Description: URL do servidor web
    Value:
      Fn::Join:
        - ''
        - - http://
          - Fn::GetAtt:
              - MyWebServer
              - PublicDnsName
```

E muito mais...

```
---  
AWSTemplateFormatVersion: "version date"  
  
Description:  
  String  
  
Metadata:  
  template metadata  
  
Parameters:  
  set of parameters  
  
Mappings:  
  set of mappings  
  
Conditions:  
  set of conditions  
  
Transform:  
  set of transforms  
  
Resources:  
  set of resources  
  
Outputs:  
  set of outputs
```


Criando uma Stack

- Linha de comando
 - `aws cloudformation create-stack`
 - `aws cloudformation list-stacks`
 - `aws cloudformation describe-stacks`
 - `aws cloudformation describe-stack-events`

Criando uma Stack

aws
Console Home

Services ▾ Resource Groups ▾

CloudFormation ▾ Stacks

Create Stack ▾ Actions ▾ Design template

Filter: Active ▾ By Stack Name

Showing 0 stacks

Create a stack

AWS CloudFormation allows you to quickly and easily deploy your infrastructure resources and applications on AWS. You can use one of the templates we provide to get started quickly with applications like WordPress or Drupal, one of the many sample templates or create your own template.

You do not currently have any stacks. Choose **Create new stack** below to create a new AWS CloudFormation stack.

Create new stack

Create a StackSet

A StackSet is a container for AWS CloudFormation stacks that lets you provision stacks across AWS accounts and regions by using a single AWS CloudFormation template.

Create new StackSet

Criando uma Stack

Create Stack ▾

Actions ▾

Design template

Filter: Active ▾

By Stack Name

	Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/>	my-stack	2018-06-19 01:56:59 UTC-0300	CREATE_COMPLETE	A sample template

Overview Outputs Resources Events Template Parameters Tags Stack Policy Change Sets Rollback Triggers

2018-06-19	Status	Type	Logical ID	Status Reason
▶ 01:57:43 UTC-0300	CREATE_COMPLETE	AWS::CloudFormation::Stack	my-stack	
▶ 01:57:39 UTC-0300	CREATE_COMPLETE	AWS::EC2::Instance	MyDBServer	
▶ 01:57:39 UTC-0300	CREATE_COMPLETE	AWS::EC2::Instance	MyWebServer	
▶ 01:57:05 UTC-0300	CREATE_IN_PROGRESS	AWS::EC2::Instance	MyDBServer	Resource creation Initiated
▶ 01:57:05 UTC-0300	CREATE_IN_PROGRESS	AWS::EC2::Instance	MyWebServer	Resource creation Initiated
▶ 01:57:04 UTC-0300	CREATE_IN_PROGRESS	AWS::EC2::Instance	MyDBServer	
▶ 01:57:03 UTC-0300	CREATE_IN_PROGRESS	AWS::EC2::Instance	MyWebServer	

Criando uma Stack

Launch Instance

▼

Connect

Actions ▼

🔍

Filter by tags and attributes or search by keyword

?

⏪

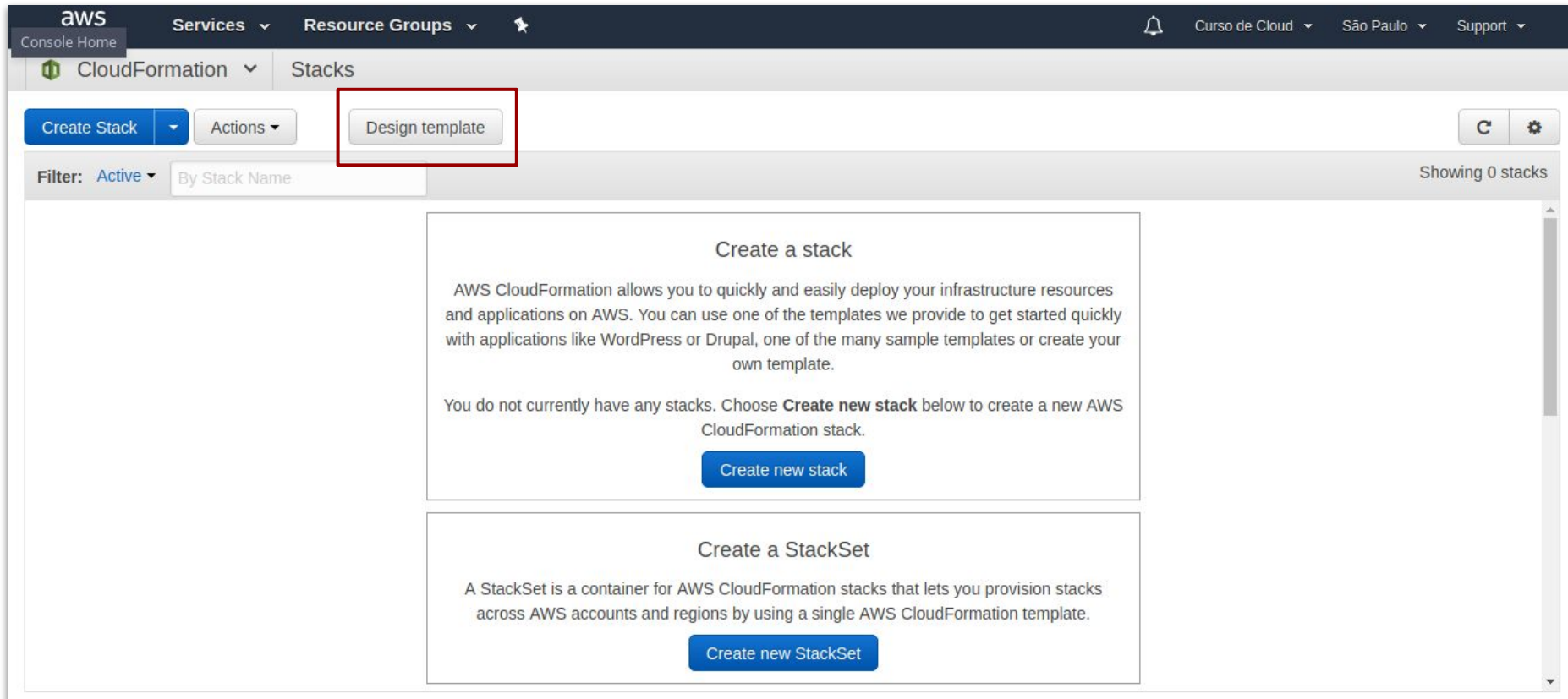
<

1 to 4 of 4

>

<input type="checkbox"/>	Name ▼	Instance ID ▲	Instance Type ▼	Availability Zone ▼	Instance State ▼	Status Checks ▼	Alarm Status	Public DNS (IPv4) ▼
<input type="checkbox"/>		i-02c494fabfc14899c	t2.micro	sa-east-1a	<div>terminated</div>		None	
<input type="checkbox"/>		i-069b6047141ac4daf	t2.micro	sa-east-1a	<div>terminated</div>		None	
<input type="checkbox"/>		i-0780d8277ca288f8d	t2.micro	sa-east-1a	<div>running</div>	<div>⌚ Initializing</div>	None	ec2-54-233-125-220.sa...
<input type="checkbox"/>		i-085b9cd2458348286	t2.micro	sa-east-1a	<div>running</div>	<div>⌚ Initializing</div>	None	ec2-18-231-115-209.sa...

Importante



The screenshot displays the AWS CloudFormation console interface. At the top, the navigation bar includes the AWS logo, 'Console Home', and tabs for 'Services', 'Resource Groups', and 'Stacks'. The 'Stacks' tab is active, showing a 'Create Stack' button and an 'Actions' dropdown menu. The 'Design template' option is highlighted with a red rectangular box. Below the navigation bar, a filter section shows 'Filter: Active' and a search input 'By Stack Name'. The main content area is titled 'Showing 0 stacks' and contains two sections: 'Create a stack' and 'Create a StackSet'. The 'Create a stack' section explains that AWS CloudFormation allows for quick deployment of infrastructure resources and applications on AWS, and provides a 'Create new stack' button. The 'Create a StackSet' section explains that a StackSet is a container for AWS CloudFormation stacks that lets you provision stacks across AWS accounts and regions by using a single AWS CloudFormation template, and provides a 'Create new StackSet' button.

aws
Console Home Services Resource Groups

CloudFormation Stacks

Create Stack Actions Design template

Filter: Active By Stack Name Showing 0 stacks

Create a stack

AWS CloudFormation allows you to quickly and easily deploy your infrastructure resources and applications on AWS. You can use one of the templates we provide to get started quickly with applications like WordPress or Drupal, one of the many sample templates or create your own template.

You do not currently have any stacks. Choose **Create new stack** below to create a new AWS CloudFormation stack.

Create new stack

Create a StackSet

A StackSet is a container for AWS CloudFormation stacks that lets you provision stacks across AWS accounts and regions by using a single AWS CloudFormation template.

Create new StackSet

Dicas

- Não se acanhe de usar a ferramenta de design
- Use o IAM para controle de acesso
- Reuse os templates

Dicas

- Não coloque as credenciais no arquivo de template
- Gerencie os recursos pelo CloudFormation
- Crie Change Sets

Exercício 2

Crie um arquivo template para o Cloud Formation e siga o passo a passo descrito no repositório para criar uma stack na AWS.

<https://github.com/michelav/espec-iac/tree/master/cf/ex02>

Exercício 3

Importe o arquivo de template no ferramenta de design do Cloud Formation e ajuste-a para que a stack possua três servidores.

Terraform

Características

- **Provisionamento de Infra**

- Construção
- Alteração
- Versionamento
- etc

- **Diversos Providers**

- AWS
- Google
- Azure
- Digital Ocean
- etc

Características

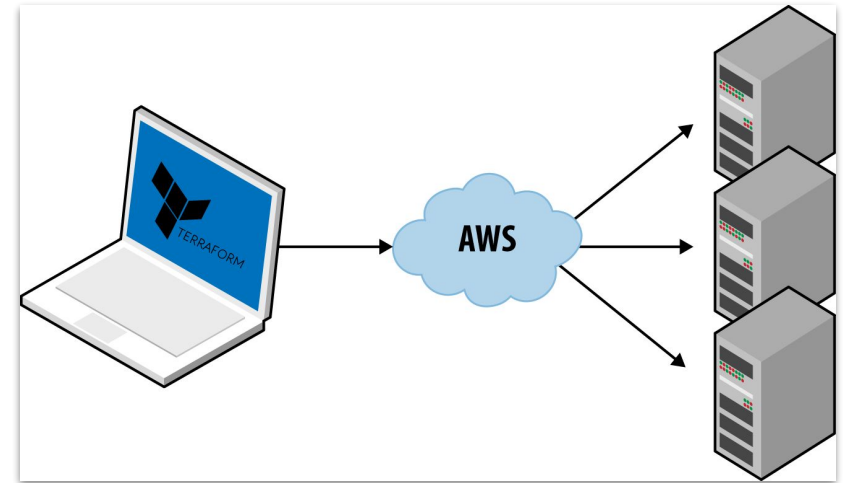
- Open Source
 - Mas possui versão enterprise
- Facilidade para se conseguir Infra imutável
 - Mudanças implicam em novos servidores \ containers

- Declarativa

```
resource "aws_instance" "example" {  
  count = 10  
  ami = "ami-v1"  
  instance_type = "t2.micro"  
}
```

Características

- Sem servidor Master e sem agentes
- Comunidade atuante (e crescente)
- Pouca maturidade
 - Versão 0.11.7



Características

- Extensível
 - Plugins
 - Modules
- Implementado em Go
 - Único executável e *footprint* pequeno

Comparativo

	Chef	Puppet	Ansible	SaltStack	CloudFormation	Terraform
Code	Open source	Open source	Open source	Open source	Closed source	Open source
Cloud	All	All	All	All	AWS only	All
Type	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	Orchestration	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Mutable	Immutable	Immutable
Language	Procedural	Declarative	Procedural	Declarative	Declarative	Declarative
Architecture	Client/Server	Client/Server	Client-Only	Client/Server	Client-Only	Client-Only

Elementos

- Configurações
- Comandos
- Providers
- Provisioners
- Recursos avançados

Configurações

- Terraform utiliza arquivos texto para definir sua Infraestrutura
 - Json
 - Formato Terraform (similar ao Json)
- Arquivos carregador por ordem alfabética
 - extensão tf ou tf.json
- Arquivos são agrupados em um único modelo
 - Recursos com mesmo nome não são permitidos

Sintaxe

- HCL
 - Hashicorp Configuration Language
 - <https://github.com/hashicorp/hcl>
- Referência básica
 - # : comentários
 - = : atribuição (chave - valor)
 - \${ } : interpolação de valores
 - Listas e Mapas também

Interpolação

- **Substituição de valores**

- `${var.my-var}`

- **Suporta expressões matemáticas e condicionais**

- `${count.index + 1}`

- **Prefixo VAR**

- Permite o uso de alguma variável
 - `variable "my-var" {...}`
 - `${var.my-var}`
- Suporta *strings*, *listas* e *mapas*
- Pode ser definida por linha de comando ou ambiente

Interpolação

- Prefixo SELF
 - Interpola informações do próprio recurso
 - `${self.private_ip}`
- Funções pré-construídas
 - permite disparar ações no arquivo de configuração
 - `${file("path.txt")}`
 - `${basename(path)}`

<https://www.terraform.io/docs/configuration/interpolation.html>

Interpolação

- Overrides

- Substituir uma informação sem comprometer o arquivo de configuração inicial
- O novo arquivo deve ser override ou terminar em `_override`
 - `override.tf`
 - `override.tf.json`
 - `temp_override.tf`

main.tf

```
resource "aws_instance" "web" {  
  ami = "ami-408c7f28"  
}
```

override.tf

```
resource "aws_instance" "web" {  
  ami = "foo"  
}
```

Resources

- Permite a definição dos recursos de sua Infra
 - Cada Provedor possui seus tipos de recurso específicos
- Meta-parâmetros
 - count
 - depends_on
 - provider

```
resource TYPE NAME {  
    CONFIG ...  
    [count = COUNT]  
    [depends_on = [NAME, ...]]  
    [provider = PROVIDER]  
  
    [LIFECYCLE]  
  
    [CONNECTION]  
    [PROVISIONER ...]  
}
```

Configuração Provider

- Permite configurar um provedor
 - Autenticação
 - URLs
 - etc
- Uma configuração pode ter múltiplas seções de Provider

```
provider "aws" {  
    access_key = "foo"  
    secret_key = "bar"  
    region     = "us-east-1"  
}
```

```
# The default provider configuration  
provider "aws" {  
    # ...  
}  
  
# Additional provider configuration  
provider "aws" {  
    alias = "west"  
    region = "us-west-2"  
}
```

Data Sources

- Permite procurar por informações que não estão definidas diretamente nos arquivos de configuração
 - Somente leitura
 - Implementados pelos Providers

```
data "aws_ami" "web" {  
  filter {  
    name     = "state"  
    values   = ["available"]  
  }  
  
  filter {  
    name     = "tag:Component"  
    values   = ["web"]  
  }  
  
  most_recent = true  
}
```


Configuração Output

- Define valores que serão retornados para o usuário após o provisionamento

```
output NAME {  
    value = VALUE  
}
```

- Itens

- value
- description
- depend_on
- sensitive

```
output "address" {  
    value = "${aws_instance.db.public_dns}"  
}
```

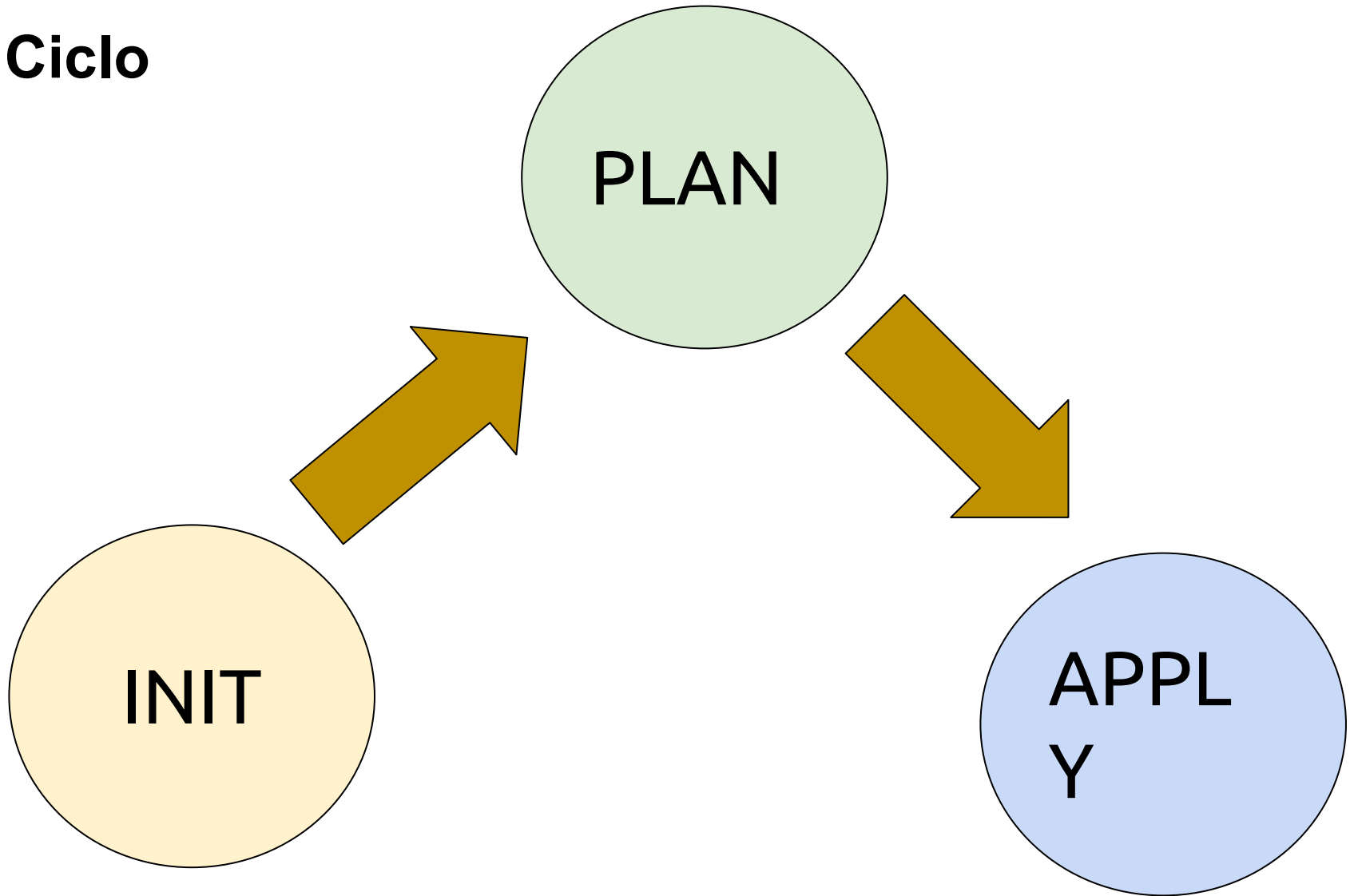
Além disso...

- Configuração Terraform
 - Permite configurar o comportamento da própria ferramenta
 - Versão requerida do Terraform
 - Backends
- Modules
 - Agrupamento de recursos que serão gerenciados de forma única
 - Permite o reuso

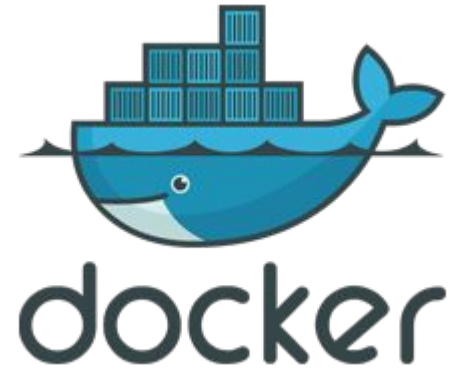
Comando

- Um único comando gerencia todo o ciclo do Terraform
 - `terraform [--version] [--help] <command> [args]`
- Subcomandos
 - `apply`
 - `console`
 - `destroy`
 - `fmt`
 - `get`
 - `graph`
 - `init`
 - `plan`
 - `output`
 - `...`

Ciclo



Providers



Provisioners

- Forma para configurar recursos criados
 - scripts
 - chef
 - puppet
 - salt
- Múltiplos provisioners são possíveis
- creation time vs destroy time

```
resource "aws_instance" "web" {  
  # ...  
  
  provisioner "local-exec" {  
    command = "echo ${self.private_ip} > file.txt"  
  }  
}
```

Exercício 3

Crie um servidor web, um grupo de segurança e veja o ciclo de vida do servidor enquanto você ajusta algumas das propriedades da instância.

<https://github.com/michelav/espec-iac>

<https://www.terraform.io/docs/providers/aws/index.html>

Exercício 4

Ajuste o que foi feito no exercício 01 para incluir 3 servidores e um balanceador de carga.

Siga as orientações disponíveis no repositório e no site do Terraform.

<https://github.com/michelav/espec-iac>

<https://www.terraform.io/docs/providers/aws/index.html>

Dica: Para loops e iterações, use o meta-parâmetro **count**!

Tópicos Avançados

State

- Terraform guarda o estado dos itens que são gerenciados por ele
 - Como um banco de dados
 - Permite acompanhar a situação dos itens gerenciados pela ferramenta
 - Melhora consideravelmente o desempenho em infraestruturas complexas

State

- Funcionamento
 - A ferramenta cria um arquivo denominado *terraform.tfstate*
 - O arquivo descreve cada um dos itens gerenciados pela ferramenta
 - Ao aplicar um template, a ferramenta armazena cada recurso criado, seu identificador e seus metadados
 - Essas informações são utilizadas em outros comandos
 - plan, apply, import, etc
 - Modificações no arquivo não são recomendadas
 - terraform state

State

```
"aws_instance.example": {  
  "type": "aws_instance",  
  "primary": {  
    "id": "i-66ba8957",  
    "attributes": {  
      "ami": "ami-2d39803a",  
      "availability_zone": "us-east-1d",  
      "id": "i-66ba8957",  
      "instance_state": "running",  
      "instance_type": "t2.micro",  
      "network_interface_id": "eni-7c4fcf6e",  
      "private_dns": "ip-172-31-53-99.ec2.internal",  
      "private_ip": "172.31.53.99",  
      "public_dns": "ec2-54-159-88-79.compute-1.amazonaws.com",  
      "public_ip": "54.159.88.79",  
      "subnet_id": "subnet-3b29db10"  
    }  
  }  
}
```

State

- Parece funcionar bem para casos simples, mas...

Como fazer para que um time com diversos membros consiga compartilhar esse estado?!

Como controlar o acesso a esse arquivo?

State

- Uma opção plausível seria adicioná-lo ao sistema de controle de versão do projeto!
 - **Será que é mesmo?!?!**
- Colocar o arquivo em uma área de armazenamento remota e acessível por todos
 - E se mais de um processo tentar atualizar o arquivo?!

State Locking

- Garante a consistência e integridade do estado da sua infraestrutura
 - Acontece automaticamente quando mais de uma operação tenta atualizar o estado da infra
 - Nem sempre há mensagens, pois acontece de forma transparente
 - Você pode desabilitar o procedimento, mas não é recomendado
- Só pode acontecer com o suporte de **backends**

Backend

- Determina como o estado do seu projeto será armazenado e carregado
 - Padrão: estado armazenado localmente
- Backend remoto
 - Garantir acesso ao estado por toda a equipe do projeto
 - Conservar informação sensível do projeto
 - Visão única por toda a equipe

Backend

- Padrão (Standard)
 - Garante o gerenciamento do estado
 - Ex.:
 - backend local: arquivo em sistema de arquivos
- Aprimorado (Enhanced)
 - Padrão + operações remotas
 - Ex.:
 - Artifactory
 - Consul
 - etcd
 - S3
 - etc

Backend

- `terraform init`
 - Qualquer novo ambiente que configure um backend
 - Mudanças na configuração do backend
 - Remoção de um backend
 - A ferramenta lembrará você disso!!
- Quando configurando pela primeira vez, a ferramenta oferece opção de migrar o estado atual para o novo backend
 - Faça backup!!!!

Backend

- Configuração Parcial
 - Complemento por meio de:
 - Interação por linha de comando
 - Arquivo
 - Chave-Valor
 - **Sem suporte a interpolação**

```
terraform {  
  backend "consul" {  
    address = "demo.consul.io"  
    path    = "example_app/terraform_state"  
  }  
}
```

```
terraform {  
  backend "consul" {}  
}
```

```
address = "demo.consul.io"  
path    = "example_app/terraform_state"
```

```
$ terraform init \  
  -backend-config="address=demo.consul.io" \  
  -backend-config="path=example_app/terraform_state"
```

Modules

- Forma de empacotar um grupo de recursos e compartilhá-los entre times
- Uso:
 - Defina um ou mais arquivos em uma pasta
 - `module "name" { source = "..."` }

```
module "frontend" {  
  source = "/modules/frontend-app"  
  
  min_size = 10  
  max_size = 20  
}  
  
resource "aws_autoscaling_policy" "scale_out"  
  name = "scale-out-frontend-app"  
  autoscaling_group_name = "${module.frontend.  
  adjustment_type = "ChangeInCapacity"  
  policy_type = "SimpleScaling"  
  scaling_adjustment = 1  
  cooldown = 200  
}
```

Extensão

- Plugins
 - Mecanismo de adaptação da ferramenta
 - Providers e Provisioners são incorporados por meio de plugins
 - Formam um processo separado que se comunica com o *core* do Terraform por meio de RPC

```
package main

import (
    "github.com/hashicorp/terraform/plugin"
)

func main() {
    plugin.Serve(new(MyPlugin))
}
```

Dicas

- Divida seus recursos, variáveis e saídas em arquivos
 - Organize esses arquivos em pasta conforme camada e perímetro

```
stage
├── vpc
├── services
│   ├── frontend-app
│   ├── backend-app
│   │   ├── vars.tf
│   │   ├── outputs.tf
│   │   ├── main.tf
│   │   └── .terraform
├── data-storage
│   ├── mysql
│   └── redis
prod
├── vpc
├── services
│   ├── frontend-app
│   └── backend-app
├── data-storage
│   ├── mysql
│   └── redis
```

Dicas

- Gerencie o estado da sua Infra em um repositório remoto
 - S3, etcd, etc
- Reuse por meio de módulos
- Use variáveis de ambiente ou variáveis por linha de comando
 - Cuidado com as senhas em arquivos planos

Exercício 5

Mude o exercício 4 para que o terraform utilize um backend remoto. Experimente compartilhar ações com uma equipe.

Dica: Utilize o bloco Terraform

```
terraform {  
  backend "s3" {  
    bucket = "mybucket"  
    key    = "path/to/my/key"  
    region = "us-east-1"  
  }  
}
```