



# Terraform

An introduction of Infrastructure as a code  
Atlantis, HCL, Terragrunt and others





# Terraform

## Conceito e Funcionamento





## IaC - Infrastructure as a Code

A infraestrutura como código (IaC) configura e gerencia a infraestrutura por meio de um modelo descritivo. Trata-se de tratar a configuração e o provisionamento de sua infraestrutura da mesma maneira que você trata o código-fonte do aplicativo.

Os módulos de configuração são normalmente armazenados em sistemas de controle de versão em formatos de código bem documentados, proporcionando maior precisão, reduzindo erros e aumentando a velocidade e a consistência.

A IaC é uma das práticas de DevOps mais importantes usadas com entrega contínua. Os benefícios que levam muitas empresas a migrar para o IAC são infraestrutura imutável, aumento na velocidade de entrega, escalabilidade, economia de custos e mitigação de riscos.





Terraform

Ansible

AWS CDK

Chef

Puppet

SaltStack

(R)?ex

Vagrant

Crossplane

Juju

CFEngine

Pallet

NixOS

Dagger

Packer

“Alternativas” ao Terraform:

Attune

Pulumi

Alternativas para ambientes específicos:

AWS CloudFormation

Azure Resource Manager

Google Cloud Deployment Manager

Libs:

Boto (python)

Fog (Ruby)



Terraform é uma ferramenta de IaC open-source criada pela HashiCorp usando uma linguagem de configuração declarativa conhecida como HashiCorp Configuration Language (HCL) e JSON.

Escrito em Go

Declarativo

<https://github.com/hashicorp/terraform>

HashiCorp  
**Terraform**



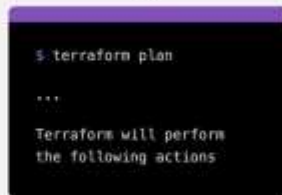
## Write

Define infrastructure in configuration files



## Plan

Review the changes  
Terraform will make to  
your infrastructure



## Apply

Terraform provisions  
your infrastructure and  
updates the state file.



O Terraform possui quatro comandos principais:


\$ terraform init

\$ terraform plan

\$ terraform apply

\$ terraform destroy

Main commands:



init	Prepare your working directory for other commands
validate	Check whether the configuration is valid
plan	Show changes required by the current configuration
apply	Create or update infrastructure
destroy	Destroy previously-created infrastructure

All other commands:

console	Try Terraform expressions at an interactive command prompt
fmt	Reformat your configuration in the standard style
force-unlock	Release a stuck lock on the current workspace
get	Install or upgrade remote Terraform modules
graph	Generate a Graphviz graph of the steps in an operation
import	Associate existing infrastructure with a Terraform resource
login	Obtain and save credentials for a remote host
logout	Remove locally-stored credentials for a remote host
output	Show output values from your root module
providers	Show the providers required for this configuration
refresh	Update the state to match remote systems
show	Show the current state or a saved plan
state	Advanced state management
taint	Mark a resource instance as not fully functional
test	Experimental support for module integration testing
untaint	Remove the 'tainted' state from a resource instance
version	Show the current Terraform version
workspace	Workspace management

## Write

Define infrastructure in configuration files

### TERRAFORM PROJECT



Terraform Configuration



Terraform State File

## Plan

Review the changes  
Terraform will make to  
your infrastructure

```
$ terraform plan
```

```
...
```

```
Terraform will perform  
the following actions
```

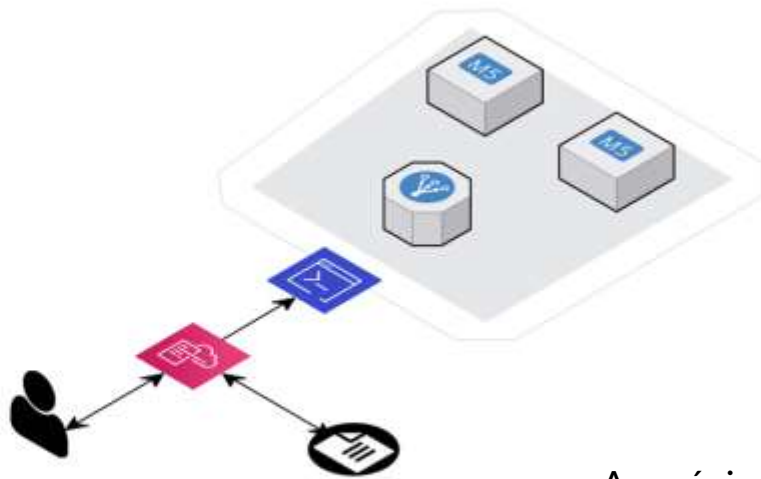
## Apply

Terraform provisions  
your infrastructure and  
updates the state file.







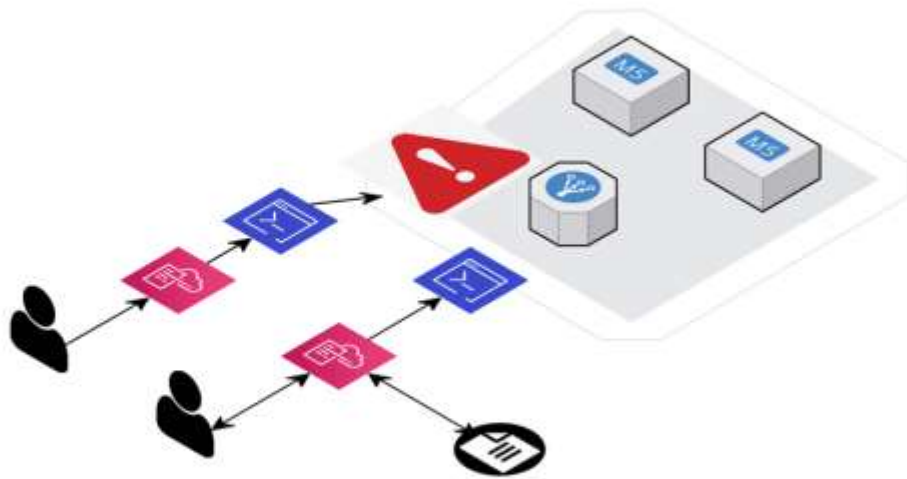


Escrevo minha Infraestrutura

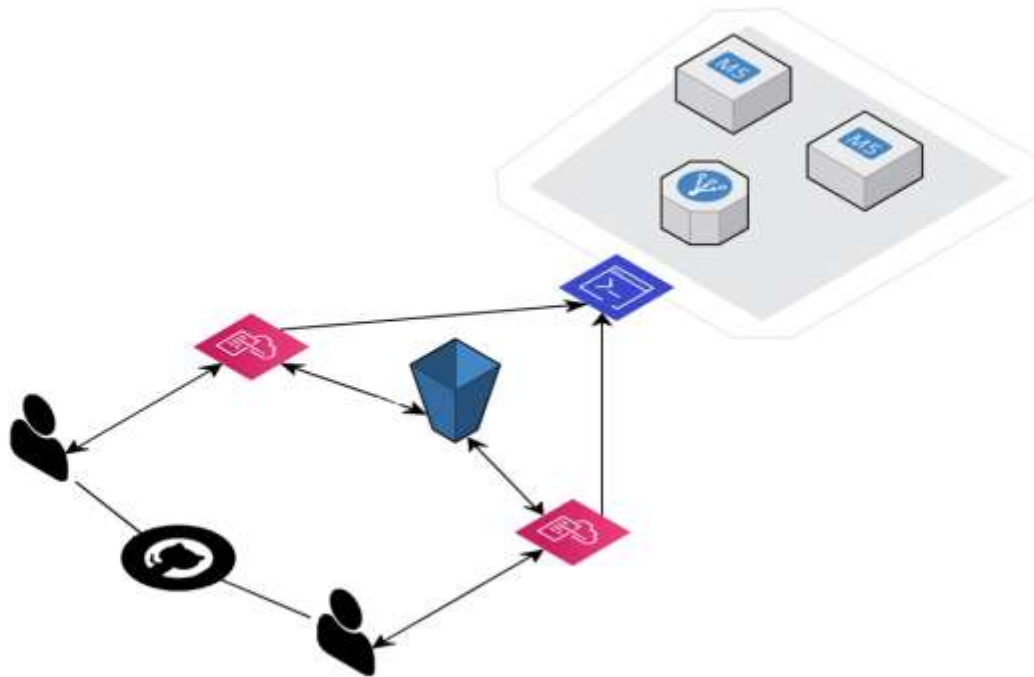
Terraform executa e cria

Terraform salva o estado da criação num json.tfstate

As próximas alterações o plan irá consultar o estado.



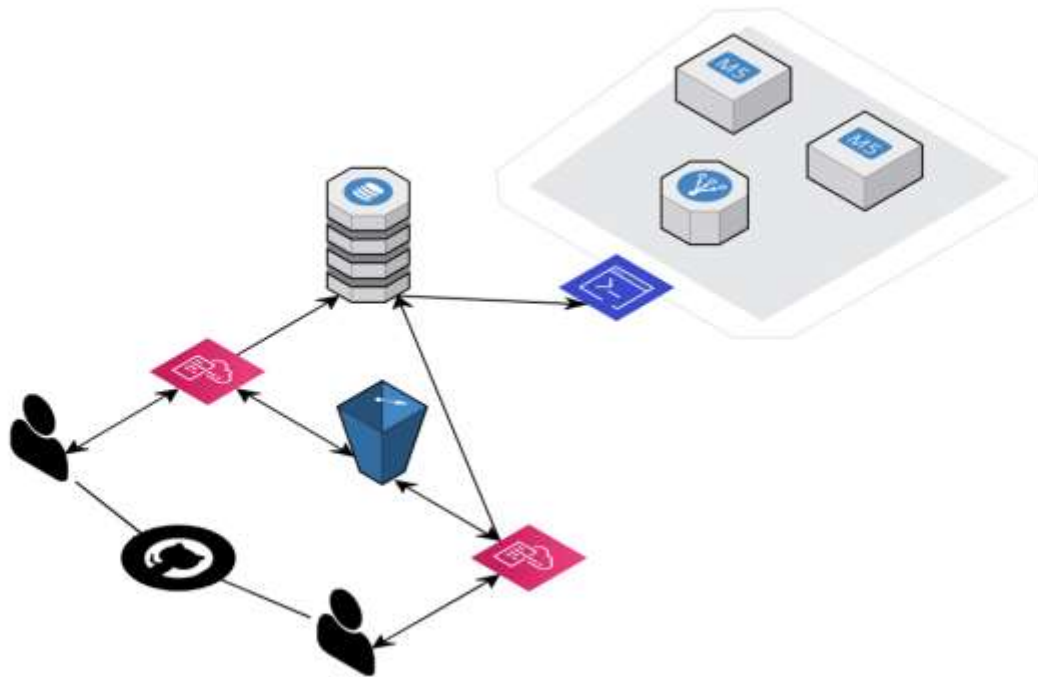
O state é local



Pode ser armazenado em qualquer lugar que aceite um json.

API rest, AWS, GCP, Azure, Manta, Artifactory, etcd, Consul, Postgres, ....

Com o state remoto, outros conseguem alterar a mesma infraestrutura.



E se mais de 1 pessoa  
tentar fazer o apply na  
mesma infraestrutura?



## Problemas resolvidos:

1

### Infraestrutura como código

regra n1 - 12factor:

*Uma base de código com rastreamento  
utilizando controle de revisão*

2

### Estado remoto

Um time consegue trabalhar  
simultaneamente no desenvolvimento da  
Infra.

3

### Lock no Apply

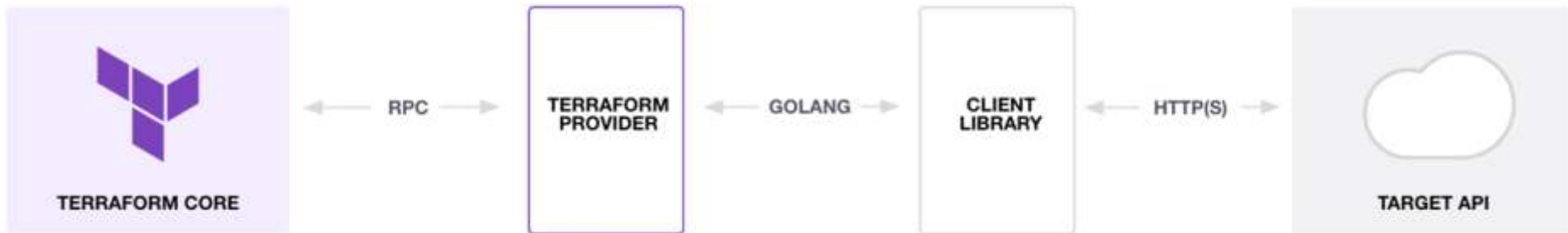
Não irá ocorrer conflitos com execução em  
paralelo.



# Terraform

Linguagem Terraform





<https://registry.terraform.io/browse/providers> - + 2000 providers no registry público (Oficial, verificado e community).





## Composição básica

backend



Define onde o state  
será armazenado

provider



Informa os plugins a serem  
usados para interagir com  
provedores de nuvem,  
provedores de SaaS e outras  
APIs

tf files



Declaração dos recursos a  
serem criados

variables files



Declara os valores das  
variáveis usadas no tf  
file.



## Terraform's Configuration Language

A linguagem do Terraform é declarativa, descrevendo um objetivo pretendido em vez das etapas para atingir esse objetivo.

*No paradigma de programação declarativo, declaro o que eu quero como resultado final ao invés de escrever como eu quero (o passo-a-passo para obtê-lo).*

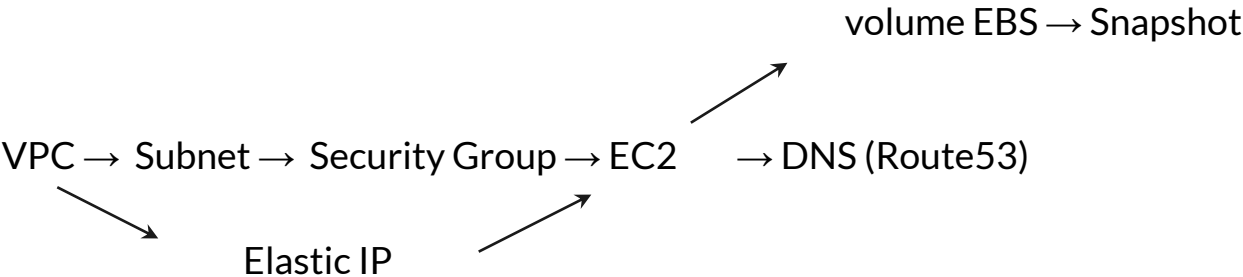
- Declarativo: Erlang, Haskell, SQL - **abstrato** - Linguagem Funcional
- Imperativo: C, C++, Java - **concreto** - Linguagem Procedural
- Ambos: Python, C#, JavaScript - Linguagem Orientada a Objeto



# Terraform's Configuration Language

A ordenação dos blocos e os arquivos em que são organizados geralmente não são significativos;

O Terraform considera apenas relacionamentos implícitos e explícitos entre recursos ao determinar a ordem de operações.





## Terraform's Configuration Language

Os blocos são contêineres para outros conteúdos e geralmente representam a configuração de algum tipo de objeto, como um recurso (resource).

Os blocos têm um **type**, podem ter zero ou mais **labels** e um corpo que contém argumentos e blocos aninhados.

A maioria dos recursos do Terraform são controlados por blocos de nível superior em um arquivo de configuração.

Os argumentos atribuem um valor a um nome.

As expressões representam um valor, literalmente ou referenciando e combinando outros valores. Eles aparecem como valores para argumentos ou dentro de outras expressões.



## Terraform's Configuration Language

```
resource "aws_vpc" "main" {  
  cidr_block = var.base_cidr_block  
}  
  
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {  
  # Block body  
  <IDENTIFIER> = <EXPRESSION> # Argument  
}
```

```
resource "aws_route53_record" "www" {  
  zone_id = Z001753YB9D130HADG3  
  name     = "www.example.com"  
  type     = "A"  
  ttl      = "300"  
  records  = example.alb.us-east-1.amazonaws.com  
}
```



## Terraform's Configuration Language

```
terraform {  
  backend "s3" {  
    bucket = "iaac-terraform"  
    key = "tfstates/project-iaac/terraform.tfstate"  
    region = "sa-east-1"  
    profile = "default"  
    dynamodb_table = "terraform_state_table"  
    encrypt = "true"  
  }  
  
  provider "aws" {  
    credential= "~/.aws/credentials"  
    region= "sa-east-1"  
  }  
}
```

```
resource "aws_route53_record" "www" {  
  zone_id = Z001753YB9D130HADG3  
  name     = "www.example.com"  
  type     = "A"  
  ttl      = "300"  
  records  = example.alb.us-east-1.amazonaws.com  
}
```



## Terraform's Configuration Language

route53.tf

```
resource "aws_route53_record" "www" {  
  zone_id = Z001753YB9D130HADG3  
  name     = "www.example.com"  
  type     = "A"  
  ttl      = "300"  
  records  = example.alb.us-east-1.amazonaws.com  
}
```



## Terraform's Configuration Language

route53.tf

```
resource "aws_route53_record" "www" {  
  zone_id = var.zone_id  
  name     = var.name  
  type     = "A"  
  ttl      = "300"  
  records  = var.records  
}
```

variables.tf

```
variables "name" {  
  default = "www.example.com"  
}  
  
variables "zone_id" {  
  default = "Z001753YB9D130HADG3"  
}  
  
variables "records" {  
  default = "example.alb.us-east-1.amazonaws.com"  
}
```





## Terraform's Configuration Language

route53.tf

```
resource "aws_route53_record" "www" {  
  zone_id = var.zone_id  
  name     = var.name  
  type     = "A"  
  ttl      = "300"  
  records  = var.records  
}
```

variables.tf

```
variables "name" {  
}  
  
variables "zone_id" {  
}  
  
variables "records" {  
}
```

variables.tfvars

```
name = "www.example.com"  
zone_id = "Z001753YB9D130HADG3"  
records = "example.alb.us-east-1.amazonaws.com"
```



## Modules

```
module "instance" {  
    source = "git@github.com:cloudposse/ec2-instance.git"  
  
    ssh_key_pair      = var.ssh_key_pair  
    instance_type     = var.instance_type  
    vpc_id            = var.vpc_id  
    security_groups   = var.security_groups  
    subnet            = var.subnet  
    name              = "ec2"  
    namespace         = "eg"  
    stage             = "dev"  
}
```

Com o módulo, o bloco com o resource já está escrito e versionado no git.

Apenas será necessário escrever as variáveis e apontar para o terraform, onde está o source do módulo.

Pode ser local (outro diretório), registry do terraform, git, bucket S3, ...



## Modules (usar ou não)?

### DRY - Don't Repeat Yourself

```
resource "aws_route53_record" "www" {  
  zone_id = var.zone_id  
  name     = var.name  
  type     = "A"  
  ttl      = "300"  
  records  = var.records  
}  
  
variables "name" {  
  default = "www.example.com"  
}  
  
variables "zone_id" {  
  default = "Z001753YB9D130HADG3"  
}  
  
variables "records" {  
  default = "example.alb.us-east-1.amazonaws.com"  
}
```

```
module "instance" {  
  source = "git@github.com:Es11h/tfmodules/r53.git"  
  
  name      = "www.example.com"  
  zone_id   = "Z001753YB9D130HADG3"  
  records   = "example.alb.us-east-1.amazonaws.com"  
}
```



## HCL - HashCorp Language

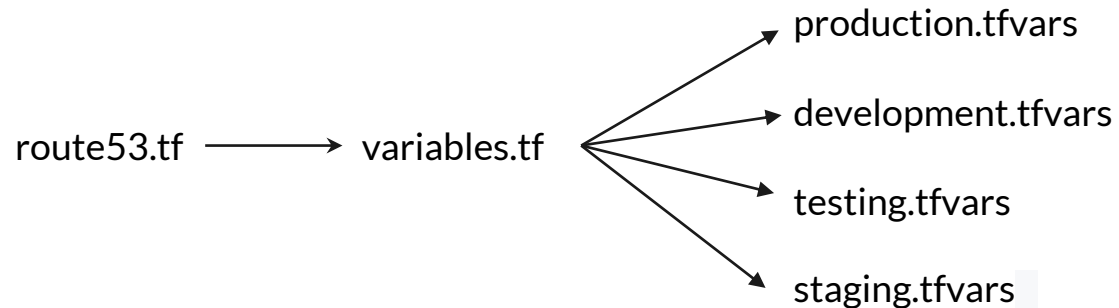
A sintaxe de baixo nível da linguagem Terraform é definida como HCL

A linguagem consiste em  
3 sub-linguagens integradas:  
estrutural  
expressão  
template

```
data external test_policy_members {  
  query = {  
    for role, members in local.cloudrun_iam_bindings["default"]:  
      role => length(lookup(members, "members", []))  
  }  
  
  program = ["python", "${path.module}/test_policy_members.py"]  
}  
  
output test_policy_members {  
  value = data.external.test_policy_members.result  
}
```



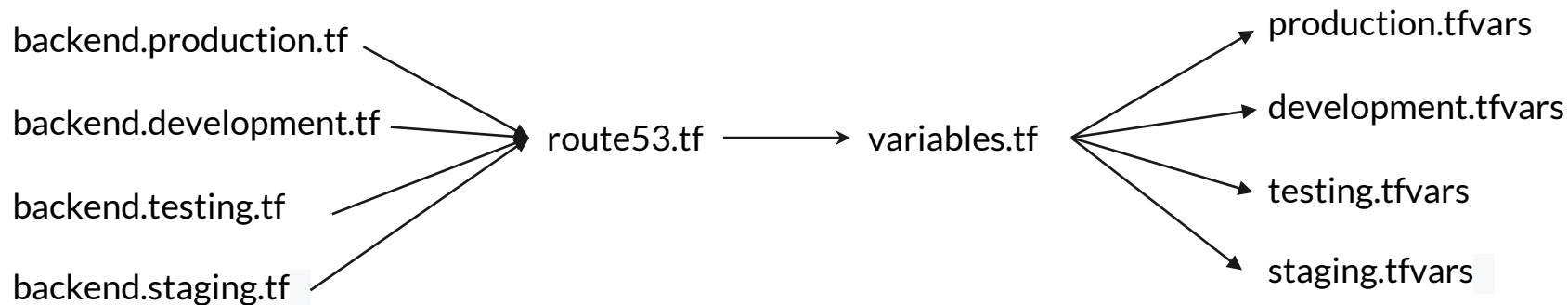
## Terraform's Configuration Language



```
terraform init --var-file=production.tfvars  
terraform plan --var-file=production.tfvars  
terraform apply --var-file=production.tfvars
```

## Terraform's Configuration Language

E se para cada ambiente eu tiver um backend (s3 e dynamodb) diferentes?



```
terraform init --backend-config=backend.production.tfvars --var-file=dev.tfvars  
terraform plan --var-file=production.tfvars  
terraform apply --var-file=production.tfvars
```



## Workspace

Na execução do terraform, para cada backend, será criado dados persistentes. Eles são armazenados e consultados.

Como, numa segunda execução (development e depois production) os dados não são sobrescritos?

Estes dados persistentes pertencem a um ***environment*** (chamado de ***workspace***)

Para production, por exemplo, podemos criar o workspace:

```
terraform workspace new production
terraform workspace select production
```

Para posteriormente executar o init/plan/apply:

```
terraform init --backend-config=backend.production.tfvars --var-file=dev.tfvars
terraform plan --var-file=production.tfvars
terraform apply --var-file=production.tfvars
```



## Terraform's Configuration Language

Poderia criar uma pasta para cada ambiente? E ter um código terraform para cada um?

Regra número 10 - 12factor:

Mantenha o desenvolvimento, teste, produção o mais semelhante possível





## Problemas resolvidos:

1

### Infraestrutura como código

regra n1 - 12factor:

*Uma base de código com rastreamento  
utilizando controle de revisão*

2

### Estado remoto

Um time consegue trabalhar  
simultaneamente no desenvolvimento da  
Infra.

3

### Lock no Apply

Não irá ocorrer conflitos com execução em  
paralelo.

4

### Sem reinventar a roda e Padronização

Usar módulos e não gastar tempo  
escrevendo um resource que já foi feito em  
outros momentos.



# Terragrunt





## DRY - Don't Repeat Yourself

O Terragrunt é um wrapper que fornece ferramentas extras para manter suas configurações DRY, trabalhando com vários módulos do Terraform e gerenciando o estado remoto.

Configuração fica num arquivo ***terragrunt.hcl***, junto com os arquivos do terraform



## DRY

```
terraform workspace new production
terraform workspace select production
terraform init --backend-config=backend.production.tfvars --var-file=dev.tfvars
terraform plan --var-file=production.tfvars
terraform apply --var-file=production.tfvars
```



+ terragrunt.hcl



```
terragrunt init
terragrunt plan
terragrunt apply
```



## Problemas resolvidos:

1

### Infraestrutura como código

regra n1 - 12factor:

*Uma base de código com rastreamento  
utilizando controle de revisão*

2

### Estado remoto

Um time consegue trabalhar  
simultaneamente no desenvolvimento da  
Infra.

3

### Lock no Apply

Não irá ocorrer conflitos com  
execução em paralelo.

5

### DRY and KISS

Não repete códigos

Mantém simples, reduzindo complexidade  
evita erros na execução



Quanto maior o time e os ambientes, mais controle precisa ser implementado e aumenta a complexidade para manutenção.

Como garantir que o código realmente condiz com a infra existente?

Como garantir que o código passou por um review?

Como justificar as mudanças aplicadas na infra existente? Qual motivo?  
Quem aprovou?

# Atlantis





Atlantis é um aplicativo para automatizar o Terraform por meio de pull requests.

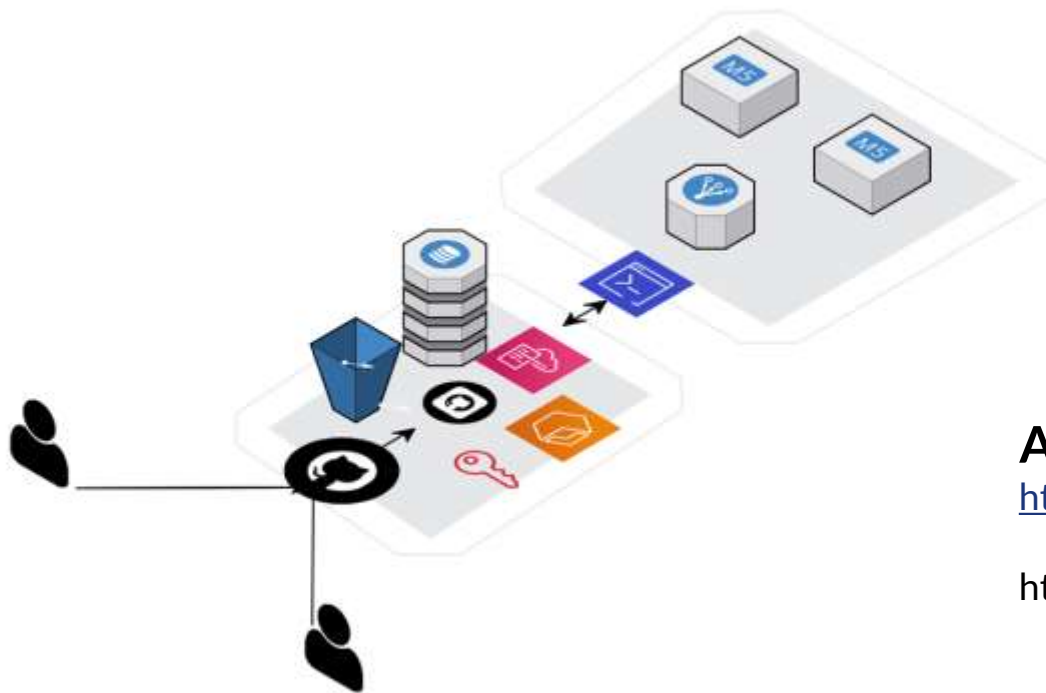
Ele é implantado como um aplicativo independente em sua infraestrutura.

Nenhum terceiro tem acesso às suas credenciais.

O Atlantis ouve os webhooks do GitHub, GitLab ou Bitbucket sobre solicitações de pull do Terraform. Em seguida, ele executa o **terraform plan** e comenta com a saída de volta na solicitação pull.

Quando você quiser aplicar, comente **atlantis apply** na solicitação pull e o Atlantis executará terraform apply e comentará de volta com a saída.





**Atlantis**

<https://runatlantis.io>

<https://github.com/runatlantis/atlantis>



# atlantis.yaml

version: 3

projects:

- dir: .

workspace: development

autoplan:

when\_modified: ["\*"]

enabled: false

terraform\_version: v1.1.4

```
- dir: .  
  workspace: production  
  autoplan:  
    when_modified: ["*"]  
    enabled: false  
  terraform_version: v1.1.4
```



## Links e referências

Terraform em 15 minutos [EN] [https://youtu.be/ISk1aj\\_GBDE](https://youtu.be/ISk1aj_GBDE)

Curso Terraform - Automatize sua Infra em Cloud AWS [EN] **2h30m** [https://youtu.be/SLB\\_c\\_ayRMo](https://youtu.be/SLB_c_ayRMo)

Curso completo para certificação - HashiCorp Terraform Associate Certification [EN] **13h10m** <https://youtu.be/V4waklkBC38>

Terraform: <https://www.terraform.io>

Terragrunt: <https://terragrunt.gruntwork.io>

Atlantis: <https://runatlantis.io>



# Obrigado!

