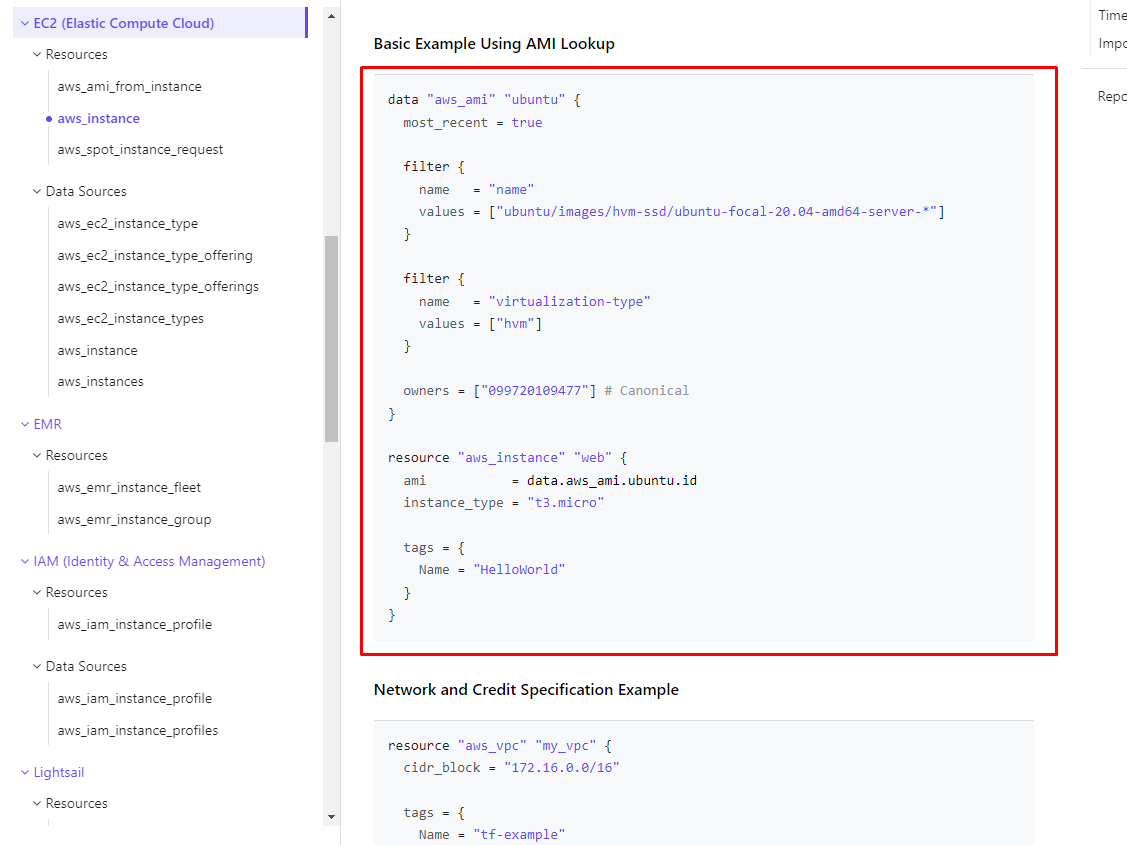
# 

# TERRAFORM:

Documentação do Terraform:

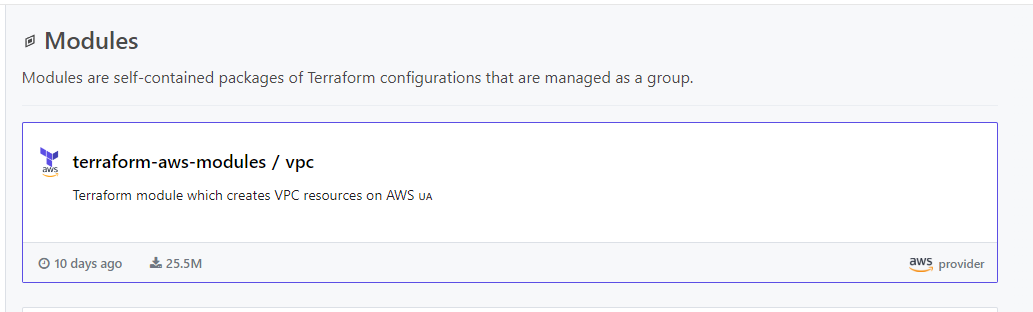
Existem as Documentação dos Provider: AWS, Google Cloud Plataform, e Alibaba Cloud.

Na guia na parte de documentation na AWS temos a barra de filtros que explica como nós podemos criar por exemplo algo simples como uma Instancia EC2:

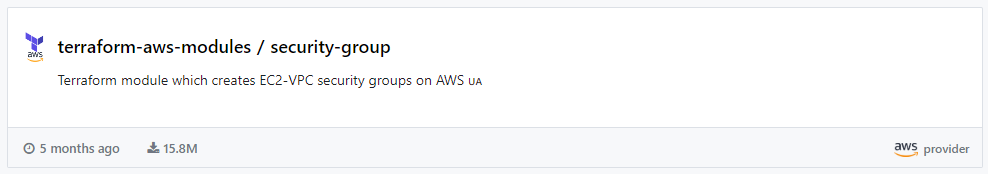


Temos as Documentação dos Modulos:

Os Modulos são conjuntos de códigos que serve para um fim especifico, por exemplo esse conjunto de código de modulo serve para criar uma estrutura de VPC:

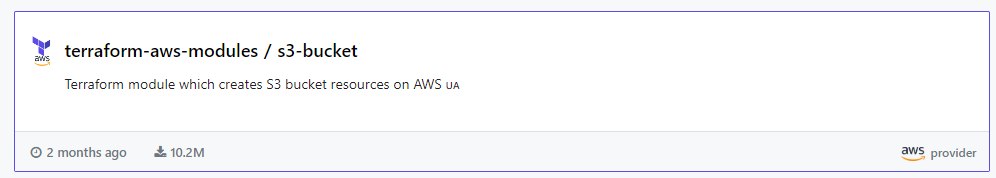


Esse outro modulo e uma estrutura de código que serve para criar um Security Group na AWS:

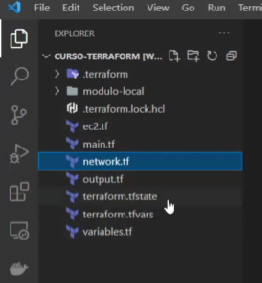


Temos milhares de Modulos, criados pelos Provider.

Modulo pra criar uma Bucket no S3:



Estrutura de Arquivos e de Diretórios do Terraform:



Importante, o Terraform.tfstate e um arquivo que o próprio Terraform vai criar localmente. O Terraform cria este arquivo State, que e onde ele armazena noções de tudo aquilo que ele gerou.

Importante outra parte, criar um arquivo com a extensão tfvars, essa parte do código e um arquivo que você declara o valor das variáveis que você vai utilizar dentro do seu código do Terraform, arquivo especificamente pra isso.

Chave e o valor, e o arquivo tfvars ele serve pra isso.

Além disso o que o Terraform cria faz parte da configuração do Terraform que e o terraform.lock.hcl

Temos também a pasta: terraform essa pasta terraform e onde o terraform vai fazer o download dos providers e dos modulos que você virá a usar dentro do seu código por exemplo:

Se você indica que quer usar um Provider da AWS, ele vai fazer o download da configuração desse Provider da AWS.

E assim por diante

Na questão dos modulos ele irá fazer o download de diversos modulos, para essa pasta, e você pode criar um próprio modulo local.

Normalmente criamos um diretório dentro do modulo local vamos colocar outros arquivos terraform, que vão fazer parte desse modulo.

ESTRUTURA DE BLOCOS DA LINGUAGEM HCL:

* Existem outros tipos de Blocos básicos no Terraform
* Terraform
* Providers
* Resources
* Data
* Module
* Variable
* Outputs
* Locais

Vamos criar o primeiro bloco do terraform:

Dentro desse bloco vão diversas configurações, que são as configurações básicas do terraform

Bloco dos Providers:

Você vai dar o nome do provider, por exemplo se for AWS (todo bloco de provider tem sua configuração especifica que será mostrada)

Bloco do Resource:

Você dá o nome de Resource, você quer criar, podemos criar uma instancia EC2

Exemplo:

Resource “aws\_instance” “aqui eu posso dar o nome que eu quiser” {

}

Por exemplo eu posso chama-la de VM1, esse nome podemos usar para referenciar esse Resource em outros blocos posteriormente. Então e muito importante que seja um nome que faça referencia a aquilo que você está criando.

Bloco do Data:

O bloco data funciona da mesma forma que o bloco de Resource:

Damos o primeiro nome ao tipo de Data, o Data vai buscar informações de fora do código do terraform, pra ser usado no terraform:

Por exemplo: uma data “aws\_ami” “ami” {

Vai buscar informações de uma imagem da AWS, e da mesma forma temos que usar um nome que traga referencia trazer de informação pra cá.

}

Bloco do Modele:

Module “name” {

Vamos só dar o nome pra ele, por exemplo se eu quiser usar um modulo de VPC e o bloco de modulo, ele serve tanto pra referenciar o modulo que você cria, no subdiretório da sua configuração do terraform, como também serve pra referenciar o modulo da comunidade, do registro do terraform.

O bloco e o mesmo, só as configurações internas dele que mudam.

}

Bloco de Variables:

Variable “name” {

As variáveis, são uma declaração que você faz, que você vai usar dentro da configuração do terraform, da mesma forma vc vai colocar o nome.

Aqui dentro vai a configuração dessa variável que será mostrada.

}

Bloco output:

Output “name” {

É um recurso do terraform, pra pegar alguma informação de dentro do código do terraform, de dentro de onde o terraform criou, e fazer o output pra fora dessa documentação, pra poder ser usado por um outro recurso da sua máquina, ou dentro da sua Pipeline ou utilizar para o que quiser

Output ele pega alguma informação de dentro do terraform e coloca pra fora

}

Bloco Locals:

Locals {

O Locals basicamente ele serve pra você pegar funções ou expressões que você usa muito repetidamente, dentro da sua configuração do terraform, da um nome pra ela, ao invés de você ter que ficar repetindo aquele código toda vez, vc apenas chama esse código que você declara aqui, e pra isso que funciona o Locals.

}

CONFIGURAÇÃO DOS BLOCOS DO TERRAFORM

Como vamos fazer a configuração do bloco do terraform:

* Dentro dos 06 Blocos e Possível fazer esses 6 tipos de configurações:

1 - required\_Version

2 – required\_Provider

3 – back-end

4 – Cloud

5 – Experiements

6 – Provider\_meta

Dentro do bloco terraform, a primeira coisa que podermos configurar, e a versão que queremos usar no nosso código do terraform:

# Required\_Version:

Terraform {

Isso nós fazemos mencionando o require\_version:

Require\_version: “value” e aqui dentro colocamos a versão ou o conjunto de versões que queremos que nosso código funcione.

}

Onde a regra é:

/\*Iremos mostrar como os operadores de Version funcionam:

Para uma versão especifica eu utilizo: 1.1.0 ou = 1.1.0

Para versões do terraform > ou = a 1.0.0 ou versões < 1.2.0

Nota: que só o intervalo de Versões de terraform entre estes 02 vai funcionar, ## required\_version = " >= 1.0.0, < 1.2.0" podemos fazer dessa forma também

      para versões do terraform ~> 1.0.0 #     Este código vai funcionar com todos as versões de terraform desde a 1.0.0 até a 1.1.n

    \*/

    required\_version = "= 1.1.0"

# Required\_Provider:

Esse serão os providers que iremos utilizar ou usar na configuração do terraform, por exemplo o Provider da aws:

Required\_providers {

Aws {

## E dentro do next block da aws, vamos ter que indicar qual a versão, e da mesma forma do require\_version desde uma versão única, ou alguma coisa da mais atual.

Version = “2.70.0”

Source = hashcorp/aws

}

## Podemos ter mais de um Provider na configuração do terraform

Azurerm {

Version = “1.70.0”

Source = “hashcorp/Azurerm”

}

## podemos ter infinitos providers declarados na estrutura do terraform, isso e encontrado na documentação do terraform.

}

Basicamente o **Backend**, ele serve para guardar o State do terraform de forma remota, no caso mencionamos o “S3”, eu guardaria o State em uma S3 bucket da aws.

# Terraform CLI:

Linha de comando do terraform:

Verificando os 05 comandos básicos do terraform:

Terraform init – **Prepare you working Directory for Other commands**

Terraform validate – **Check Weather the configuration is valid**

Terraform plan – **Show change required by the current configuration**

Terraform apply – **Create and Update Infrastructure**

Terraform destroy – **Destroy previously-created infrastructure**

-Existem outros comandos

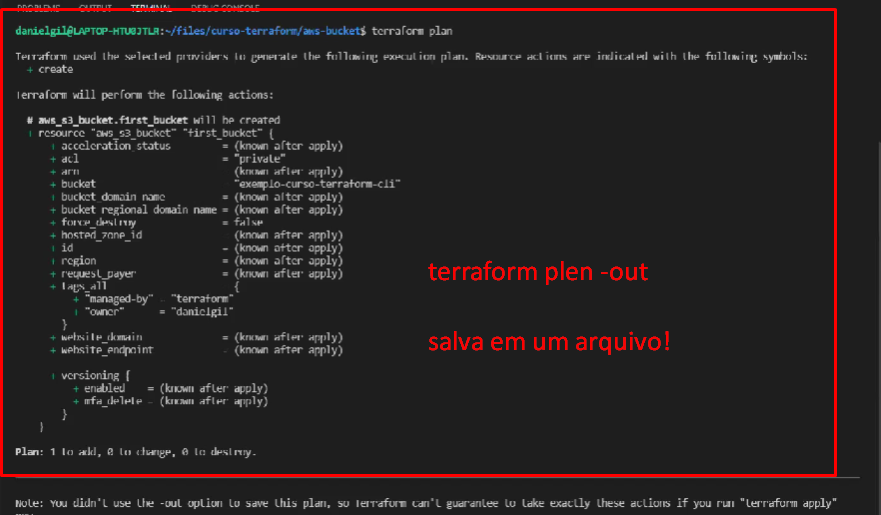
Resumo dos comandos importantes:

**Terraform init** – inicializa o terraform, baixando a configuração do Provider, além do arquivo terraform.lock.hcl

**Terraform validate** – E um comando que serve pra ver se o código que você escreveu do terraform está valido, e não contém nenhum erro.

**Terraform plan** – Serve pra mostrar na tela, um plano do que o terraform vai fazer se ele rodar essa configuração que colocamos.

Um comando que é importante e o **-out** que ele serve pra pegar esse plano, que ele esta demostrando e salva em um arquivo:



**Terraform plan -out**=(nome do arquivo)

**Terraform apply** – O terraform apply vai fazer o Deploy de tudo o que estamos fazendo no nosso código.

Terraform apply plan.out (ele criara o plano sem perguntar nada)

**Terraform destroy** - Destrói todo o ambiente ou environment da cloud.

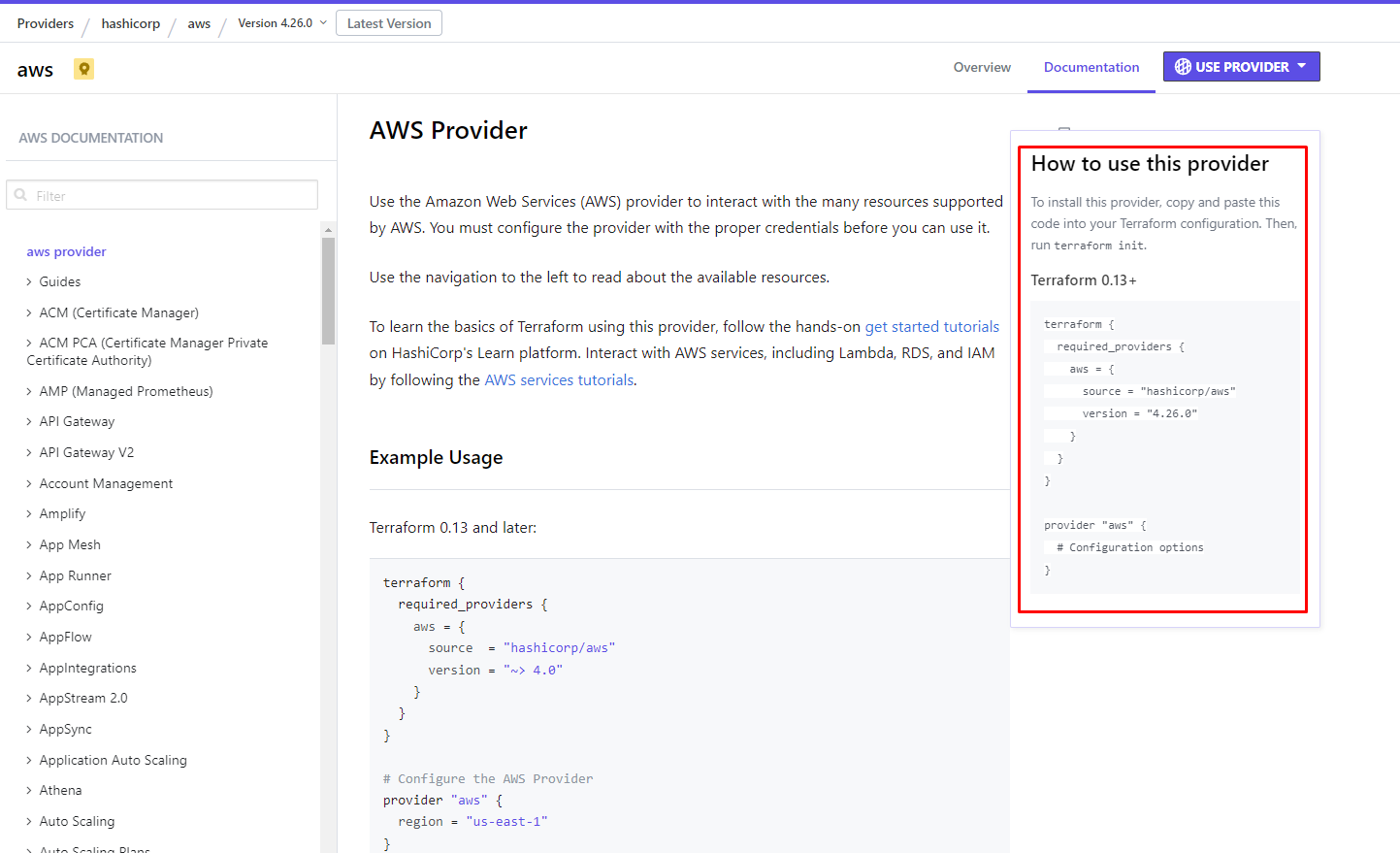
Terraform fmt – ele tem um sentido estético puramente assim, quando o nosso código está muito grande ele deixa visualmente mais fácil de entender.

O terraform fmt organiza de uma forma todo o nosso código, ou terraform formate.

# Criando uma Bucket no Terraform:

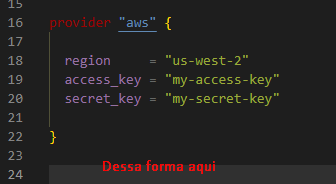
Importante, pegar a documentação do Provider da AWS na documentação do terraform:

Dessa maneira, econômico o retrabalho, já que esta tudo lá na documentação do terraform



Depois de configurado o Provider, dentro do bloco do terraform, devemos configurar o bloco do Provider que vai receber as informações de região e outras informações especificas que precisamos:

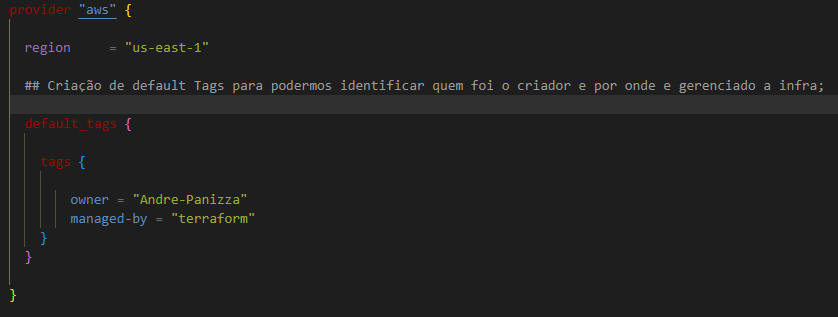
(Nota: colocaremos além da região as nossas credenciais também:)



Existem varias formas de colocarmos nossas credencias no terraform, utilizando Environment Variables, porque se deixarmos dentro do bloco do terraform, as nossas credencias (vão ficar gravadas no nosso código) se usarmos como Evironment Variables, não teremos esse problema.

Podemos colocar dentro do nosso Provider, uma solução bem útil, que são as default tags – os defaults tags, são tags que vai ficar escrita ali no nosso Provider e ela vai ser usada para todos os recursos que aquela configuração for criando na AWS.

**Dessa forma:**



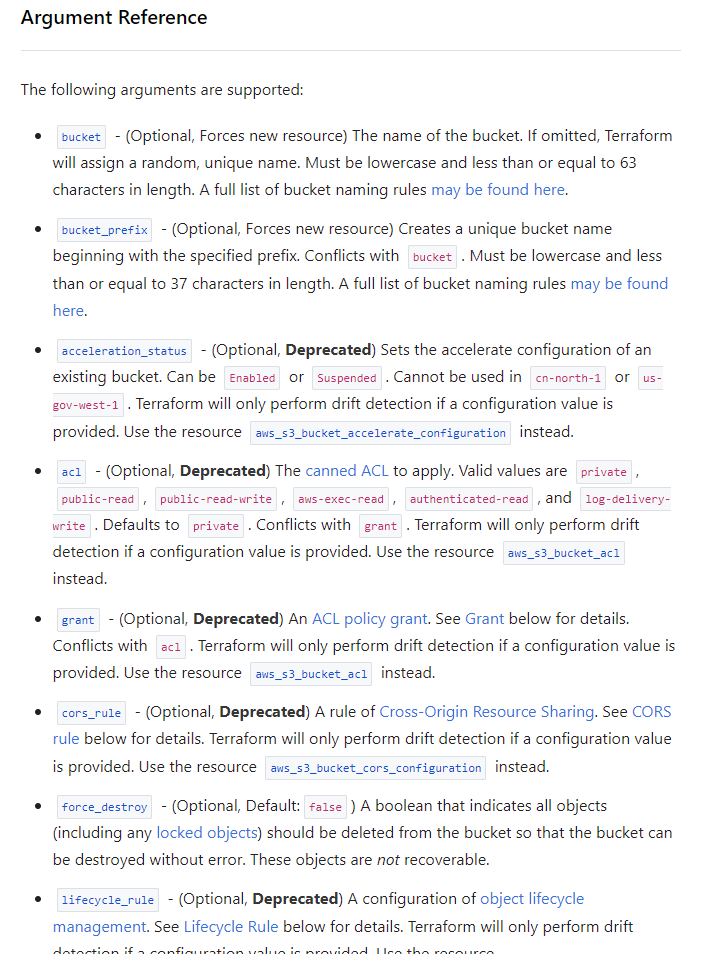
# Criando um segundo arquivo para inicializarmos a nossa bucket:



Vamos voltar na documentação do terraform:

Filtrando como Bucket S3, rolando a pagina mais para baixo na documentação, temos a opção de Argument Reference

Aí temos tudo que podemos utilizar na criação do nosso recurso:



Referente as variáveis de Credenciais de Ambiente devemos:

$ export AWS\_ACCESS\_KEY\_ID="SDSDEFOIYHJ34kJHVJ"

$ export AWS\_SECRET\_ACCESS\_KEY="AWSDXSFESFE (\*8866"

$ export AWS\_REGION="us-west-2"

Nota: colocar esse bloco de configuração, colando na nossa linha de comando:

Assim os arquivos serão baixados, para nossa máquina básicos referente a configuração do providers.

Importante, antes de rodar o Terraform apply:

Terraform validate – para validar o código

Terraform fmt – para formatar o código

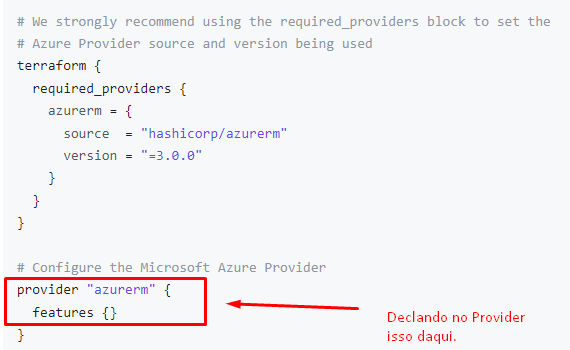
Terraform plan – para ver o plano de construção, se você quiser salvar um arquivo com o terraform -out= (nome do arquivo)

E por fim estando tudo certo o terraform apply, confirmando a construção do seu deployment.

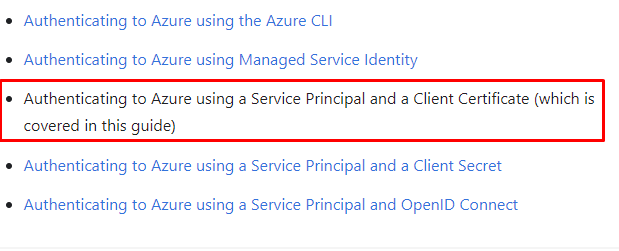
## Criando um Storage Account na Azure ou AWS:

Dentro da documentação da Azure, devemos copiar o bloco de Use Provider da Azure

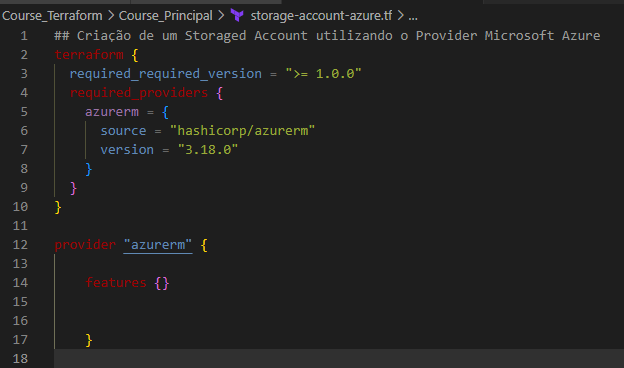
Depois temos que configurar o Provider que pode ser configurado utilizando uma das opções de acesso da documentação da Azure:



A opção escolhida de autenticação foi:



Como consta no meu bloco de código:



# 2 Steps – Vamos criar um novo arquivo chamado:

Storaged-account.tf



Na Azure funciona um pouco diferente, por isso que a criação da Storage-account da Azure, porque podemos trabalhar diversas ferramentas do terraform neste Hands On

Para criarmos uma Storage-Account na Azure, primeiro precisamos criar um Resource-Group, dentro deste Resource-Group vamos ter Storaged-account, dentro dela vamos criar um Container:

-Vamos criar 03 blocos de Resource diferentes.

O 1° deles, Resource Group

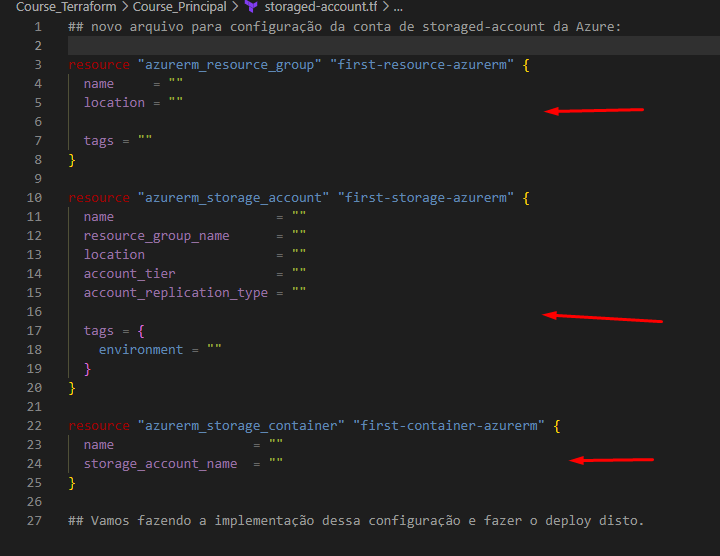
2° deles será o Storaged-Account

3° e último bloco será o Azure-Storaged-Container

Temos que verificar na documentação, tudo o que é Required e Optional para colocarmos somente aquilo que precisamos que e requirido o campo.

Por hora deixaremos dessa forma, em branco os campos.

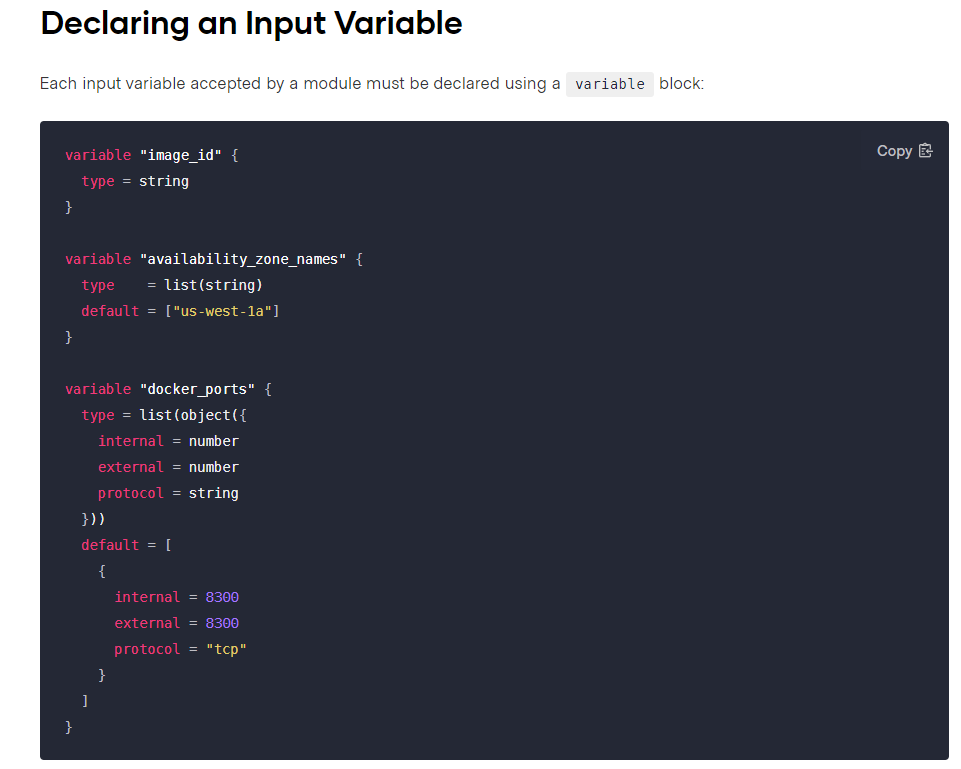
Na aula seguinte iramos passar cada um dos blocos:



# Trabalhando com Variáveis no Terraform:

Configuração da storaged-account, trabalhar com variáveis no terraform:

Input Variables, para usarmos as variáveis devemos utilizar a declaração de bloco de variáveis



Ela aceita este tipo de Arguments:

Default – **a default value which then makes the Variable Optional**

Type – **This Arguments specifies what value types are acepted for the Variable**

Description – **This specifies the input Variable documentation**

Validation – **A block to define validation rules, usually in addition to type constrains**

Sensitive – **Limits Terraform UI output when the Variable is used in configuration**

Nullable – **Specify if the Variable can be null within the module**

Quando declaramos o tipo da Variable, se ela e uma string, um inteiro ou um booleano:

Nota: ela pode receber todos esses tipos de opções dentro do type:

- String

- Number

- Bool

The type constructors allow you Specify complex types such as collections

- List (<Types)

- set (<Types)

- map(<Types)

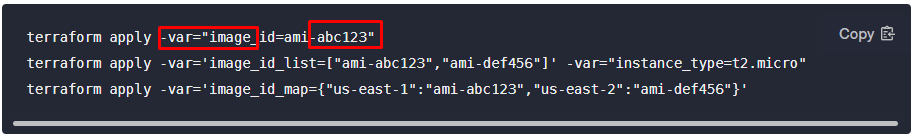
- Object ({<ATTR NAME> = <Types, ....})

- Tuple ([<TYPE]>, ... ])

Existem varias formas de colocarmos os valores de nossas variáveis:

- A 1° e dentro do próprio bloco da variável, nos colocamos o valor default dessa variável

A 2° forma, e informando isso via linha de comando, usando o terraform apply, ou terraform plan com esta opção seguida do nome da variável:



Nós também podemos criar arquivos que terminam com tf.vars, ou tf.vars.json

Ele precisa ter o nome da variável e o valor

Ou se você colocar a variável em um arquivo, ela pode ter a terminação em terraform. tfvars ou terraform. tfvars.json.

Ou então qualquer outro nome que você queira que tenha a terminação .auto.tfvars.json

Se quisermos podemos dar para o terraform os valores das variáveis como variável de ambiente onde fazemos um Export (já seguido pelo nome da variável e o valor da variável dessa forma:



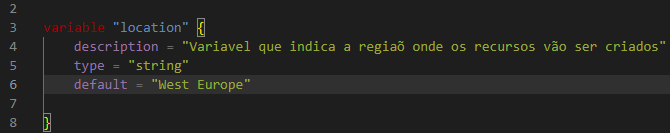
# Inicializando com a criação dos arquivos das variáveis:

Voltando ao meu Storaged-Account, digamos que eu queira criar uma variável location

Então temos:

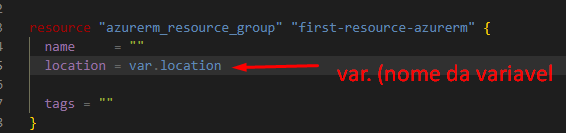
O bloco da variável = “nome da variável”

Depois vamos colocar os Arguments que vimos na documentação do terraform de variáveis

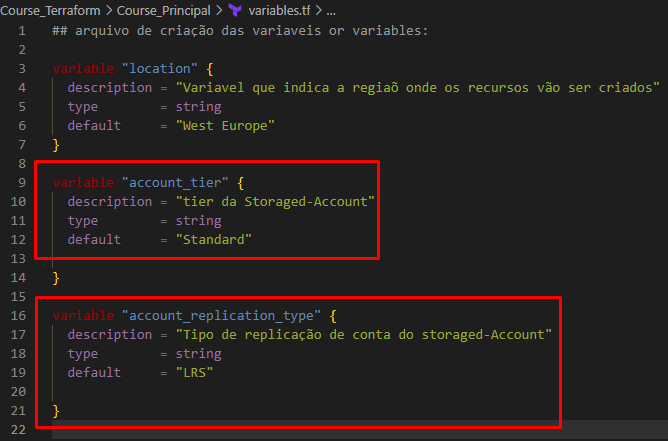


* O Description não e obrigatório, mais serve pra quem for ler o seu código
* O tipo também não e obrigatório
* E aqui está com o valor default que eu tenho a região na Azure

# Agora, como eu uso a minha variável no bloco storaged-account:

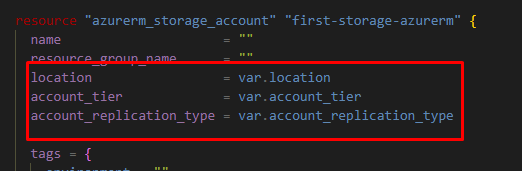


Criando mais duas variáveis para account tier, account\_replication\_type



Voltamos na documentação para verificarmos o valor default do account\_tier e do account\_replication\_type.

Depois de preenchidos os valores default, temos que preencher com a nosso valor da variável:



Dessa forma, e aí está pronto o preenchimento dessa parte do bloco.

Outra forma de sobrescrevermos a variável e declaramos ela no Command line temos 03 formas de fazer isso:

export TF\_VAR\_location=” Brasil South”

Terraform plan -var=’location=Brazil South’ (vale tanto para o plan, quanto para o apply.

Uma outra forma de sobrescrever as variáveis e criando um arquivo terraform.tfvars



Fazendo dessa forma:



Eu também sobrescreveria o valor default que está aqui.

Também podemos criar um arquivo com qualquer outro nome como:

Var.tf

Nota: Se tivermos mexendo com um valor sensível ou credencias podemos usar o sensitive

Se você fizer desta forma, o terraform não vai trazer o valor dessa variável quando ele fizer o plan!

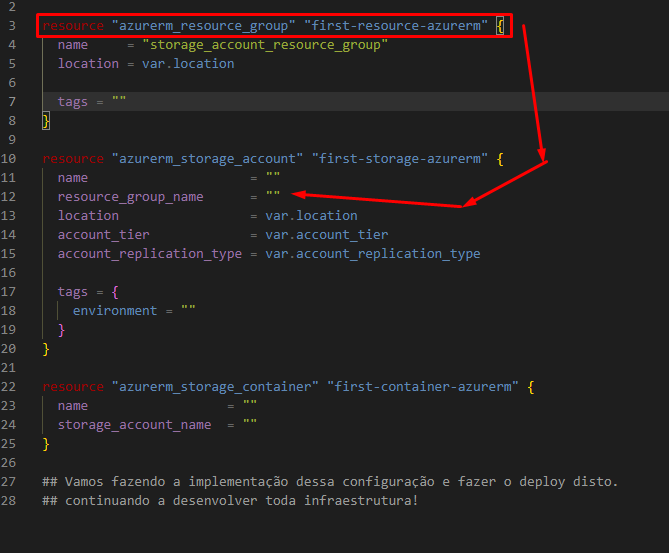
**REFERENCIANDO ATRIBUTOS DE OUTROS BLOCOS:**

Agora vamos referenciar atributos de outros blocos como eu faço pra resolver esta questão:

Neste link, e ensinado a referenciação de atributos desses blocos

[References to Values - Configuration Language | Terraform by HashiCorp](https://www.terraform.io/language/expressions/references)

Como podemos referenciar valores de um bloco dentro de outro bloco:



Escrevendo dessa forma:

azurerm\_resource\_group.first-resource-azurerm.name

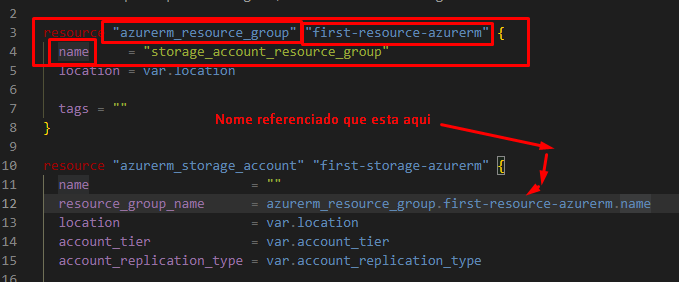
então temos o nome do resource: azurerm\_resource\_group.first-resource-azurerm.name

1° O nome do recurso.

2° depois o nome que eu coloco opcional

3° e por ultimo o campo (neste caso o name)

**O terraform vai entender, que o que estamos referenciando e o nome que está aqui:**



Esse pratica de referenciar blocos do terraform, serve justamente para todos os blocos citados aqui abaixo:

-**resource**

**-module**

**-variable**

**-Locals**

**-Output**

**-Data**

OBS: com qualquer um desses blocos podemos fazer essa referenciação de atributos

Na documentação do Azure, nos temos mais abaixo mais Arguments, em **Atribute References:**

Como:

**ID** – The ID of the Storaged-Account

**Primary\_location** – The Primary location of the Storaged-Account

**Secondary\_location** – The Secondary location of the Storaged-Account

**Primary\_blob\_endpoint** – The endpoint URL for Blob storage in the Primary location

**Primary\_Blob\_Host** – The Hostname with port if applicable for Blob storage in the Primary location

**Secondary\_Blob\_Host** – The Hostname with port if applicable for Blob storaged in the Secondary location

**Primary\_Queue\_endpoint** – The endpoint URL for Queue storage in the primary location

**Primary\_Queue\_Host** – The Hostname with for if applicable for queue storage in the primary location

**Secondary\_Queue\_endpoint** – The endpoint URL for Queue storage in the Secondary location

**Secondary\_Queue\_endpoint** – The endpoint URL for Queue storage in the Secondary location

**Secondary\_Queue\_Host** – The hostname with for if applicable for queue storage in the Secondary location

**Primary\_table\_endpoint** – The endpoint URL for table storage in the location

Se quisermos referenciar um ID por exemplo, podemos fazer dessa forma:

Azurerm\_storage\_account.first\_storage\_resource.id

É o terraform já vai entender que estamos referenciando o ID, porem ele ainda não foi listado, declaramos antes mesmo de saber o valor dessa ID.

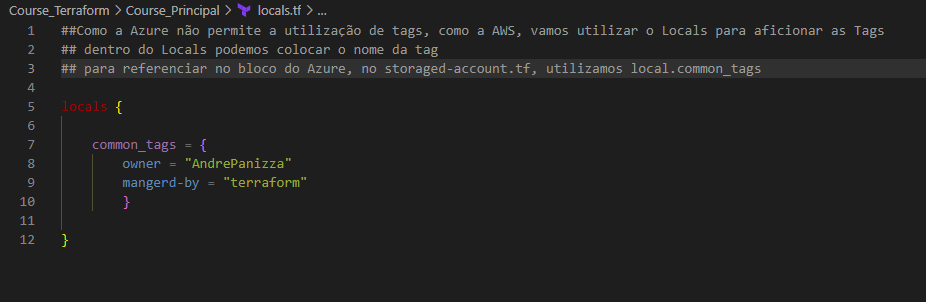
Iniciando

Local Values:

Local values é um recurso do terraform que permite darmos um nome para uma determinada função, e quando essa função e mencionada muitas vezes no código, nos podemos usar o local values para evitar ficar escrevendo repetidas vezes a mesma função.

Link da documentação que fala sobre o Local Values:

[Local Values - Configuration Language | Terraform by HashiCorp](https://www.terraform.io/language/values/locals)



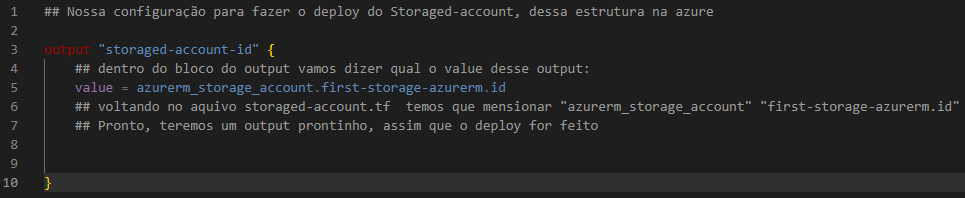
Iniciando com Outputs no Terraform:

Nada nada mais e enviar pra fora do terraform, alguma informação que você criou, ou recebi dentro do meu código:

Exemplo no VScode:

Mantendo organização e estrutura de arquivos





Um dos comandos importantes para vermos o output como “ID” que foi mencionado e utilizando o comando:

Terraform output storaged-account-id:



Ou terraform output (você terá todos os outputs que vc tiver na sua configuração de uma vez só)

Fazendo o Deploy na conta da Azure:

Terraform init – para inicializar o terraform e baixar as configurações dos Providers

Terraform fmt – Para formatar o código e organiza-lo

Terraform validate – Para validar se a estrutura está correta.

Terraform Plan – Para verificar o plano que será criado, e sua estrutura]

Terraform apply – Para iniciar a construção da Aplicação no Provider

E por fim: Terraform destroy

Iniciando com LOCAL STATE:

Uma das bases do funcionamento do terraform e o State, o State nada mais e que um arquivo que o terraform cria automaticamente onde ele armazena informações de todos os recursos que ele gerir.

E graças a informação que está nesse arquivo, nesse State que o terraform funciona que vc poderá aplicar a mesma configuração varias vezes na mesma Config, que ele não vai setar repetidamente os mesmos recursos, se por acaso recurso já foi criado, ele não irá criar novamente.

Ensinar como funciona o State:

[State | Terraform by HashiCorp](https://www.terraform.io/language/state)

Aqui temos um Overview de como funciona o State:

Vamos pro VScode:

Quando pegamos as Credentials da AWS e copiamos e colamos e jogamos no Terminal e fazemos depois um terraform init, ele irá fazer o download da configuração da AWS

Vamos fazer um terraform validate – para checar se a configuração está ok.

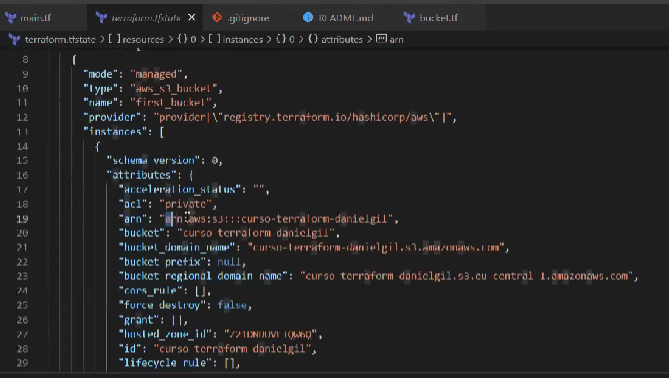
E depois vamos gerar um arquivo chamado terraform plan -out=” plan.out”

O arquivo será gerado!!

Vamos fazer um terraform apply plan.out

Ele gera um arquivo chamado **terraform.tfstate** (que o terraform cria automaticamente)

Dentro dele, tem toda a informação da bucket que nós criamos:



Importante sobre o State:

Mesmo que você defina um variável como uma variável sensível (sensitive true), aqui no State vai aparecer esta informação, vale sempre tomar cuidado onde iremos salvar o seu State.

Dando um terraform destroy, todas as informações do terraform.tfstate vai ser perdida, ficando apenas algumas configurações básicas no arquivo.

O terraform State, por segurança cria o terraform.tfstate.backup e dentro deste State backup, tem a última versão desse State com as informações da bucket que eu tinha.

Falando sobre o arquivo lock que vimos sendo criado: fazendo um terraform apply sem mencionar o plan.

Qual e a utilidade do tfstate.lock, pro terraform o arquivo State e muito importante.

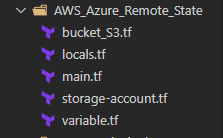
O arquivo .terraform.tfstate.lock nada mais que um arquivo pra impedir que outras pessoas mexam no arquivo ao mesmo tempo.

Tem outras coisas que podemos fazer com o State, fazendo um remote por exemplo!

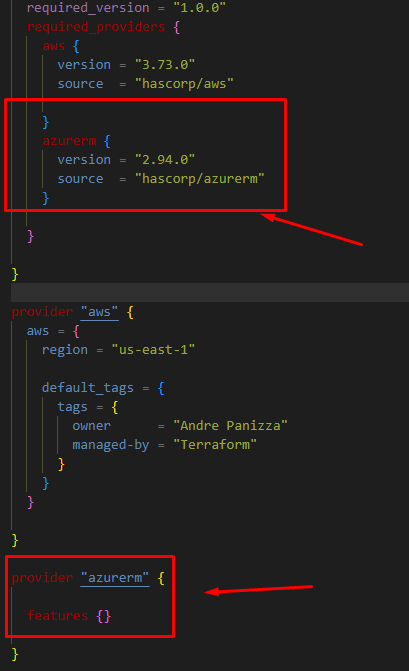
Remote State No terraform:

O remote State nada mais e do que armazenar remotamente o arquivo do State do terraform, existem várias opções para que possamos fazer isto, mais o que vamos fazer e usar uma bucket S3 e usar uma Storage-Account.

Vamos criar uma pasta chamada AWS-Azure-remote-State:



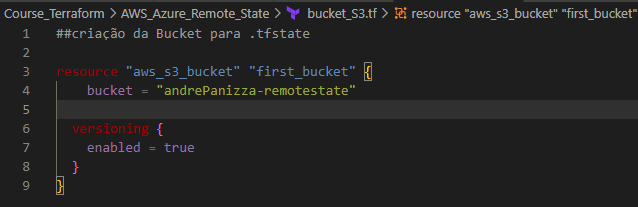
Criamos esses arquivos, a única coisa que temos que alterar e no main.tf onde vamos adicionar mais um Provider que e o da Azure.



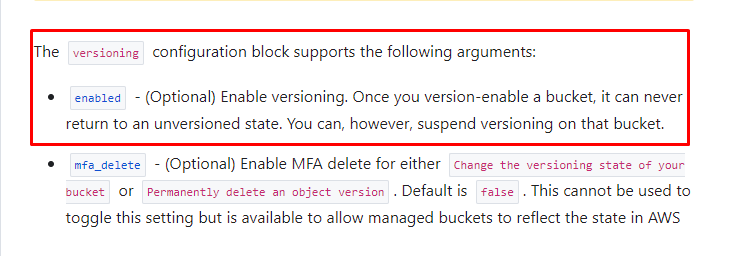
Inserido o Provider, vamos fazer um adicionamento na bucket de acordo com a documentação

Adicionamos na bucket essa informação:

Parte de Versioning, se criamos um arquivo com o mesmo nome na aws, ele não e sobrescrito, e sim o arquivo e colocado em uma outra versão.



-E isto consta na documentação do terraform



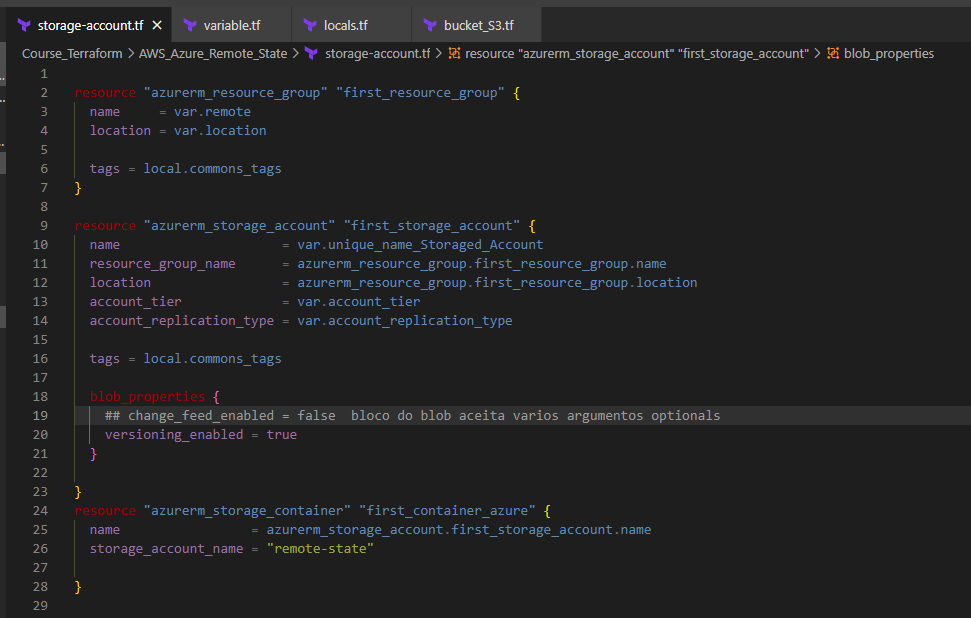
Também iremos fazer pequenas alterações no arquivo storage-account.tf



São essas as alterações:

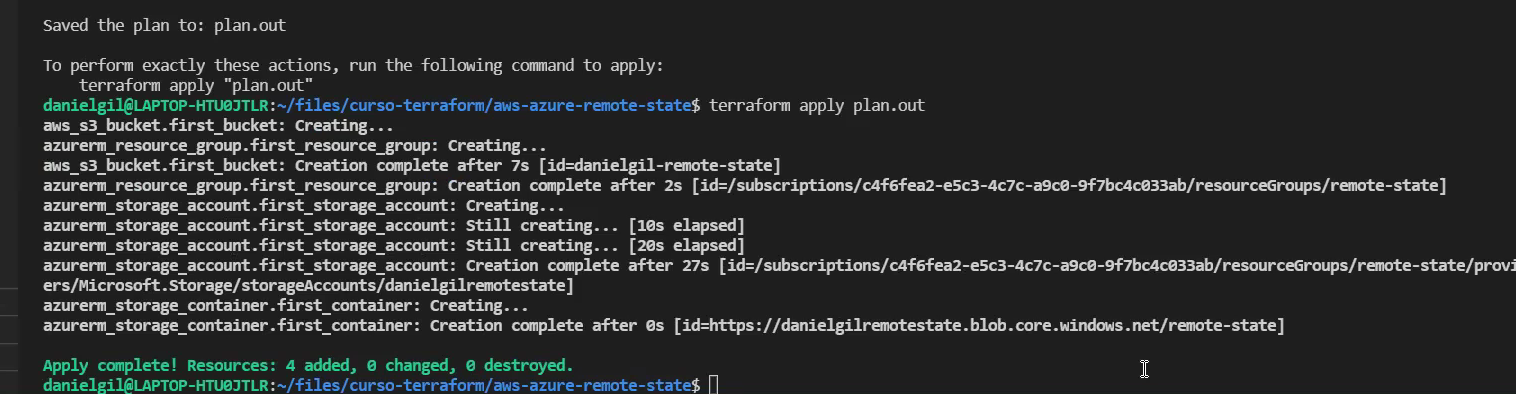
Nota: Na Storage\_Account também poderemos habilitar o versionamento – Versioning

Usaremos o bloco do Blob\_properties e usaremos o Versioning\_enable = true:



Pronto a parte do Storage\_Account está pronta, iremos fazer o Deploy disso tudo agora.

Depois de ter feito o terraform apply, vamos usar esta bucket e essa Storage\_Account, para armazenar nosso state de outras configurações que vamos criar nas próximas aulas.



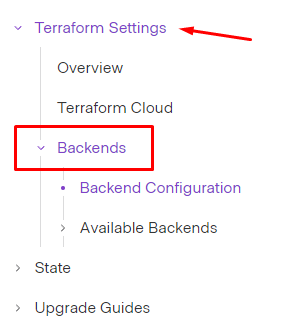
REMOTE STATE

Introdução ao remote State – salvar remotamente o seu State do terraform.

Na documentação do terraform, por default o State do terraform e armazenado localmente, e que nós podemos configurar o remote State, para que podemos configurar de forma remota.

Para que nós possamos configurar o remote State, devemos confugirar o back-end que fica dentro do terraform da nossa configuração.

Indo no terraform settings, onde explica como faz a configuração do bloco do terraform.

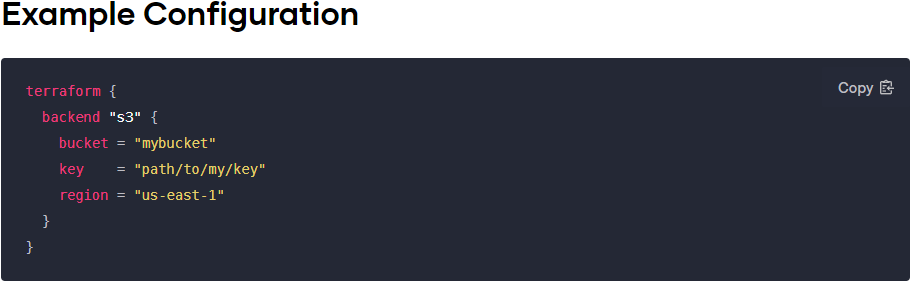


Vamos entrar no back-end, e veremos que ele tem o available Backends:

No BACKENDS CONFIGURATION – ele vai mostrar como funciona a configuração, ele e um next block, dentro do bloco do terraform, o que vamos usar neste curso, e o da Azure quando formos fazer um Deploy de configuração da Azure vamos usar este:



Quando formos fazer o Deploy da estrutura da AWS, vamos usar uma bucket da S3:



E com isso que vamos trabalhar nas nossas próximas aulas:

CRIAÇÃO DE VPC AWS com Remote State

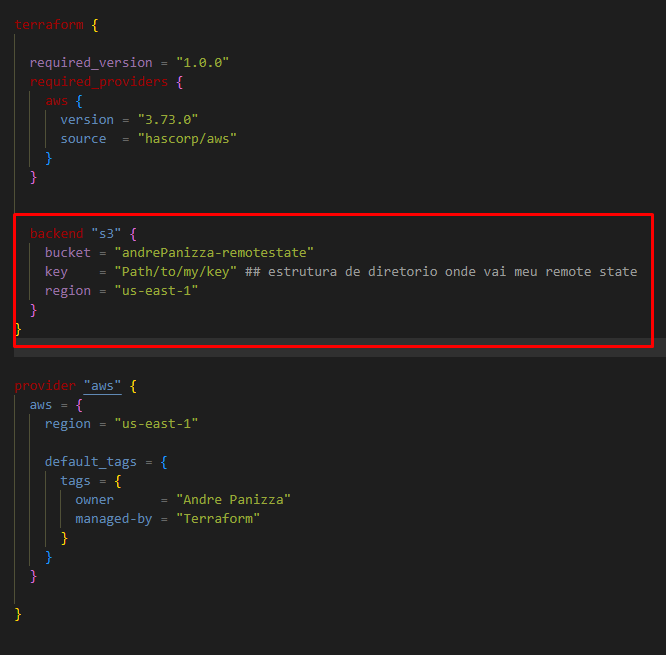
**Estrutura da VPC que vamos construir:**

* **VPC**
* **Subnet**
* **Internet Gateway**
* **Route Table**
* **Security Group**
* **Route Table Association**

Foi criado uma nova pasta com um diretório:



Dentro do main.tf tiramos a parte da configuração da Azure e colocamos somente da AWS e Adicionamos o bloco de configuração do S3 Backend:



**Dessa forma aqui.**

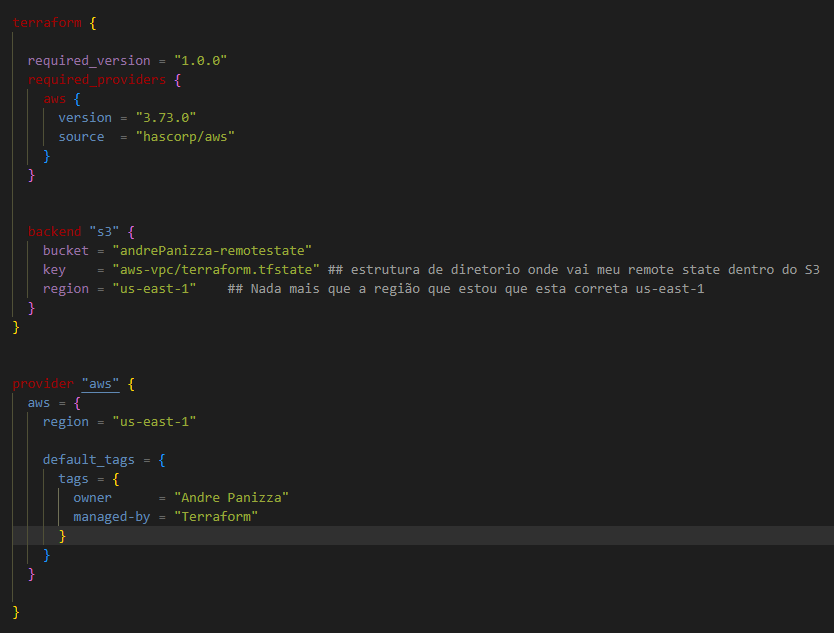
**Teremos algumas coisas que devemos alterar:**

**Nome da nossa bucket recém criada.**

**Key e a pasta que pode ser criada dentro da bucket: aws-vpc/terraform.tfstate**

**Region que por padrão pra mim sempre foi “us-east-1” na AWS**

**Então nosso arquivo ficou assim:**

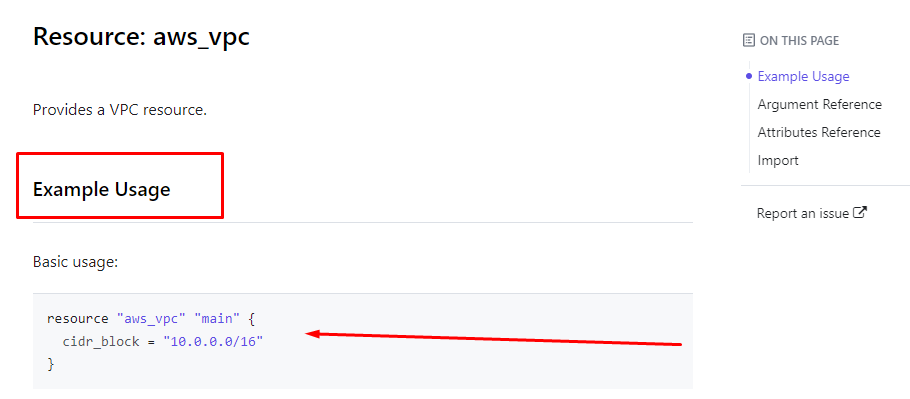


Agora vamos criar um arquivo chamado **Networking.tf**



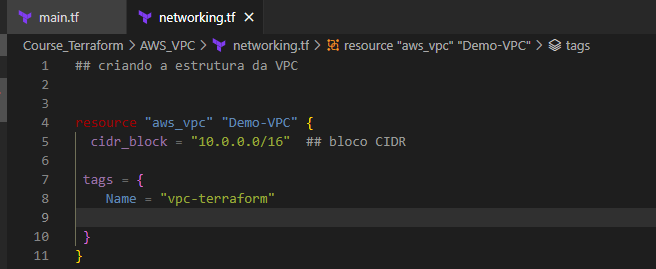
Vamos na documentação do terraform, verificamos onde e que criamos o bloco de VPC:





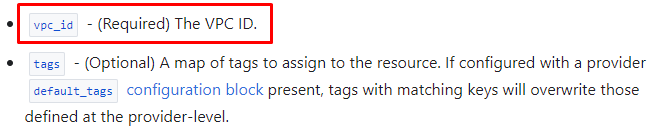
Se verificarmos na parte de Arguments References, todas as opções são opcionais e não a nada de Required. Só vamos adicionar uma Tag com um nome para termos um nome em nossa VPC.

Ficara desta forma:

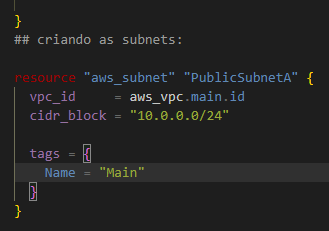


Vamos agora procurar a parte que menciona sobre como criaremos nossas subnets:

Busque por AWS Subnets na documentação e como você pode ver que todos os itens são opcionais menos:



Com este bloco aqui:

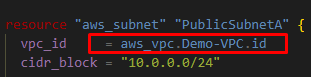


**Note: o vpc\_id estou fazendo uma referência ao recurso que está sendo criado com o bloco de VPC.**

**O nome interno que eu dei pro bloco de VPC e Demo-VPC:**

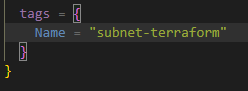


**Então terá que ficar da mesma forma:**



**Seguindo:**

**Vamos alterar a tag, para subnet\_terraform**

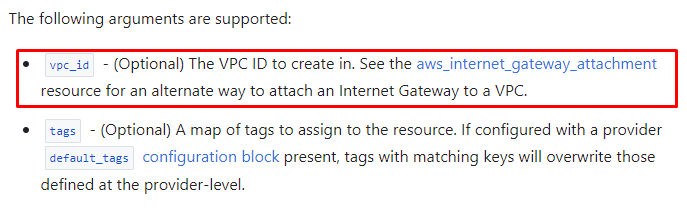


Próximo Item que vamos precisar:

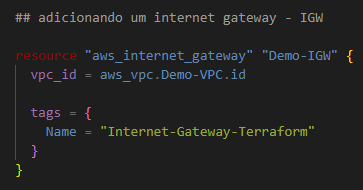
**Internet Gateway**

Acessando a documentação do terraform, na seção Resources\_Internet\_Gateway

Temos o bloco do internet\_gateway:



Onde todos são opcionais, enfim vamos colocar apenas o bloco do vpc\_id que e comum ser colocado.



Pronto, recurso foi adicionado.

Agora vamos para o próximo passo e criar a nossas 02 Router tables: PublicRouteTable and PrivateRouterTable

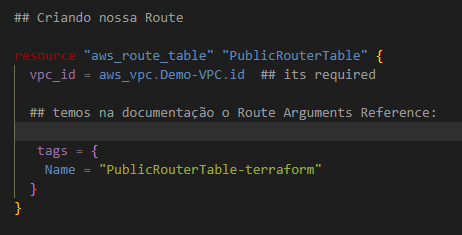
Criando a Route Table:

Daremos inicio a nossa criação da Route Table:

A estrutura que vamos criar e esta que será apontada abaixo, Note que temos que criar uma PublicRouterTable e uma PrivateRouterTable:



Ficou dessa forma a estrutura de PublicRouterTable:



Agora vamos adicionar o Route que consta na documentação do Terraform, pois com ele que iremos apontar a questão do bloco nas nossas Router:

Dentro do Bloco do Router podemos mencionar o CIDR\_Block:



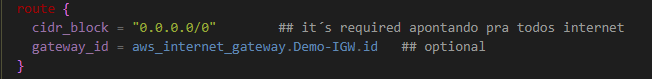
E o gateway\_ID:



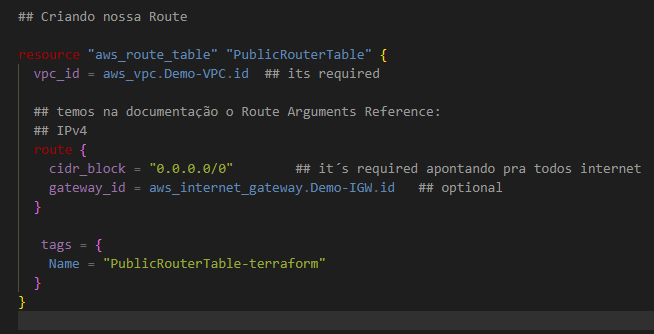
Importante quando criamos na AWS fazemos dessa forma:

Create Router Table -> required: nome da Route, Add new tags (optional)

Nota: que por default, a gente já possui na AWS a table Default.



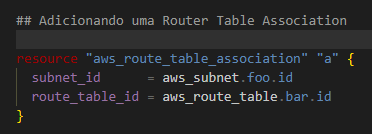
Devemos criar dessa forma acima. (above)



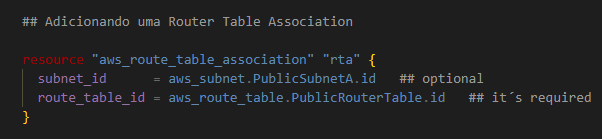
O resultado final devera ser dessa forma acima:

Criando RoterTableAssociation:

Vamos criar uma routertableAssociation:



Vamos substituir o nome da nossa “aws\_route\_table\_association” para rta.



Nome da minha subnet e **PublicSubnetA**

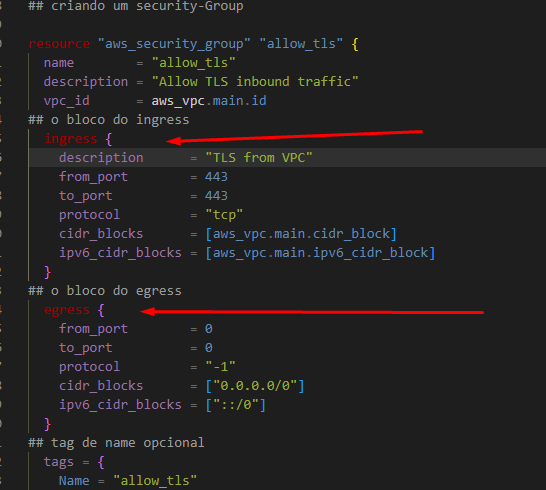
Nome da minha RouterTable e **PublicRouterTable**

Criando um Security-Group:

Temos na Documentação o resource sobre o Security-Group:



Vamos adicionar ao Security-Group um Name, a VPC-id, um bloco de ingress e um de egress:



Primeiramente, vamos dar um nome a nosso SG:

Por padrão na AWS ele vem com o nome de **Lauch-Wizard-1**

Nome: Lauch-Wizard-1-terraform

Descrição (Description): Não precisaremos de Description

Vpc\_id: aws\_vpc.Demo-VPC.id

**Ingress:**

Description: Não vou precisar de Description

From port: 22

To port: 22

Protocol: “tcp” mantém

CIDR\_Blocks: [0.0.0.0/0] abrindo pra todos

Não necessito de Ipv6

**Egress:**

From port: 0

To port: 0

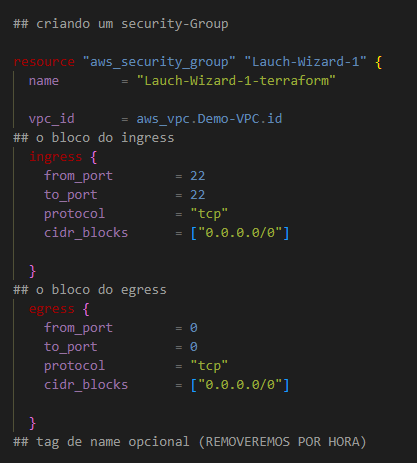
Protocol: “tcp” mantém

**Não precisaremos da última opção**

**Tags:**

Removeremos as tags

Ficando dessa forma



Iremos criar um Output.tf

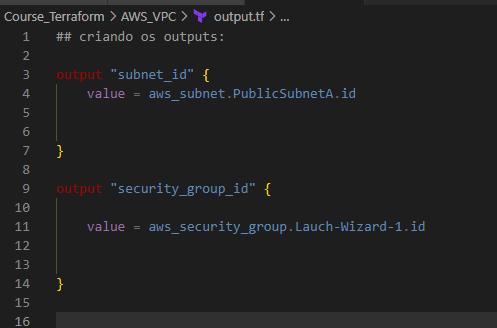
Iremos criar o arquivo output, pois temos coisas que precisaremos utilizar adiante:

Iremos criar 02 blocos de output para armazenarmos os Ids da vpc e do security-group que basicamente e os nomes deles para usarmos depois:

Criando o arquivo:

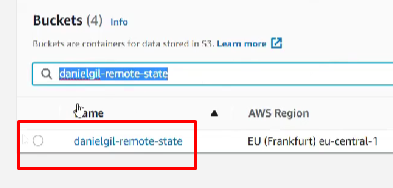


Adicionando os blocos e colocando as Ids da minha estrutura da VPC



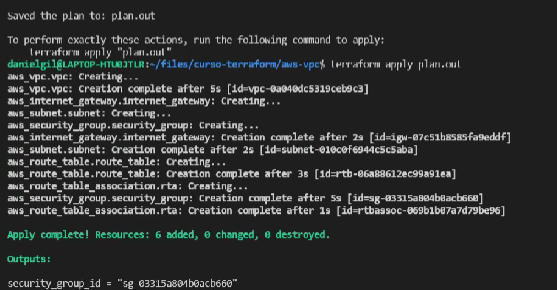
# Nota importante:

Existe uma bucket com o meu nome, que está dentro da AWS, essa bucket ela vai armazenar o arquivo **terraform.tfstate**. assim que eu fizer o Deploy desssa aplicação:

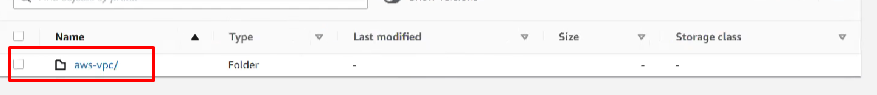


Assim que eu fizer o Deploy da aplicação com terraform apply plan.out

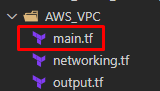
A estrutura vai ser criada conforme na imagem do console:

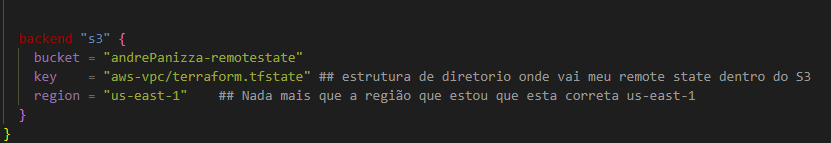


# Note que dentro do Meu S3 agora consta o arquivo terraform.tfstate:



Da forma que configuramos a nossa Bucket:





Como podemos ver o diretório de pastas foi criado, aws-vpc/terraform dentro da Bucket

AndrePanizza-remotestate – O nome da minha Bucket

Está tudo criado dentro da AWS:

Criação de Vnet na Azure com Remote State