

Cloud **INFRASTRUCTURE** Automation



Boa Noite!

Leonardo DAngelo

 leonardodg2084@gmail.com

 <https://www.linkedin.com/in/leonardodg/>

1.

INTRODUÇÃO

Breve
introdução ao
terraform e
seus
componentes

“ O terraform é uma ferramenta destinada para criação de infraestrutura como **código**, com seu foco em nuvens públicas como a AWS, Azure e GCP.

O terraform utiliza-se de uma linguagem declarativa, para definição da infraestrutura, você precisará declarar "**O QUE**" você quer provisionar e não "**COMO**" ela deve executar.

Diferenças entre Ansible x **TERRAFORM**



Ambas as ferramentas são conhecidas como "**ferramentas de infraestrutura como códigos**" e ambas possuem funções em comum, então qual delas utilizar?

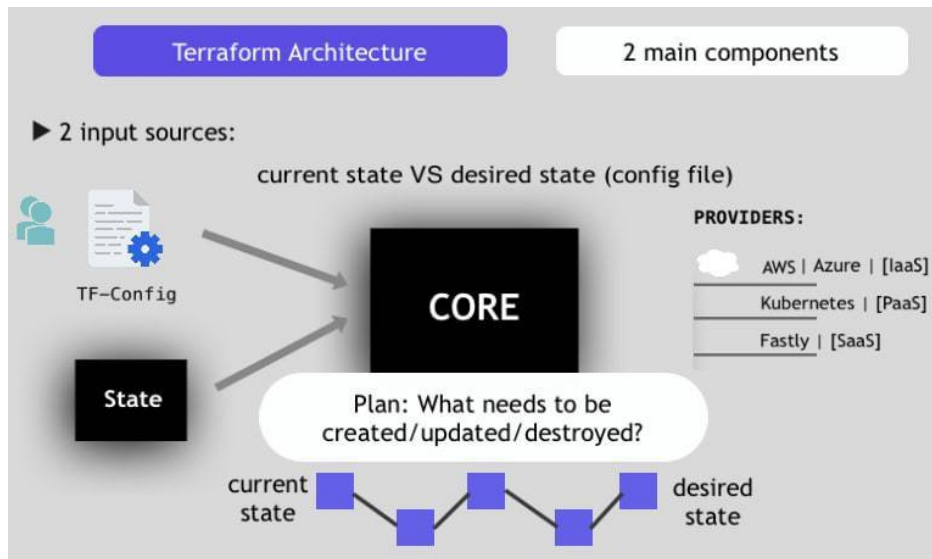
TERRAFORM

- Ferramenta com foco em criação de infraestrutura
- Ferramenta relativamente nova
- Opções mais avançadas de orquestração.
- Armazenar estado da infraestrutura.
- Organização das dependências.
- A maioria das alterações e remoção de infraestrutura podem ser parametrizadas.

ANSIBLE

- Ferramenta com maior maturidade
- Ferramenta com suporte a vários provedores
- Ferramenta destinada a configuração de infraestrutura
- Toda alteração de infraestrutura deve ser declarada como fazer.

Componentes do **TERRAFORM**



- Arquivos de declaração da infraestrutura (*.tf)
- Arquivo de estado das configurações (*.tfstate)
- Arquivo de parâmetros para as variáveis (*.tfvars) (Opcional)
- Arquivo de configuração do provider (*.tf)
- Arquivo de configuração do backend para os arquivos tfstate (*.tf)

Passos de execução do **TERRAFORM**

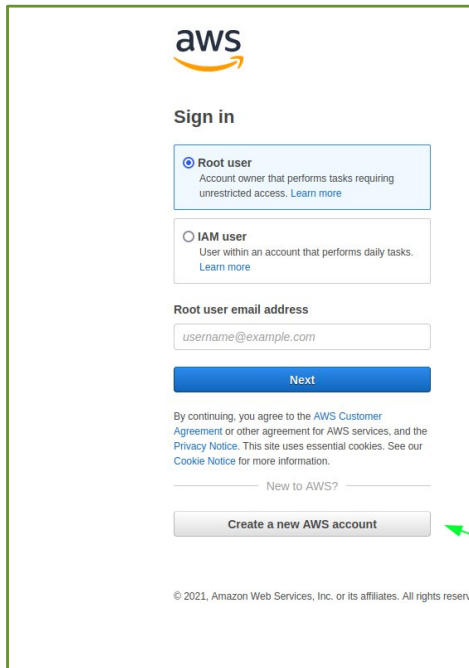
- **INIT/REFRESH** Esse passo é onde o terraform identifica os providers, ou seja, contra qual infraestrutura o terraform ira criar os recursos bem como as credenciais.
- **PLAN** No passo de planejamento é onde o terraform irá avaliar o provedor e verificar quais recursos existem e quais recursos precisam ser criados.
- **APPLY** No passo de aplicar, o terraform irá criar ou alterar a infraestrutura de acordo com o que foi encontrado de alterações no passo de planejamento.
- **DESTROY** Nesse passo você pode remover parcialmente ou totalmente a infraestrutura.



Criação de conta na AWS



- Criar uma nova conta na AWS ou utilizar alguma caso já possua



The image shows the AWS Sign in page. At the top is the AWS logo. Below it is the 'Sign in' heading. There are two main options: 'Root user' (selected with a blue circle) and 'IAM user'. The 'Root user' option has a description: 'Account owner that performs tasks requiring unrestricted access. Learn more'. The 'IAM user' option has a description: 'User within an account that performs daily tasks. Learn more'. Below these is a text input field for 'Root user email address' with the placeholder 'username@example.com'. A blue 'Next' button is below the input field. At the bottom, there is a link 'New to AWS?' and a button 'Create a new AWS account' which is highlighted with a green arrow.

aws

Sign in

☒ **Root user**
Account owner that performs tasks requiring unrestricted access. [Learn more](#)

☐ **IAM user**
User within an account that performs daily tasks. [Learn more](#)

Root user email address

username@example.com

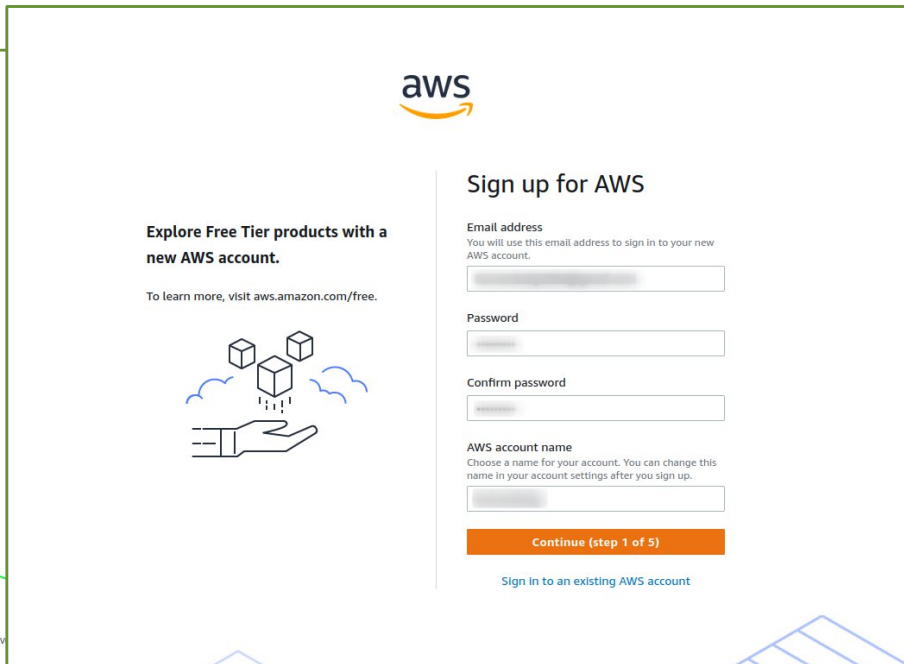
Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

☐ New to AWS?

Create a new AWS account

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.




The image shows the AWS Sign up for AWS page. At the top is the AWS logo. Below it is the heading 'Sign up for AWS'. There is a sub-heading 'Explore Free Tier products with a new AWS account.' and a link 'To learn more, visit aws.amazon.com/free.' Below this is an illustration of a hand holding three cubes. The main form has four sections: 'Email address' (with a text input field), 'Password' (with a text input field), 'Confirm password' (with a text input field), and 'AWS account name' (with a text input field). Below the form is an orange 'Continue (step 1 of 5)' button and a link 'Sign in to an existing AWS account'.

aws

Sign up for AWS

Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.



Email address

You will use this email address to sign in to your new AWS account.

Password

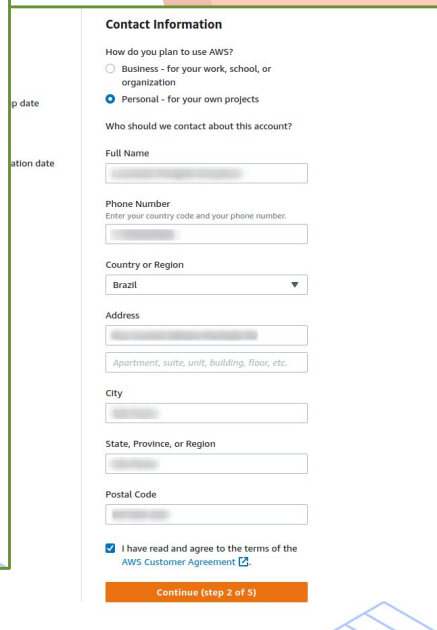
Confirm password

AWS account name

Choose a name for your account. You can change this name in your account settings after you sign up.

Continue (step 1 of 5)

[Sign in to an existing AWS account](#)



The image shows the AWS Contact Information page. At the top is the heading 'Contact Information'. Below it is a section 'How do you plan to use AWS?' with two radio buttons: 'Business - for your work, school, or organization' and 'Personal - for your own projects' (selected). Below this is a section 'Who should we contact about this account?' with a text input field for 'Full Name'. Below that is a section 'Phone Number' with a text input field and a label 'Enter your country code and your phone number.' Below that is a section 'Country or Region' with a dropdown menu showing 'Brazil'. Below that is a section 'Address' with a text input field and a label 'Apartment, suite, unit, building, floor, etc.'. Below that is a section 'City' with a text input field. Below that is a section 'State, Province, or Region' with a text input field. Below that is a section 'Postal Code' with a text input field. At the bottom, there is a checkbox 'I have read and agree to the terms of the AWS Customer Agreement' and an orange 'Continue (step 2 of 5)' button.

Contact Information

How do you plan to use AWS?

☐ Business - for your work, school, or organization

☒ Personal - for your own projects

Who should we contact about this account?

Full Name

Phone Number

Enter your country code and your phone number.

Country or Region

Brazil

Address

Apartment, suite, unit, building, floor, etc.

City

State, Province, or Region

Postal Code

☒ I have read and agree to the terms of the [AWS Customer Agreement](#)

Continue (step 2 of 5)

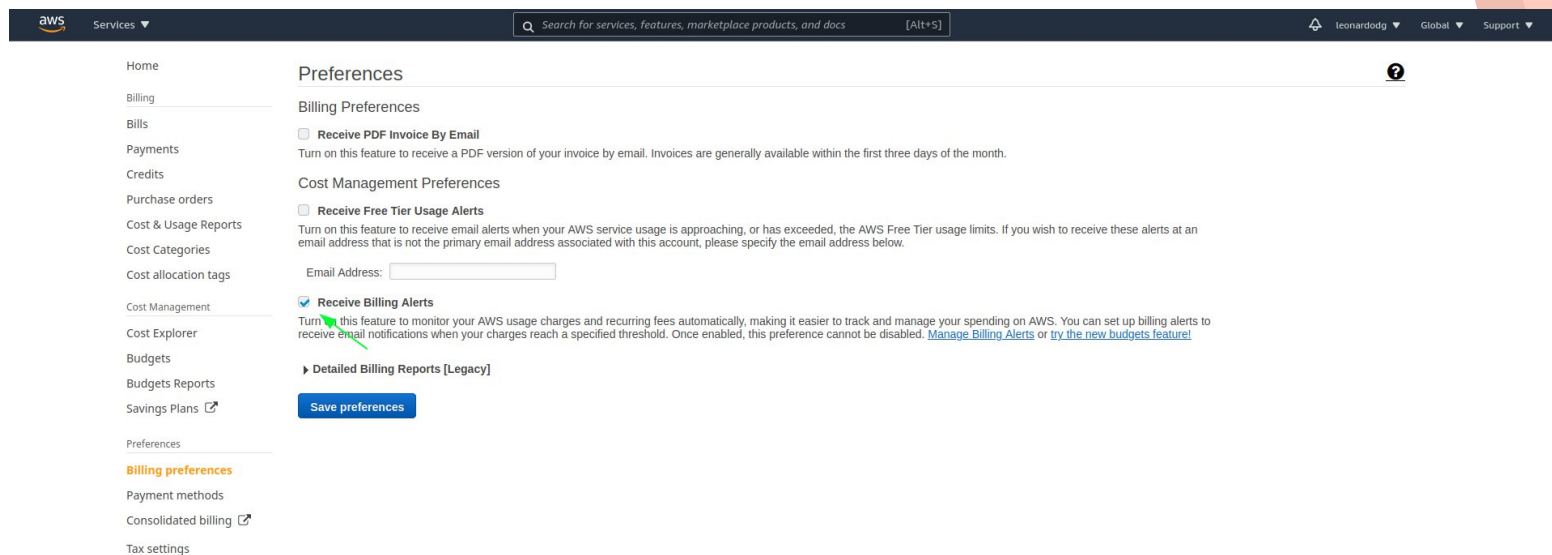
Criação de alarme de gastos na AWS

- Após a criação da conta, vamos criar um alarme em caso de altos custos na AWS, para evitar surpresas.

The screenshot displays the AWS Billing & Cost Management Dashboard. The left sidebar contains a navigation menu with the following items: Home, Billing, Bills, Payments, Credits, Purchase orders, Cost & Usage Reports, Cost Categories, Cost allocation tags, Cost Management, Cost Explorer, Budgets, Budgets Reports, Savings Plans, Preferences, Billing preferences (highlighted with a green arrow), Payment methods, Consolidated billing, and Tax settings. The main content area is titled 'Billing & Cost Management Dashboard' and includes a 'Getting Started with AWS Billing & Cost Management' section with links to 'Manage your costs and usage using AWS Budgets', 'Visualize your cost drivers and usage trends via Cost Explorer', 'Dive deeper into your costs using the Cost and Usage Reports with Athena integration', and 'Learn more: Check out the AWS What's New webpage'. Below this is a 'Spend Summary' section with a 'Cost Explorer' button. The summary text reads: 'Welcome to the AWS Account Billing console. Your last month and month-to-date costs appear below. Current month-to-date balance for August 2021, the exchange rate for the Payment Currency is estimated. 0.00 USD which converts to 0.00 BRL at today's exchange rate of 5.29'. To the right of the summary is a 'Month-to-Date Spend by Service' section with a 'Bill Details' button and a donut chart showing '\$0'. At the bottom, there is a 'No Amount Due' status bar with a '\$0.00' value.

Criação de alarme de gastos na AWS

- ▶ Marcar a opção **"Receive billing alerts"**



The screenshot shows the AWS Billing Preferences page. The left sidebar contains a navigation menu with the following items: Home, Billing, Bills, Payments, Credits, Purchase orders, Cost & Usage Reports, Cost Categories, Cost allocation tags, Cost Management, Cost Explorer, Budgets, Budgets Reports, Savings Plans, Preferences, **Billing preferences** (highlighted), Payment methods, Consolidated billing, and Tax settings. The main content area is titled 'Preferences' and includes a help icon. Under 'Billing Preferences', there are three sections: 1. 'Receive PDF Invoice By Email' with an unchecked checkbox and a description. 2. 'Cost Management Preferences' with an unchecked checkbox for 'Receive Free Tier Usage Alerts' and a description. 3. 'Receive Billing Alerts' with a checked checkbox and a description. Below this is an 'Email Address' input field. At the bottom, there is a 'Detailed Billing Reports [Legacy]' link and a 'Save preferences' button. The top navigation bar includes the AWS logo, 'Services', a search bar, and user account information.

aws Services ▾ Search for services, features, marketplace products, and docs [Alt+S] leonardodg ▾ Global ▾ Support ▾

Preferences ?

Billing Preferences

☐ **Receive PDF Invoice By Email**
Turn on this feature to receive a PDF version of your invoice by email. Invoices are generally available within the first three days of the month.

Cost Management Preferences

☐ **Receive Free Tier Usage Alerts**
Turn on this feature to receive email alerts when your AWS service usage is approaching, or has exceeded, the AWS Free Tier usage limits. If you wish to receive these alerts at an email address that is not the primary email address associated with this account, please specify the email address below.

Email Address:

☒ **Receive Billing Alerts**
Turn on this feature to monitor your AWS usage charges and recurring fees automatically, making it easier to track and manage your spending on AWS. You can set up billing alerts to receive email notifications when your charges reach a specified threshold. Once enabled, this preference cannot be disabled. [Manage Billing Alerts](#) or [try the new budgets feature!](#)

▶ [Detailed Billing Reports \[Legacy\]](#)

[Save preferences](#)

Home
Billing
Bills
Payments
Credits
Purchase orders
Cost & Usage Reports
Cost Categories
Cost allocation tags
Cost Management
Cost Explorer
Budgets
Budgets Reports
Savings Plans [↗](#)
Preferences
Billing preferences
Payment methods
Consolidated billing [↗](#)
Tax settings

Criação de alarme de gastos na AWS

- ▶ Agora no serviço de cloudwatch deve aparecer uma opção nova chamada **billing**

The screenshot displays the AWS CloudWatch console interface. On the left-hand navigation pane, under the 'Alarms' section, the 'Billing' option is highlighted with a green arrow. The main content area is titled 'Billing alarms (0)' and includes a search bar, filters for 'Any state' and 'Any type', and buttons for 'Clear selection', 'Create composite alarm', and 'Create alarm'. Below the filters, a table header lists columns: Name, State, Last state update, Conditions, and Actions. The main content area contains informational text about monitoring AWS bill charges and a green arrow pointing to the 'Create alarm' button.

CloudWatch > Alarms

Billing alarms (0)

☐ Hide Auto Scaling alarms

Clear selection Create composite alarm Actions

Search

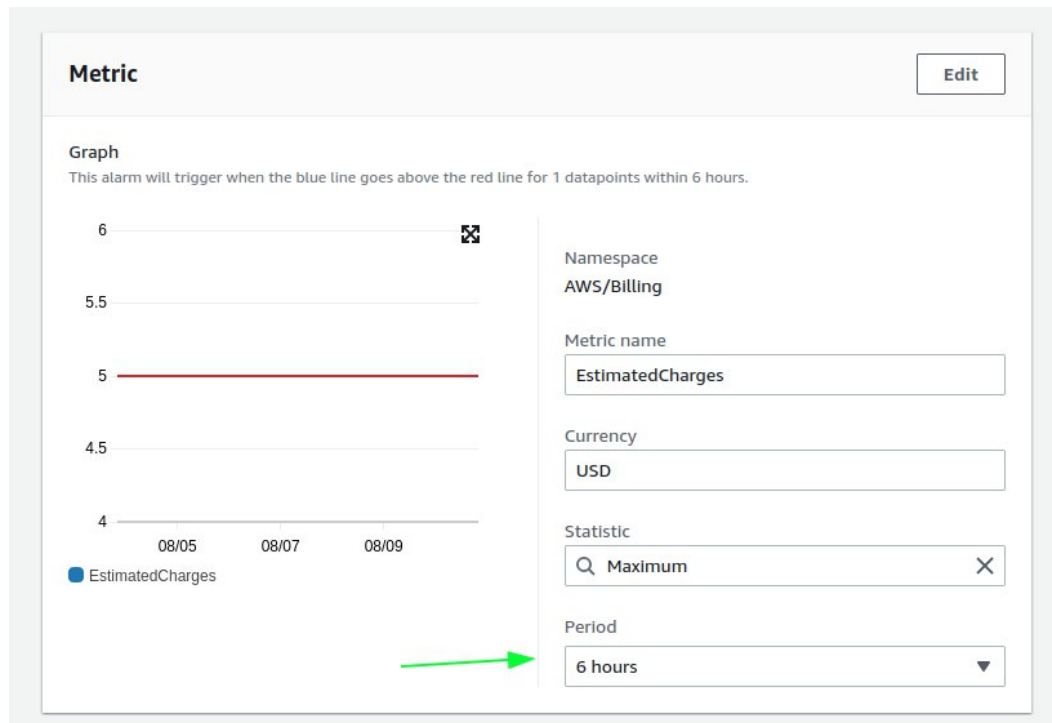
Any state Any type

< 1 > ⚙

Name	State	Last state update	Conditions	Actions
<p>Amazon CloudWatch can help you monitor the charges on your AWS bill by sending you email alerts when charges exceed a threshold you define.</p> <p>Once you update your preferences in the Account Billing console, you will begin receiving Amazon CloudWatch metrics that reflect your month-to-date AWS charges. Then, you can create a billing alarm by specifying a spending threshold and an e-mail address to notify. Learn more about billing alerts</p> <p>You get 10 free alarms and 1,000 free e-mail notifications each month as part of the AWS Free Tier</p> <p><input type="button" value="Create alarm"/></p>				

Criação de alarme de gastos na **AWS**

- Selecione as horas conforme a **imagem**



Criação de alarme de gastos na AWS

- ▶ Selecione o valor conforme a **imagem**

Conditions

Threshold type

☒ Static
Use a value as a threshold

☐ Anomaly detection
Use a band as a threshold

Whenever EstimatedCharges is...

Define the alarm condition.

☒ Greater
> threshold

☐ Greater/Equal
≥ threshold

☐ Lower/Equal
≤ threshold

☐ Lower
< threshold

than...

Define the threshold value.

5 USD

Must be a number

▶ Additional configuration

Cancel Next

Criação de alarme de gastos na AWS

- Selecione o valor conforme a **imagem**

Notification

Alarm state trigger
Define the alarm state that will trigger this action. Remove

☒ **In alarm**
The metric or expression is outside of the defined threshold.

☐ **OK**
The metric or expression is within the defined threshold.

☐ **Insufficient data**
The alarm has just started or not enough data is available.

Select an SNS topic
Define the SNS (Simple Notification Service) topic that will receive the notification.

☐ Select an existing SNS topic

☒ **Create new topic**

☐ Use topic ARN

Create a new topic...
The topic name must be unique.

SNS topic names can contain only alphanumeric characters, hyphens (-) and underscores (_).

Email endpoints that will receive the notification...
Add a comma-separated list of email addresses. Each address will be added as a subscription to the topic above.

user1@example.com, user2@example.com

Criação de alarme de gastos na **AWS**

- Clique em **next**

Email (endpoints)
leonardodg2084@gmail.com - [View in SNS Console](#)

Add notification

Auto Scaling action

Add Auto Scaling action

EC2 action

This action is only available for EC2 Per-Instance Metrics.

Add EC2 action

Systems Manager action [Info](#)

This action will create an Incident or Opsitem in Systems Manager when the alarm is **In alarm** state.

Add Systems Manager action

Cancel Previous **Next**

Criação de alarme de gastos na **AWS**

- Crie um **nome** para este alarme.

Add name and description

Name and description

Alarm name

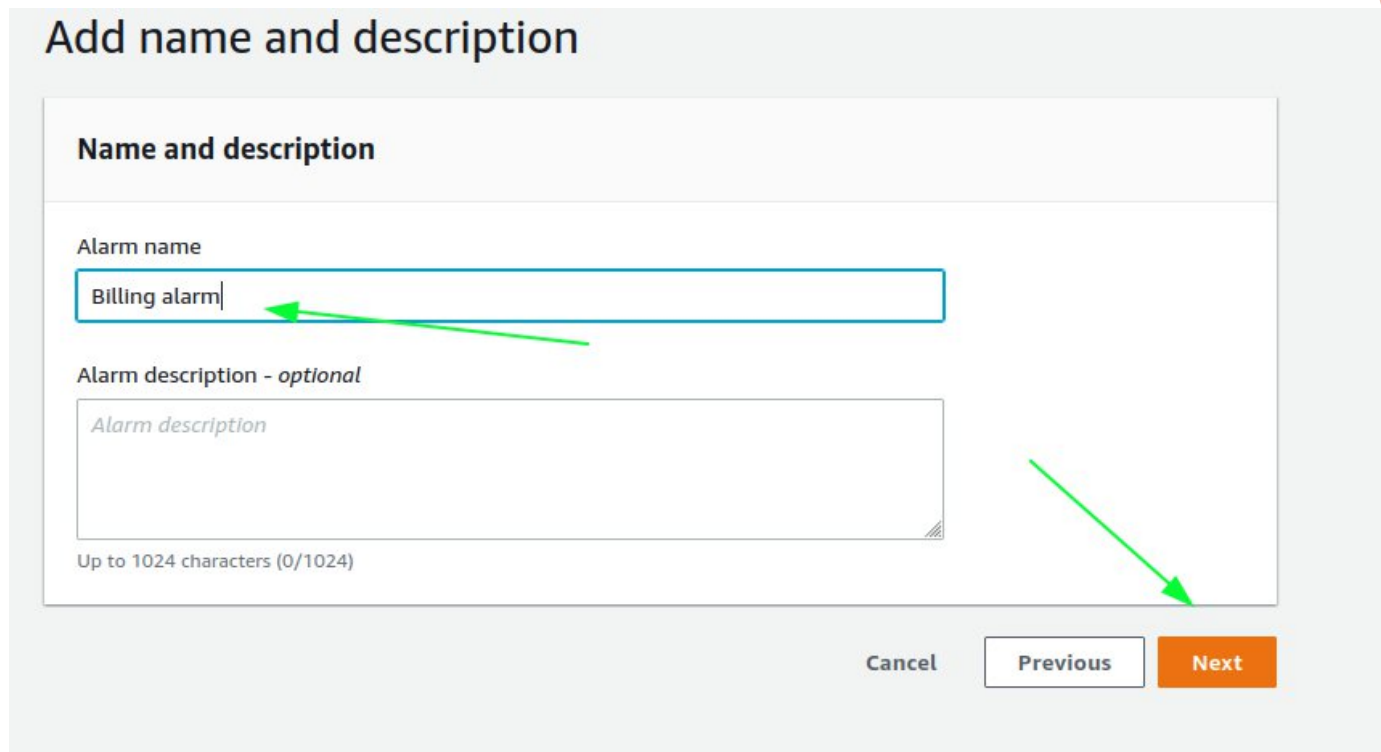
Billing alarm

Alarm description - *optional*

Alarm description

Up to 1024 characters (0/1024)

Cancel Previous **Next**



Criação de alarme de gastos na **AWS**

- ▶ Clique no botão **Creat alarm**.

Step 3: Add name and description Edit

Name and description

Name
Billing alarm

Description
-

Cancel Previous Create alarm

Criação de alarme de gastos na AWS

- **Alarme** criado com sucesso.

✓ Successfully created alarm Billing alarm.

×

ⓘ Some subscriptions are pending confirmation

Amazon SNS doesn't send messages to an endpoint until the subscription is confirmed

View SNS Subscriptions

×

CloudWatch > Alarms

Billing alarms (1)

☐ Hide Auto Scaling alarms

Clear selection

Create composite alarm

Actions ▾

Create alarm

Q Search

Any state ▾

Any type ▾

< 1 > ⚙

<input type="checkbox"/>	Name ▾	State ▾	Last state update ▾	Conditions	Actions ▾
<input type="checkbox"/>	Billing alarm	⊖ Insufficient data	2021-08-10 20:10:32	EstimatedCharges > 5 for 1 datapoints within 6 hours	🟢 1 action(s) enabled <u>Warning</u>

Criação de um usuário na **AWS**

- ▶ Para a criação de recursos na AWS através do terraform é necessário um usuário com permissão para a criação desses recursos. Abaixo vamos executar o procedimento para criação desse usuário.
- ▶ Salvar as credenciais em um lugar seguro e nunca enviar para o git.
- ▶ Pode-se utilizar as credenciais em forma de variáveis de ambiente utilizando as seguintes variáveis:

```
export AWS_ACCESS_KEY_ID="VALOR"
```


```
export AWS_SECRET_ACCESS_KEY="VALOR"
```

```
export AWS_DEFAULT_REGION="us-east-1"
```



Criação de uma **VM** via Vagrant

Para os nosso estudos vamos utilizar uma VM, provisionada via Vagrant para executarmos os nossos scripts em Terraform.




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

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/hirsute64"
  config.vm.hostname = "iaac-station"
  config.vm.provision "shell", path: "iaac.sh", run: "once"
  config.vm.network :private_network, ip: "10.0.0.10"
  config.vm.provider "virtualbox" do |v|
    v.cpus = 2
    v.memory = 4096
  end
end
```

Criação de uma **VM** via Vagrant

Arquivo chamado **iaac.sh**, com o bootstrap da VM.



```
apt update
apt dist-upgrade -y
apt install -y git git-flow docker.io docker-compose awscli
gnupg software-properties-common curl
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key
add -
apt-add-repository "deb [arch=amd64]
https://apt.releases.hashicorp.com $(lsb_release -cs) main"
apt update  && apt install terraform -y
``
```

Criação de uma **VM** via Vagrant

Seguem os comandos abaixo para início e acessar a VM:



```
$ vagrant up
```

```
--
```

```
$ vagrant ssh
```

2.

USUÁRIO

Usuário de
execução
do
terraform

Criação de **usuário** para o terraform

Para a criação de recursos na AWS através do terraform é necessário um usuário com permissão para a criação desses recursos. Abaixo vamos executar o procedimento para criação desse usuário.

Criação de **usuário** para o terraform

Vá ate **IAM** e vamos criar um usuário para o terraform.



Identity and Access Management (IAM)

- Dashboard
- Access management
 - User groups
 - Users**
 - Roles
 - Policies
 - Identity providers
 - Account settings
- Access reports
 - Access analyzer
 - Archive rules
 - Analizers
 - Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

Search IAM

IAM dashboard

Sign-in URL for IAM users in this account
<https://628073007650.signin.aws.amazon.com/console> | [Customize](#)

IAM resources

Users: 0	Roles: 2
User groups: 0	Identity providers: 0
Customer managed policies: 0	

Security alerts

⚠ The root user for this account does not have Multi-factor authentication (MFA) enabled. [Enable MFA](#) to improve security for this account.

Best practices

- Grant [least privilege access](#): Establishing a principle of least privilege ensures that identities are only permitted to perform the most minimal set of functions necessary to fulfill a specific task, while balancing usability and efficiency.
- Use [AWS Organizations](#): Centrally manage and govern your environment as you scale your AWS resources. Easily create new AWS accounts, group accounts to organize your workflows, and apply policies to accounts or groups for governance.
- Enable Identity federation: Manage users and access across multiple services from your preferred identity source. Using [AWS Single Sign-On](#) centrally manage access to multiple AWS accounts and provide users with single sign-on access to all their assigned accounts from one place.
- Enable MFA: For extra security, we recommend that you require multi-factor authentication (MFA) for [all users](#).
- [Rotate credentials](#) regularly: Change your own passwords and access keys regularly, and make sure that all users in your account do as well.
- Enable [IAM Access Analyzer](#): Enable IAM Access Analyzer to analyze public, cross-account, and cross-organization access. [Learn more about all security best practices](#)

What's new

Learn about the latest releases for AWS Identity & Access Management (IAM)

- [AWS Identity and Access Management now makes it easier to relate a user's IAM role activity to their corporate identity](#)
3 months ago (4/13/2021)

Additional information

- [IAM documentation](#)
- [Videos, IAM release history and additional resources](#)

Tools

- [Web identity federation playground](#)
- [Policy simulator](#)

Quick links

- [My access key](#)

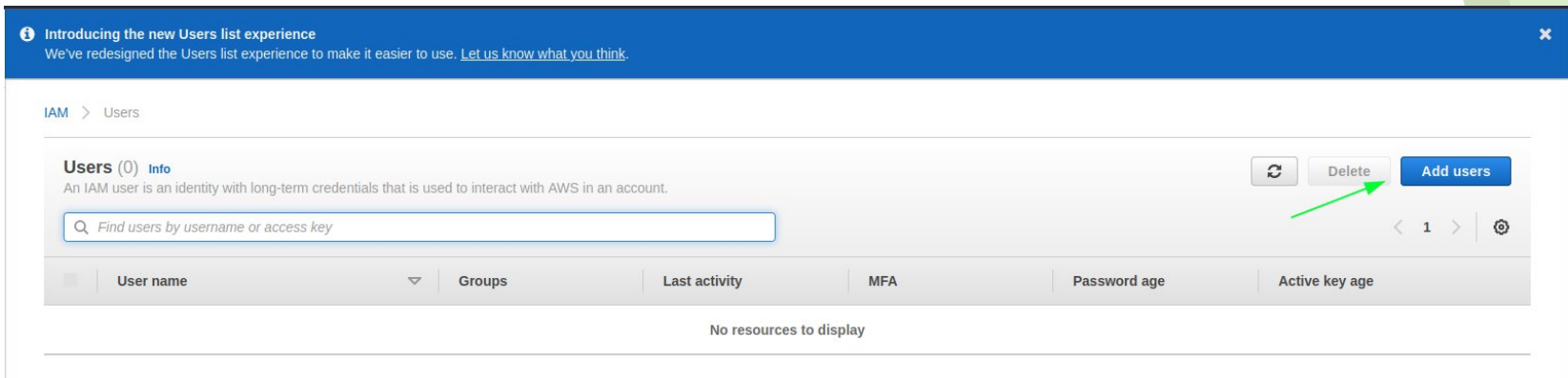
Related services

- [AWS Organizations](#)
- [AWS Single Sign-on \(SSO\)](#)

AWS account ID: 628073007650

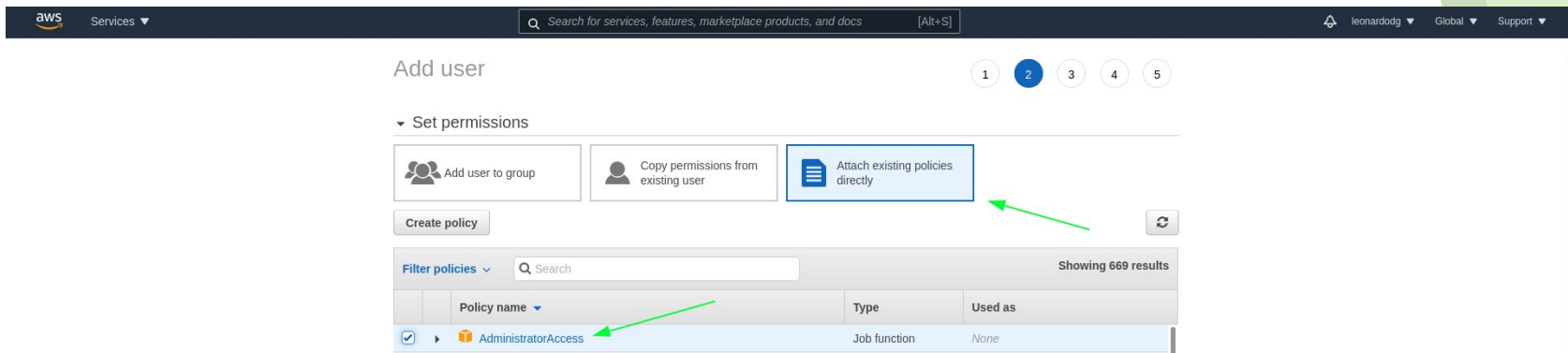
Criação de **usuário** para o terraform

Clique no botão e **Add User**.



Criação de **usuário** para o terraform

Adicionar a policy
AdministratorAccess.



The screenshot shows the AWS IAM console interface for creating a new user. The page title is "Add user". There are five numbered steps at the top, with step 2, "Set permissions", being the active one. Under "Set permissions", there are three options: "Add user to group", "Copy permissions from existing user", and "Attach existing policies directly". The "Attach existing policies directly" option is selected and highlighted with a blue border. Below these options is a "Create policy" button. Underneath is a section for "Filter policies" with a search bar and a "Showing 669 results" indicator. A table lists the available policies. The first policy, "AdministratorAccess", is selected with a checkbox and highlighted in blue. Two green arrows point to the "Attach existing policies directly" option and the "AdministratorAccess" policy row.

Policy name	Type	Used as
<input checked="" type="checkbox"/> AdministratorAccess	Job function	None

Criação de **usuário** para o terraform

Adicionar as **Tags**.

Add user



Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
<input type="text" value="Name"/>	<input type="text" value="terraform_user"/>	✕
<input type="text" value="Add new key"/>	<input type="text"/>	

You can add 49 more tags.

Criação de **usuário** para o terraform

Valide as informações e clique **Create User**.

Add user

12345

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name

terraform_user

AWS access type

Programmatic access - with an access key

Permissions boundary

Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Tags

The new user will receive the following tag

Key	Value
Name	terraform_user

Cancel

Previous

Create user

Criação de **usuário** para o terraform

Obtenha a **AccessKey e SecretKey** e armazene em um local seguro e **jamaiz** faça upload do arquivo no github.


3.

TERRAFORM

Comandos
básicos

Comandos básicos do TERRAFORM


Seguem os comandos abaixo para início e acessar a VM:



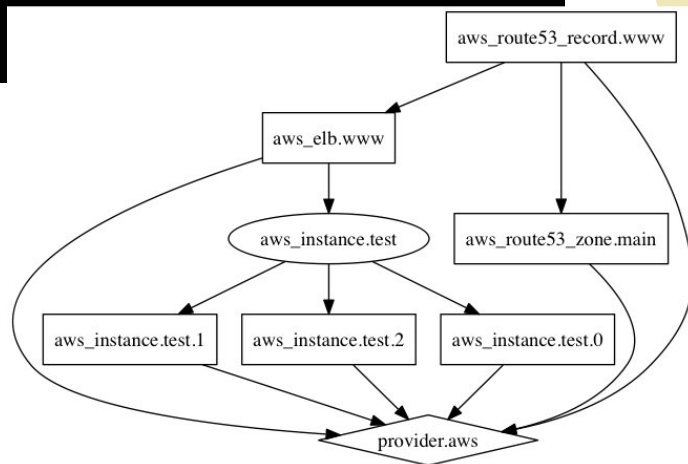
```
# Inicia o terraform, faz download dos providers
$ terraform init
# Verifica as alterações da infraestrutura
$ terraform plan
# Aplica as mudanças encontradas no plano
$ terraform apply --auto-approve
# Aplica as mudanças apenas em um recurso
$ terraform apply --target=aws_instance.exemplo
# Verifica mudanças no provider
$ terraform refresh
# Valida sintaticamente os arquivos do terraform
$ terraform validate
# Exibe os objetos criados pelo terraform
$ terraform state show
```


Comandos básicos do TERRAFORM

Seguem os comandos abaixo para início e acessar a VM:



```
# Exibe as saidas criadas pelo usuario
$ terraform output
# Fornece uma saida no formado GraphViz com as arvores de
dependências entre recursos
$ terraform graph
```



Comando **Validate** do **TERRAFORM**

O comando terraform validate como o nome já diz serve para validar sintaticamente os arquivos do terraform, logo antes de executar o comando terraform plan deve-se executar o comando validate afim de diferenciar os erros de sintaxe aos erros possíveis do provider.



Link:

<https://www.terraform.io/docs/cli/commands/validate.html>

```
$ terraform validate
```

Providers do **TERRAFORM**

Um provider no terraform nada mais é que uma abstração de uma API fornecida pelo mantenedor do software ou pela própria hashcorp.

Todos os providers possuem parâmetros mandatórios e parâmetros opcionais, para mais informações, visite a pagina do provider, lá está disponível os atributos bem como exemplos.



```
provider "aws" {  
    region = "us-east-1"  
}
```

```
$ terraform init
```

Variáveis no **TERRAFORM**

Embora o terraform não seja uma linguagem de programação e sim uma linguagem declarativa, o terraform possui variáveis para reutilização de valores, criação de constantes, chaves/valor, listas, etc..

Documento oficial:

<https://www.terraform.io/docs/language/values/variables.html>

Variáveis no TERRAFORM

Como boas praticas as variáveis devem ficar em um arquivo chamado **variables.tf**, as variáveis podem possuir valores default caso o usuário não as adicione.

String

```
variable "var_string"  
  type = string  
}
```

nome da variável

Tipo da variável

Lista

```
variable "zonas_disponiveis" {  
  type      = list(string)  
  default = ["us-central1-a", "us-central1-b", "us-central1-c"]  
}
```

Valor padrão da variável

Variáveis no TERRAFORM

Estrutura de dados

Map

```
variable "planos" {  
  type = map  
  default = {  
    "small"  = "1xCPU-1GB"  
    "medium" = "1xCPU-2GB"  
    "large"  = "2xCPU-4GB"  
  }  
}
```

```
variable "docker_ports" {  
  type = list(object({  
    origem = number  
    destino = number  
    protocol = string  
  }))  
  default = [  
    {  
      internal = 8100  
      external = 8300  
      protocol = "tcp"  
    }  
  ]  
}
```

Objeto

Variáveis no TERRAFORM

```
output "var_string" {  
  value = var.var_string  
}
```

```
output "var_zonas" {  
  value = var.zonas_disponiveis[1]  
}
```



```
output "var_planos" {  
  value = var.planos["medium"]  
}
```

```
output "var_docker_ports" {  
  value = var.docker_ports[0]["internal"]  
}
```

Variáveis no TERRAFORM

Caso alguma variavel seja declarada mas não possuir um valor default, esse valor será **solicitado** no momento do plan e apply, para evitar isso podemos usar a flag **-var** e declarar os valores



```
$ terraform apply -var="var_string=impacta_cmd"
```

```
$ terraform apply -var="zonas_disponiveis=["brazil", "eua"]"
```


Variáveis no TERRAFORM

Há outro modo de enviar as variáveis para um determinado terraform, esse é o maior nível de abstração dos códigos do terraform, o arquivo nada mais é do que declaração de variáveis, abaixo um exemplo do arquivo:



```
var_string = "impacta"  
planos = {  
    "medium" = "4xCPU-8G"  
}  
zonas_disponiveis = ["Brasil", "Eua"]
```



```
$ terraform apply -var-file=var.tfvars
```

Datas no TERRAFORM

Na criação de qualquer recurso em terraform inevitavelmente iremos solicitar informações do provider como a imagem mais nova do sistema operacional, um determinado IP, tipos de maquinas, as configurações atuais de um determinado serviço que já foi criado, etc., para isso temos o objeto do tipo **data**, cada recurso no terraform possui um **data** equivalente.

Link

<https://www.terraform.io/docs/language/data-sources/index.html>

Datas no TERRAFORM

```
provider "aws" {  
    region = "us-east-1"  
}
```

```
output image_id {  
    value = data.aws_ami.amazon2.id  
}
```



```
$ terraform init  
$ terraform plan  
$ terraform apply
```

```
data "aws_ami" "amazon2" {  
    owners      = ["amazon"]  
    most_recent = true  
  
    filter {  
        name     = "name"  
        values   = ["Amazon*"]  
    }  
  
    filter {  
        name     = "architecture"  
        values   = ["x86_64"]  
    }  
}
```

Output no TERRAFORM

O recurso de output vai exibir os dados contidos nos recursos bem como variáveis dentro do terraform, é uma das maneiras para debug do terraform, o output possui alguns parâmetros

Link:

<https://www.terraform.io/docs/language/values/outputs.html>

Description - Uma breve descrição do que se trata esse output

Sensitive - Caso a informação a ser exibida seja uma senha, token, etc.. será exibido sensitive na saída do comando apply e plan.



```
output image_id {  
    description = "Id da imagem"  
    value = data.aws_ami.amazon2.id  
    sensitive = true  
}
```

Locals no TERRAFORM

O terraform possui uma função chamada **locals**, ela é usada quando precisamos reutilizar inumeras vezes um determinado valor, como nomes, variáveis, etc..

Link:

<https://www.terraform.io/docs/language/values/locals.html>

Em um exemplo prático, imagine que você precise que todos os recursos provisionados na AWS precisem que determinadas tags estejam presentes, de forma mandatória, um jeito de atingir esse objetivo é utilizar o locals, como sugere o exemplo abaixo:



Locals no TERRAFORM

```
variable "resource_tags" {  
    type = map(string)  
    default = {}  
}  
  
locals {  
    required_tags = {  
        "project" = "impacta",  
        "environment" = "prod"  
    }  
    tags = merge(var.resource_tags, local.required_tags)  
}  
  
output "tags" {  
    value = local.tags  
}
```



```
`$ terraform apply -var='resource_tags={"nome": "leonardo"}'`
```

Função format no TERRAFORM

A função **format** é largamente utilizada no terraform, ela serve para nomear recursos, concatenar valores e esses valores podem ser strings, números, json.

```
variable "project" {  
    type = string  
}  
  
variable "enviroment" {  
    type = string  
}  
  
output "format_value" {  
    value = format("%s_%s", var.project, var.enviroment)  
}
```



Função lookup no TERRAFORM

O terraform possui varias funções mas uma deve ser destacada que é a função de **lookup**, essa função dado uma variável do tipo **map** e uma string que servirá como "chave" dentro do map irá retornar o valor dessa chave.

O **lookup** é fundamental quando queremos que o mesmo terraform seja usado para DEV/QA e Produção, pois apenas trocando a variável de ambiente os recursos irão obter os valores daquele ambiente.

```
variable "env" {  
    type = string  
}
```



```
output "ambiente"{  
    value = lookup(var.size, var.env)  
}
```

```
variable "size" {  
    type = map  
    default = {  
        "qa" = "Large",  
        "dev" = "small",  
        "prod" = "xLarge"  
    }  
}
```


Função template no TERRAFORM

Assim como no ansible, o terraform permite trabalhar com templates, templates nada mais são do que arquivos gerados dinamicamente, a partir de variáveis,

Abaixo algumas aplicações do uso de templates.

Políticas de acesso a recursos (IAM)

User_data, execução de comandos durante o provisionamento de uma

VM

Criação de qualquer tipo de arquivo de configuração.

```
data "template_file" "user_data" {  
  template = file("user_data.sh")  
  vars = {  
    curso = var.curso  
    pacotes = join(" ", var.pacotes)  
  }  
}
```



Função template no **TERRAFORM**

```
variable "curso" {  
}  
  
variable "pacotes" {  
    default = ["docker", "vim"]  
}
```

```
data "template_file" "user_data" {  
    template = file("user_data.sh")  
    vars = {  
        curso = var.curso  
        pacotes = join(" ", var.pacotes)  
    }  
}
```

```
output "user_data" {  
    value = data.template_file.user_data.rendered  
}
```

user_data.sh

```
#!/bin/bash  
  
# Pacotes utilizados pela o curso de: ${curso}  
apt install ${pacotes}
```

Função Key no **TERRAFORM**

Quando estamos construindo uma infraestrutura é comum trabalharmos com estrutura de dados mais complexas como listas e objetos, esses valores servem para determinarmos grupos de recursos ou criar estruturas repetitivas como um "for" e nesses cenários que utilizamos as funções chamadas keys, values e elements.

A função **KEYS** serve para obter as chaves de um determinado objeto.

```
variable "planos" {  
  type = map  
  default = {  
    "small" = "1xCPU-1GB"  
    "medium" = "1xCPU-2GB"  
    "large" = "2xCPU-4GB"  
  }  
}
```

```
output "chaves" {  
  value = keys(var.planos)  
}
```

Função Value no TERRAFORM

Abaixo um exemplo de como utilizar a função **VALUE**, ainda utilizando a variável **PLANOS** vamos criar um novo output chamado valores, nele vamos utilizar a função **VALUE**

```
variable "planos" {  
  type = map  
  default = {  
    "small" = "1xCPU-1GB"  
    "medium" = "1xCPU-2GB"  
    "large" = "2xCPU-4GB"  
  }  
}
```

```
output "valores" {  
  value = value(var.planos)  
}
```

Função Element no TERRAFORM

A função **ELEMENT** serve para localizar um determinado valor dentro da lista indicando a posição, essa função é utilizada em casos de repetição, como adicionar variáveis de ambiente em um container, adicionar uma lista de usuários em uma politica de IAM, etc...

Utilizando a variável **PLANOS** vamos selecionar o valor da segunda posição da lista e exibi-lá

```
variable "planos" {  
  type = map  
  default = {  
    "small" = "1xCPU-1GB"  
    "medium" = "1xCPU-2GB"  
    "large" = "2xCPU-4GB"  
  }  
}
```

```
output "elemento" {  
  value = element(values(var.planos), 1)  
}
```

Função Count no TERRAFORM

O meta atributo **COUNT** pode ser utilizado em qualquer recurso do terraform, ele pode ser utilizado para que um recurso seja criado um numero determinado de vezes, pode ser utilizado para não se criar um determinado recurso, serve como um iterador em uma lista de elementos, etc..

```
provider "aws" {  
}
```

```
variable "usernames" {  
  type = list  
  
  default = [  
    "neo",  
    "leonardo",  
  ]  
}
```

```
resource "aws_iam_user" "names" {  
  count = length(var.usernames)  
  name  = var.usernames[count.index]  
}
```

Condicionais no TERRAFORM

No terraform é possível condicionar a criação ou não de um recurso bem como condicionar a escolha de um valor ou outro.

Condicionando um valor

Há casos que uma determinada variável deve ser escolhida caso satisfaça uma condição.

```
condition ? true_val : false_val
```

```
var.env == "prod" ? "Producao" : "Qualidade"
```

```
var.env != "prod" ? "Producao" : "Qualidade"
```

```
var.env == "prod" ? "PROD" : var.env == "qa" ? "QA" : "DEV"
```

Condicionais no TERRAFORM

No terraform é possível condicionar a criação ou não de um recurso bem como condicionar a escolha de um valor ou outro.

Condicionando um valor

Há casos que uma determinada variável deve ser escolhida caso satisfaça uma condição.

arquivo.txt

```
${value}
```

```
variable "env" {  
  type = string  
}
```

```
data "template_file" "conditional" {  
  template = file("arquivo.txt")  
  vars = {  
    "value" = var.env == "prod" ? "PROD" : "QA"  
  }  
}
```

```
output "conditional" {  
  value = data.template_file.conditional.*.rendered  
}
```


Condicionais no TERRAFORM

Há casos que há diferenças mais substanciais entre um ambiente de QA e PROD do que meramente valores, para isso podemos condicionar a criação de um recurso utilizando o parâmetro `count`.

```
data "template_file" "conditional" {  
  template = file("arquivo.txt")  
  count = var.env == "prod" ? 1 : 0  
  vars = {  
    "value" = "Producao"  
  }  
}
```

State no TERRAFORM

O arquivo de **STATE** é mandatório para o funcionamento do terraform, esse arquivo é responsável por armazenar **TODOS** os recursos já criados pelo terraform, ele mantém o tracking dos recursos criados e seus metadados.

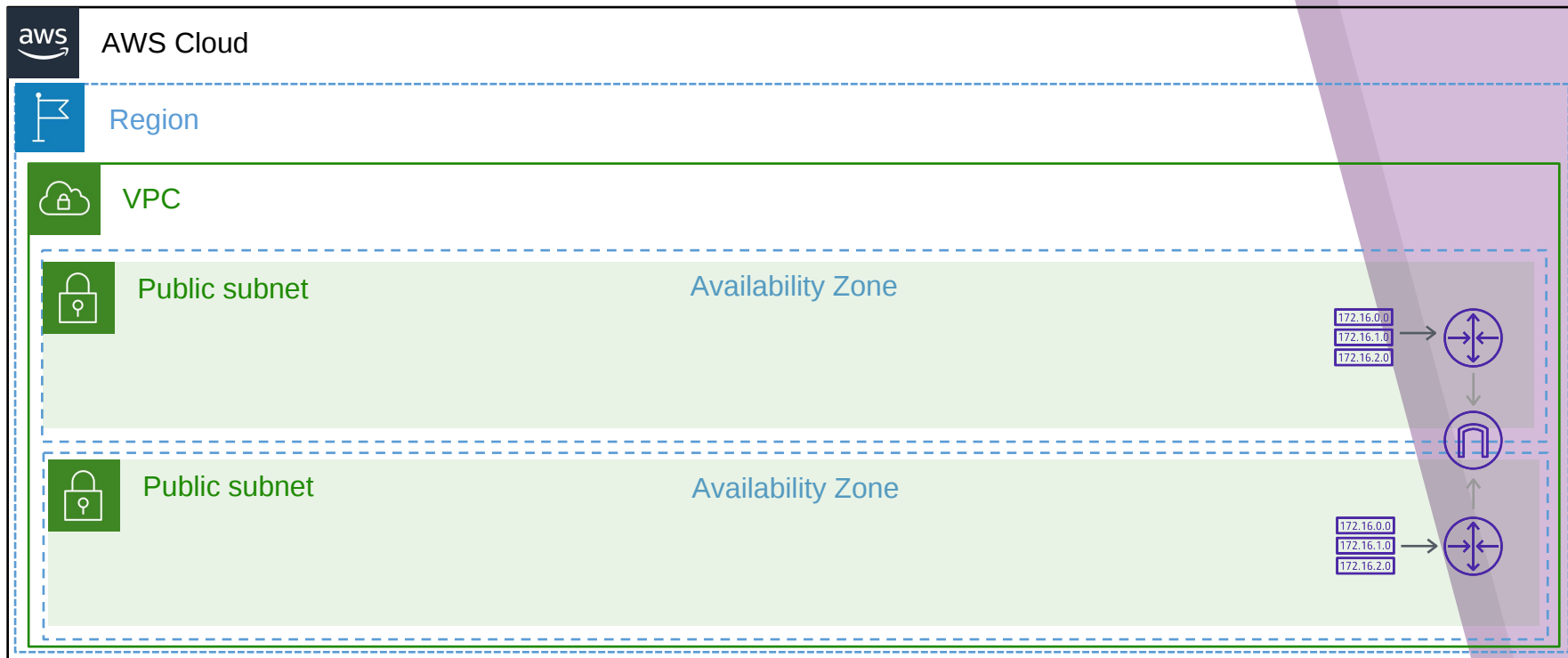
O **STATE** também funciona como um mapeamento de dependências, no arquivo há as informações de quais recursos dependem de quem e a ordem de modificação ou deleção.

Outra utilidade do arquivo de **STATE** é a performance, muitos dados são armazenados no arquivo de estado, como atributos, tags, etc.. e muitos providers possuem restrições de uso de API e logo podem fazer com que uma execução de um plano demore desameadamente.

Uma boa pratica é armazenar o arquivo **STATE** em um local remoto como um Bucket S3 por exemplo, o armazenamento do arquivo de **STATE** na nuvem você irá prevenir problemas de perda do arquivo e mudanças simultâneas e também garante que você esta sempre com a ultima versão das configurações presentes.

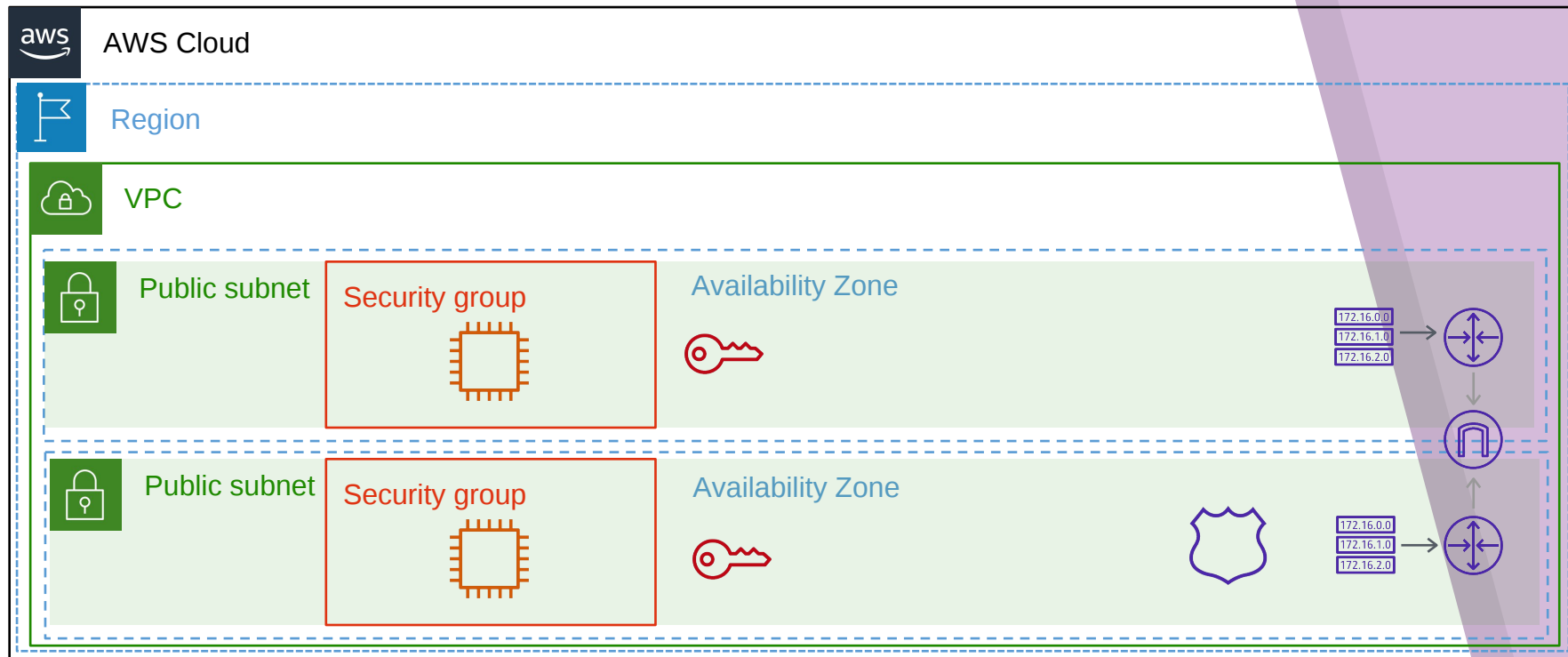
Criação de Infraestrutura VPC

Nesse diagrama há uma estrutura de redes que iremos criar à partir de um módulo do terraform.



Deploy da app **slacko**

A arquitetura abaixo é umas das opções de deploy da nossa api, para relembrar a nossa aplicação é composta por uma aplicação em python que utiliza como backend um mongodb.



Utilizando Módulos VPC

```
module "vpc" {  
    source = "terraform-aws-modules/vpc/aws"  
    name = "my-vpc"  
    cidr = "10.0.0.0/16"  
    azs      = ["us-east-1a", "us-east-1c"]  
    public_subnets = ["10.0.101.0/24", "10.0.102.0/24"]  
    enable_nat_gateway = false  
    single_nat_gateway  = false  
    enable_vpn_gateway = false  
    one_nat_gateway_per_az = false  
    enable_dns_hostnames = true  
    create_egress_only_igw = true  
    tags = { }  
}
```

Exercícios 1 e 2

Exercício 1:

Transformar o terraform que provisiona a app slacko-app em um módulo, com os seguintes parâmetros:

- tags
- nome dos recursos
- subnet_cidr
- vpc_id

Favor fazer upload dos códigos referentes aos 2 exercícios no github/gitlab e enviar o link do repositório

Exercício 2:

Criar um servidor jenkins na AWS utilizando uma EC2 do tipo t3.medium em uma rede pública, para atingir esse objetivo deve-se utilizar a imagem do ubuntu e liberar a porta 8080 para acesso ao jenkins.

E executar o comando:

```
$ cat /var/lib/jenkins/secrets/initialAdminPassword
```

Utilizar o seguinte script abaixo como user_data

```
sudo apt-get update

sudo apt install openjdk-11-jdk -y

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins -y

sudo systemctl start jenkins
```

Workspaces

Como já vimos, ao utilizar o Terraform o mesmo cria um arquivo de tfstate, que irá armazenar a infraestrutura criada, com isso o primeiro problema surge. **E se precisamos criar uma segunda infraestrutura utilizando o mesmo código do terraform?**

O workspaces vem para solucionar esse problema, com o workspaces é possível provisionar vários ambientes utilizando o mesmo código.

O workspace cria um diretório chamado **terraform.tfstate.d** caso esteja usando o backend local, dentro desse diretório o terraform cria outros diretórios que recebem o nome do respectivo workspace. O terraform também cria um variável chamada **terraform.workspace** com o nome do workspace.

```
# Cria um workspace chamado prod
$ terraform workspace new prod
# Lista todos os workspaces criados
$ terraform workspace list
# Muda/Seleciona a workspace para prod
$ terraform workspace select prod
# Apaga a workspace chamada prod
$ terraform workspace delete prod
```

Workspaces

```
locals {  
  envs = {  
    default = {  
      instance_type = "t2.micro"  
      ami = "ami-0ff8a91507f77f867"  
      region = "us-east-1"  
    }  
    dev = {  
      instance_type = "t2.medium"  
      ami = "ami-0ff8a91507f77f867"  
      region = "us-east-1"  
    }  
    qa = {  
      instance_type = "t2.xlarge"  
      ami = "ami-0ff8a91507f77f867"  
      region = "eu-east-1"  
    }  
    prod = {  
      instance_type = "t3.large"  
      ami = "ami-0ff8a91507f77f867"  
      region = "sa-east-1"  
    }  
  }  
  env_vars = contains(keys(local.envs), local.env) ? local.env : "default"  
  workspace = merge(local.envs["default"], local.envs[local.env_vars])  
}
```

DEV



```
locals {  
  env = terraform.workspace  
}
```

```
output "variaveis" {  
  value = local.workspace  
}  
  
output "region" {  
  value = local.workspace["region"]  
}
```


Criando **MÓDULOS**

Os módulos tem a função de criar um ambiente ou uma determinada arquitetura, um módulo na mais é que um conjunto de recursos, variáveis, outputs, locals, etc, do próprio terraform. Um código escrito em terraform pode facilmente se tornar um modulo com pequenos ajustes, e esses módulos podem ficar armazenados remotamente, como um GIT

Abaixo seguem alguns casos de uso de módulos.

- Criação de um ambiente padronizado para desenvolvimento, exigindo tags, e determinadas configurações de segurança e compliance;
- Criação de um projeto de infraestrutura recorrente, como um LAMP (Linux, Apache, Mysql e PHP) como o wordpress.
- Padronização na criação de recursos, forçando parâmetros que são opcionais em parâmetros obrigatórios, etc...

Criando MÓDULOS

No nosso exemplo vamos transformar o nosso terraform que provisionou o ambiente para o funcionamento da app Slacko em um módulo, onde será provisionada a nossa aplicação na **aws**, sem que o usuário final precise lidar com todos os outros recursos, como security_group, chaves ssh, etc..

Para criar um modulo localmente basta criar um diretório chamado modules onde ficarão todos os arquivos *.tf e expondo os parâmetros de configuração através de variáveis.

Os outputs criados dentro do modulo funcionam como atributos de saida do modulo, podendo ser convocados com a seguinte sintaxe.

`module.slackoapp.nomedooutput`

Obs: caso algum arquivo seja referenciado dentro do modulo, como templates, chaves ssh, etc.. deve-se concatenar a variável path.module ao caminho do arquivo conforme abaixo:

`file("${path.module}/files/mongodb.sh")`

```
module "slackoapp" {  
    source = "../modules/slacko-app"  
    vpc_id = "vpc-01d1fda4da478a34d"  
    subnet_cidr = "10.0.102.0/24"  
}
```

Recurso **NULL** Resource e Remote **EXEC**

O terraform possui um recursos chamado `null_resource`, ele não cria nenhum recurso, porém ele pode ser acionado via um gatilho bem como usar outros tipos de provisioners e connections.

Um `null_resource` precisa de uma trigger para ser convocado, no exemplo ao lado estamos utilizando o `null_resource` em conjunto com o provisioner “remote-exec” nesse caso de uso queremos executar um comando no servidor após ele ser provisionado, esse recurso pode ser bem útil em caso de aplicações que possam exigir pós configuração.

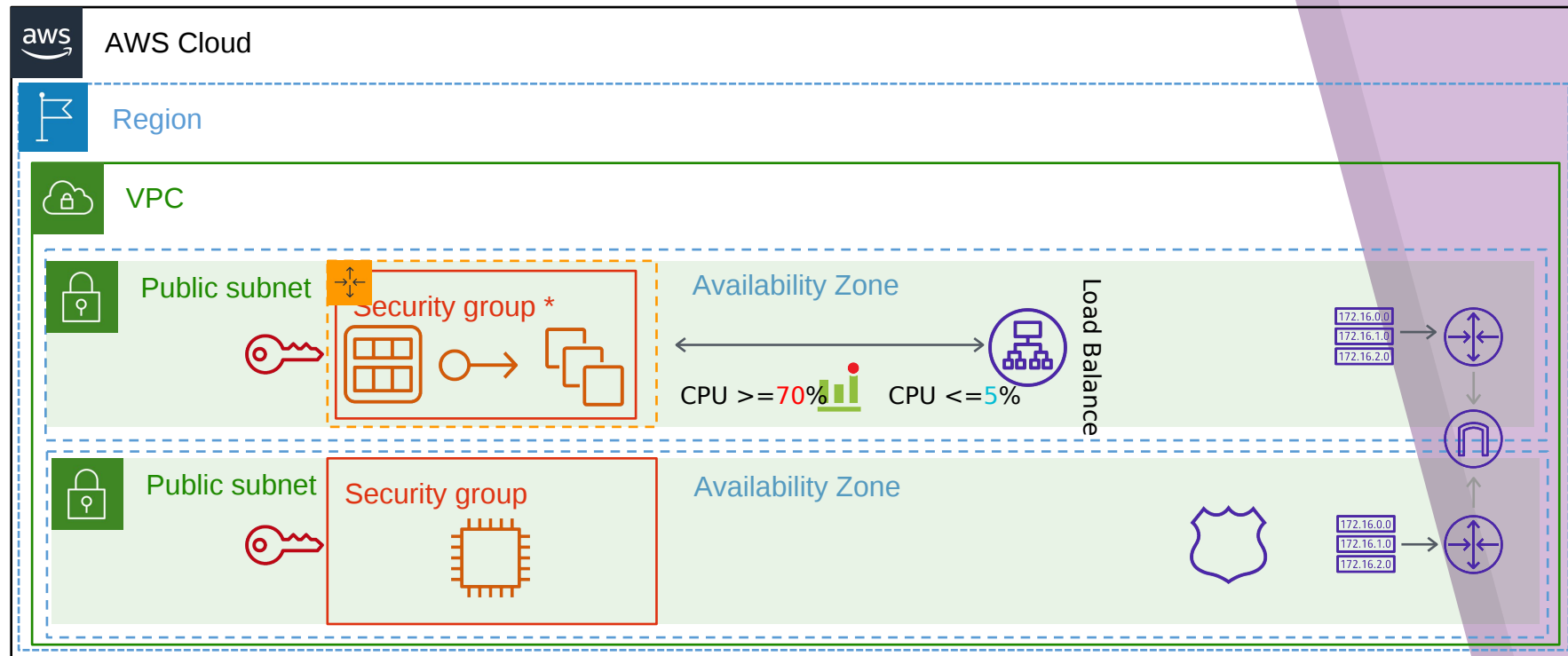
```
resource "null_resource" "gethostname" {
  triggers = {
    instance = module.slackoapp.skacko-app
  }

  connection {
    type      = "ssh"
    user      = "ec2-user"
    private_key = file("slacko")
    host      = module.slackoapp.skacko-app
  }

  provisioner "remote-exec" {
    inline = [
      "cat /etc/hostname",
    ]
  }
}
```

Deploy da app **slacko** com **ALB** e **ASG**

A arquitetura abaixo é umas das opções de deploy da nossa api, já utilizando um modelo de autoscaling e load-balance, mais parecido com um ambiente em produção



Comando **TAINT/REPLACE** do **TERRAFORM**

A função Taint serve quando se faz necessária a substituição de um recurso, seja por mau funcionamento ou caso alguma dependência não inicie o processo de substituição.

Obs: Essa função está depreciada e no lugar dela está a função --replace

```
$ terraform apply -replace="aws_instance.example[0]"
```

