Terraform

Uma breve breve introdução

Leandro Silva
https://github.com/leandrosilva

Com um oferecimento...

Todos os meses geramos mais de **15 milhões** de cartazes em PDFs de alta qualidade.

E isso não é tudo.

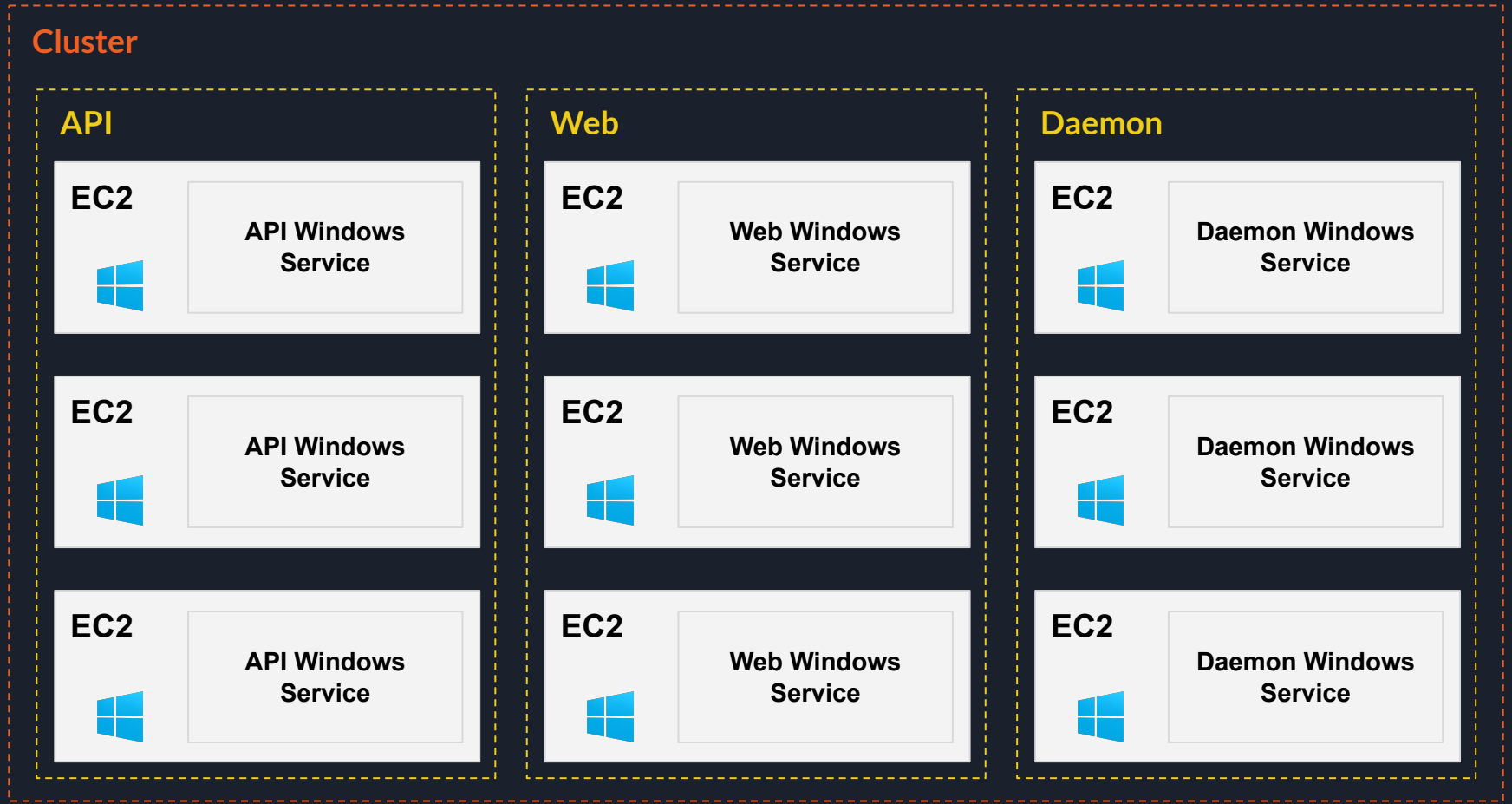
www.pricefy.com.br

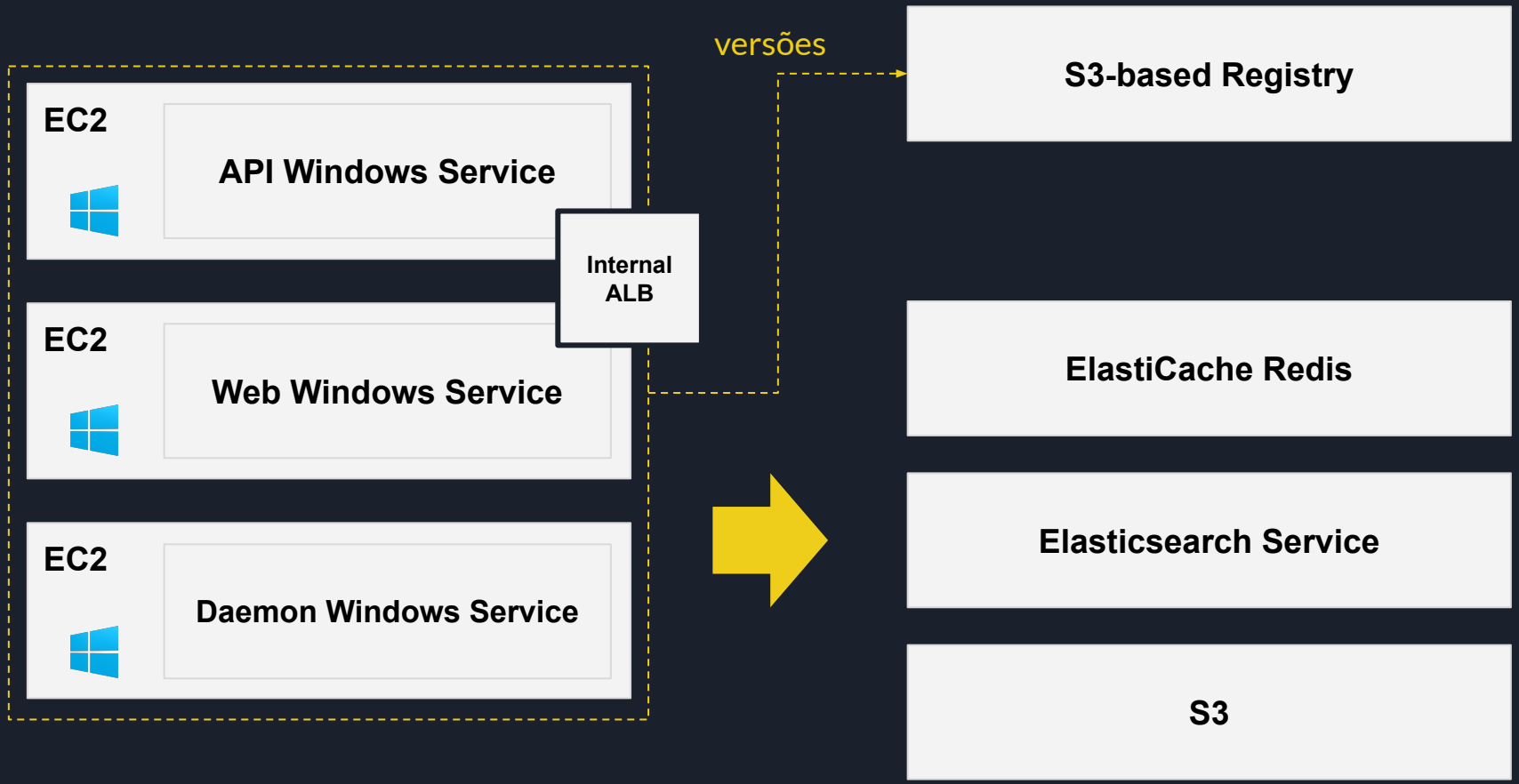
Cenário hipotético

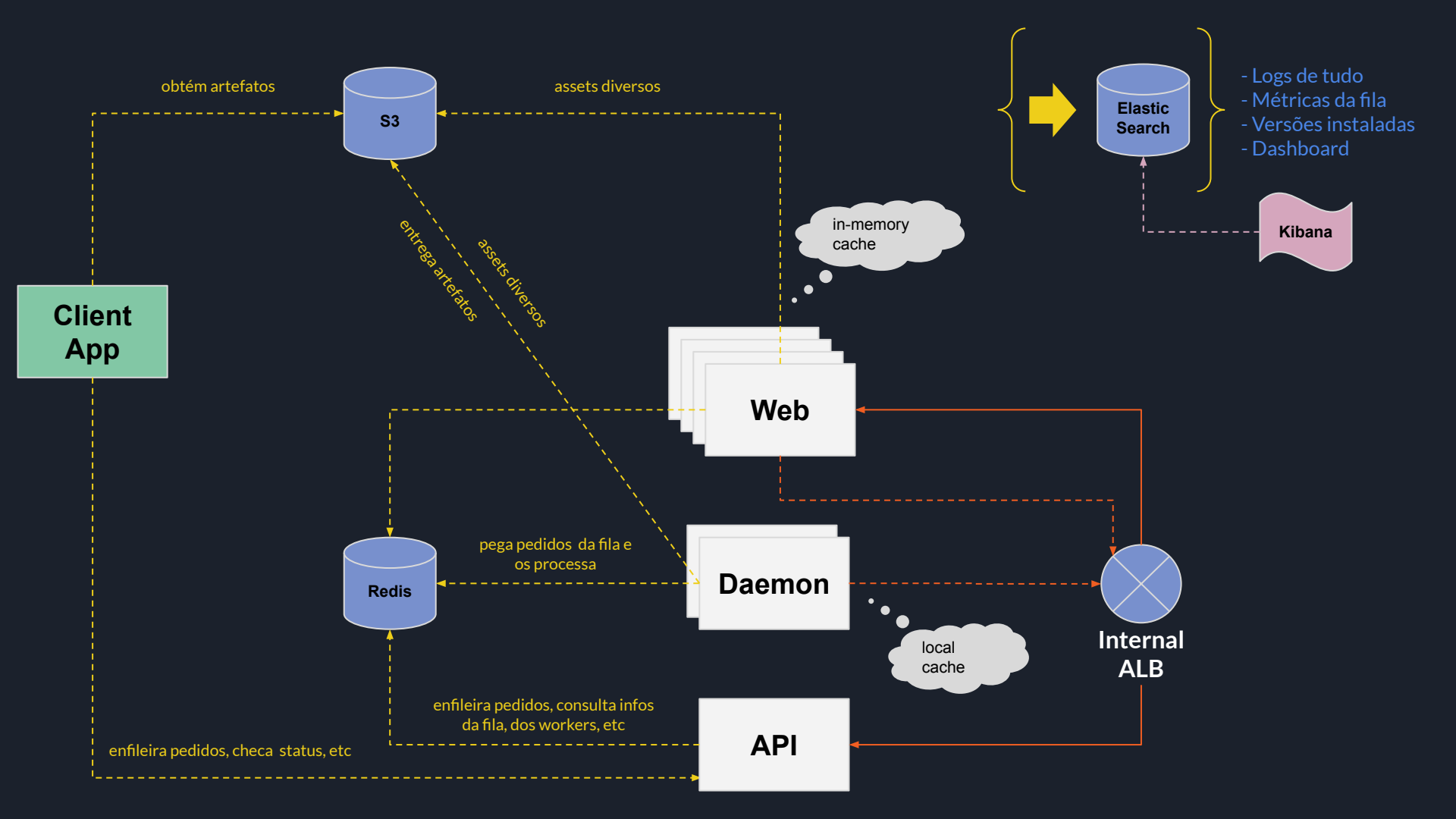
Cenário hipotético

Nosso estudo de caso propõe:

- Uma solução de software definida por um cluster com três nós: API, Web e Daemon;
- Cada nó está encapsulado em um serviço Windows rodando em uma EC2 dedicada;
- A qualquer momento, pode haver mais de uma instância de qualquer um dos nó rodando;
- Estes nós fazem uso de diversos serviços AWS, por exemplo: S3, Redis e Elasticsearch;
- Suas versões são publicadas no S3 e instaladas em suas novas instâncias, de forma automatizada, via script PowerShell, no momento de sua criação;
- Existe, portanto, uma imagem de Windows base para cada um destes serviços, que é complementada com a versão determinada do software no catálogo de versões;
- A performance dos nós pode ser monitorada individualmente e notificada à operação;
- O auto scaling também acontece por nó, de acordo com tráfego de rede, CPU, fila de processamento, período do dia e da semana, etc.


Cluster
API
EC2
API Windows Service
EC2
API Windows Service
EC2
API Windows Service
Web
EC2
Web Windows Service
EC2
Web Windows Service
EC2
Web Windows Service
Daemon
EC2
Daemon Windows Service
EC2
Daemon Windows Service
EC2
Daemon Windows Service

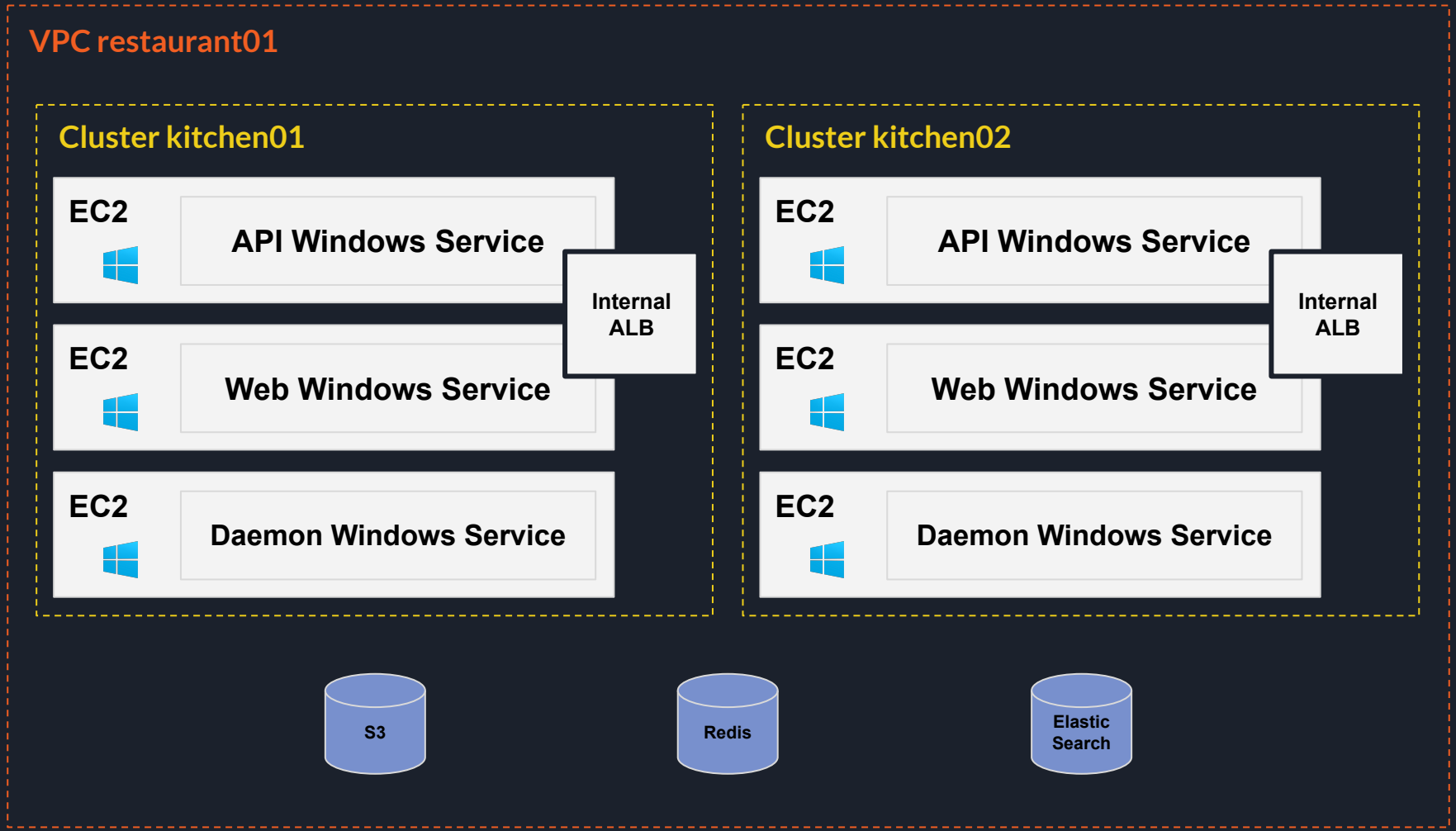

versões
S3-based Registry
EC2
API Windows Service
Internal ALB
EC2
Web Windows Service
ElastiCache Redis
EC2
Daemon Windows Service
Elasticsearch Service
S3


obtém artefatos
S3
assets diversos
Elastic Search
- Logs de tudo
- Métricas da fila
- Versões instaladas
- Dashboard
Kibana
in-memory cache
entrega artefatos
assets diversos
Client App
Web
Redis
pega pedidos da fila e os processa
Daemon
local cache
Internal ALB
enfileira pedidos, consulta infos da fila, dos workers, etc
API
enfileira pedidos, checa status, etc

Cenário hipotético

Outra proposição do nosso estudo de caso é que:

- A qualquer momento, pode haver mais de um cluster com a exata arquitetura vista até aqui, rodando em paralelo, sem que um tenha notícia do outro;
- Pode haver também o caso de um ou mais deles estarem em uma versão de arquitetura diferente, por uma questão experimental, por segregação de carga de trabalho, ou por uma evolução da solução;
- Portanto, eles podem ou não compartilhar as mesmas instâncias dos serviços da AWS;
- Dito isso, é evidente assumir também que a versão de software de cada cluster independe da versão do outro, o que é determinado pelo catálogo de versões;
- O catálogo de versões, portanto, mantém o registro de versão de cada cluster.


VPC restaurant01
Cluster kitchen01
EC2
API Windows Service
EC2
Web Windows Service
EC2
Daemon Windows Service
Internal ALB
Cluster kitchen02
EC2
API Windows Service
EC2
Web Windows Service
EC2
Daemon Windows Service
Internal ALB
S3
Redis
Elastic Search

Cenário hipotético

Prós:

- Cada nó encapsulado em um serviço Windows rodando em uma EC2 dedicada;
- Versões publicadas no S3 e instaladas nas instâncias EC2 via script PowerShell na criação;
- Performance dos nós podendo ser individualmente monitorada e escalada;
- Auto scaling individualizado por nó, de acordo com tráfego de rede, CPU, fila, etc;
- Possibilidade de ter múltiplos clusters da solução em versões diferentes.

Contras:

- É difícil de gerenciar e dar manutenção em arquiteturas diferentes, com clusters diferentes, com versões de software diferentes;
- É impossível se lembrar de todas as configurações e conseguir reproduzir rapidamente.

Sim, eu disse isso mesmo. Tente recriar um cluster desses, na mão.

Entra o Terraform.

Uma breve introdução ao Terraform

Problema

Reproduzir comportamento de software rodando em infraestruturas diferentes parece um problema trivial, mas é f%d@ sempre:

- Há pouca rastreabilidade de mudanças de infraestrutura;
- Máquinas rodando há muito tempo sofrem mudanças que ninguém se lembra depois;
- A infraestrutura vai se degradando com o tempo, mas ninguém quer refazer;
- Quanto mais componentes, menor a coragem de refazer uma infra completa;

E criar ambiente de teste & troubleshooting?

- Dá um puta trabalho reproduzir a infra de produção;
- Custa caro em todos os sentidos;
- Quando você finalmente cria, não quer destruir - por quê? `#loop`

Problema

Você mesmo nunca fez isso, a gente sabe, mas:

- Tem sempre aquele arquivinho que alguém modificou em uma instância e não se lembra mais nem que mudou, quando mudou, nem o que mudou, só sabe que agora o serviço está funcionando corretamente e pelo amor de tudo que é mais sagrado, não mexa nele!

OK, vamos exercitar só mais um pouco esse cenário:

- Como reproduzir esse comportamento em uma nova instância?
- Aliás, esta instância em questão, agora modificada, ninguém sabe como, o quão compatível estará com a futuras versões do software?
- Como testar se estará, se não será possível criar uma instância de testes igual a esta? (Tudo bem, você sempre pode clonar a EC2, mas isso só vai empurrar o problema para frente, como uma bola de neve.)

Quanto tempo você leva para recriar totalmente a stack de uma aplicação, na mão, do zero até a primeira requisição atendida com sucesso? E depois, para destruí-la, quando você não precisar mais?

Solução: Infraestrutura como Código

- A infraestrutura toda é descrita como código;
- Torna as coisas mais previsíveis;
- Favorece a revisão conceitual da infra;
- Facilita bastante testar arquiteturas diferentes - cria, testa, destrói;
- Oferece versionamento no git;
- Uma vez codificada, você cria, recria, atualiza e destrói tudo (ou parte) com um comando.

No final das contas sai mais barato (tempo e dinheiro) criar, testar, modificar, manter e destruir a infraestrutura.

HashiCorp

Terraform

Write, Plan, and Create Infrastructure as Code

Terraform | o que faz?

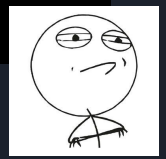
- Permite descrever sua infraestrutura como código ✅
- Oferece uma linguagem única entre nuvens (AWS, Azure, GCP, Digital Ocean, etc)
- Seus arquivos de código podem ser versionados e compartilhados no git ✅
- É possível ter um plano de mudança antes da mudança acontecer de fato
- As dependências entre os recursos são graficamente visíveis
- Permite manter paridade entre os ambientes (staging, QA, prd, etc) ✅
- Mantém um histórico local ou remoto da sua infraestrutura ✅
- É possível criar, modificar, destruir quantas vezes for necessário e incrementalmente ✅
- Fluxos mais complexos também podem ser criados encadeando comandos em scripts

Terraform | como começar?

#1 Fazer download e instalação

#2 Inicializar o projeto

```
Windows PowerShell
PS D:\Temp\terrashop> terraform init
Terraform initialized in an empty directory!

The directory has no Terraform configuration files. You may begin working
with Terraform immediately by creating Terraform configuration files.
PS D:\Temp\terrashop>
```



https://www.terraform.io/downloads.html
https://www.terraform.io/docs/commands/init.html

macOS
64-bit

FreeBSD
32-bit | 64-bit | Arm

Linux
32-bit | 64-bit | Arm

OpenBSD
32-bit | 64-bit

Solaris
64-bit

Windows
32-bit | 64-bit

Seu primeiro recurso na AWS

Terraform | seu primeiro recurso na AWS

main.tf

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

0 references
resource "aws_instance" "myapi" {
  ami           = "ami-[redacted]"
  instance_type = "t2.large"
}
```

Windows PowerShell

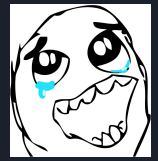
```
PS D:\Temp\terrashop> terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.27.0...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 2.27"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Terraform | seu primeiro recurso na AWS

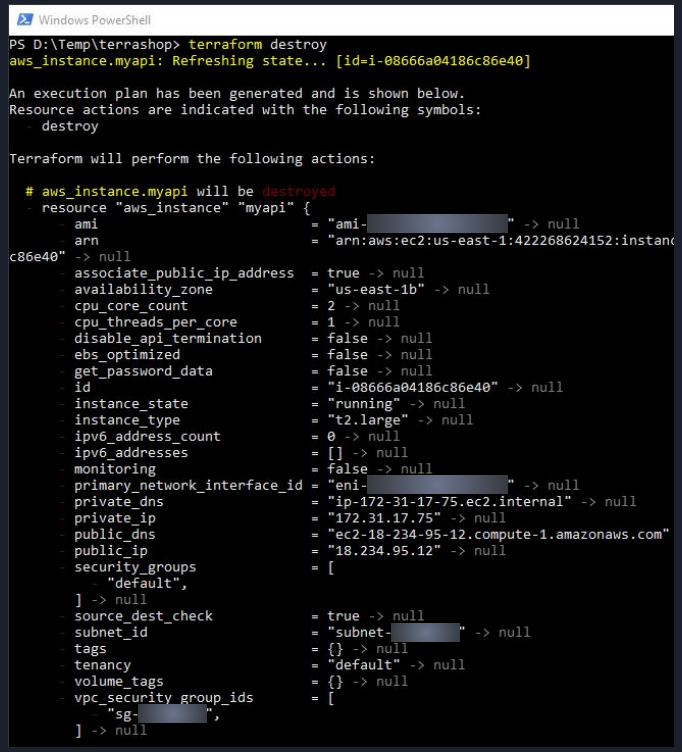
```
Windows PowerShell
PS D:\Temp\terrashop> terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.myapi will be created
  + resource "aws_instance" "myapi" {
      + ami                          = "ami-[redacted]"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.large"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + network_interface_id         = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
      + private_dns                  = (known after apply)
      + private_ip                   = (known after apply)
      + public_dns                   = (known after apply)
      + public_ip                    = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tenancy                      = (known after apply)
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = (known after apply)

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }

      + ephemeral_block_device {
          + device_name  = (known after apply)
          + no_device    = (known after apply)
          + virtual_name = (known after apply)
        }

      + network_interface {
          + delete_on_termination = (known after apply)
          + device_index          = (known after apply)
          + network_interface_id  = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

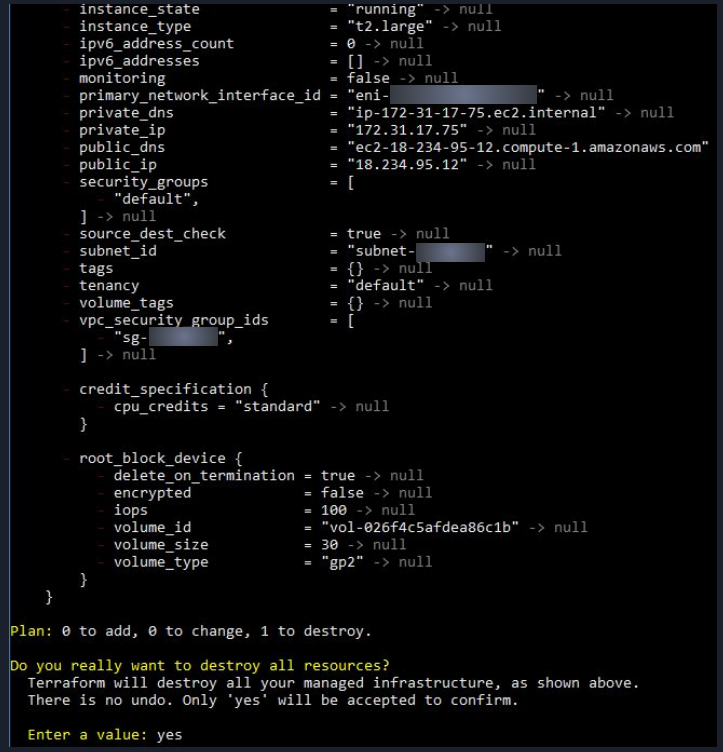
Terraform | seu primeiro recurso na AWS

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.myapi: Creating...
aws_instance.myapi: Still creating... [10s elapsed]
aws_instance.myapi: Still creating... [20s elapsed]
aws_instance.myapi: Creation complete after 29s [id=i-08666a04186c86e40]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```



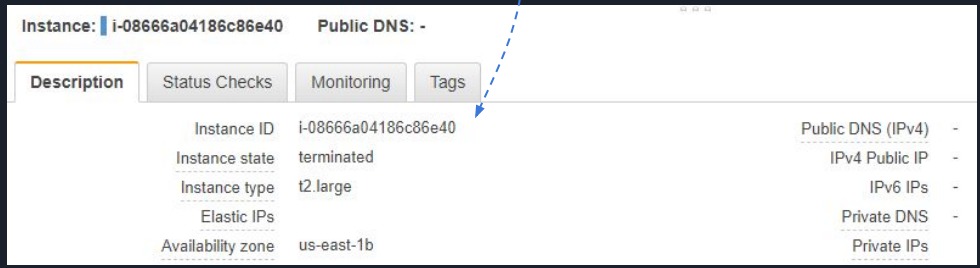
Terraform | seu primeiro recurso na AWS



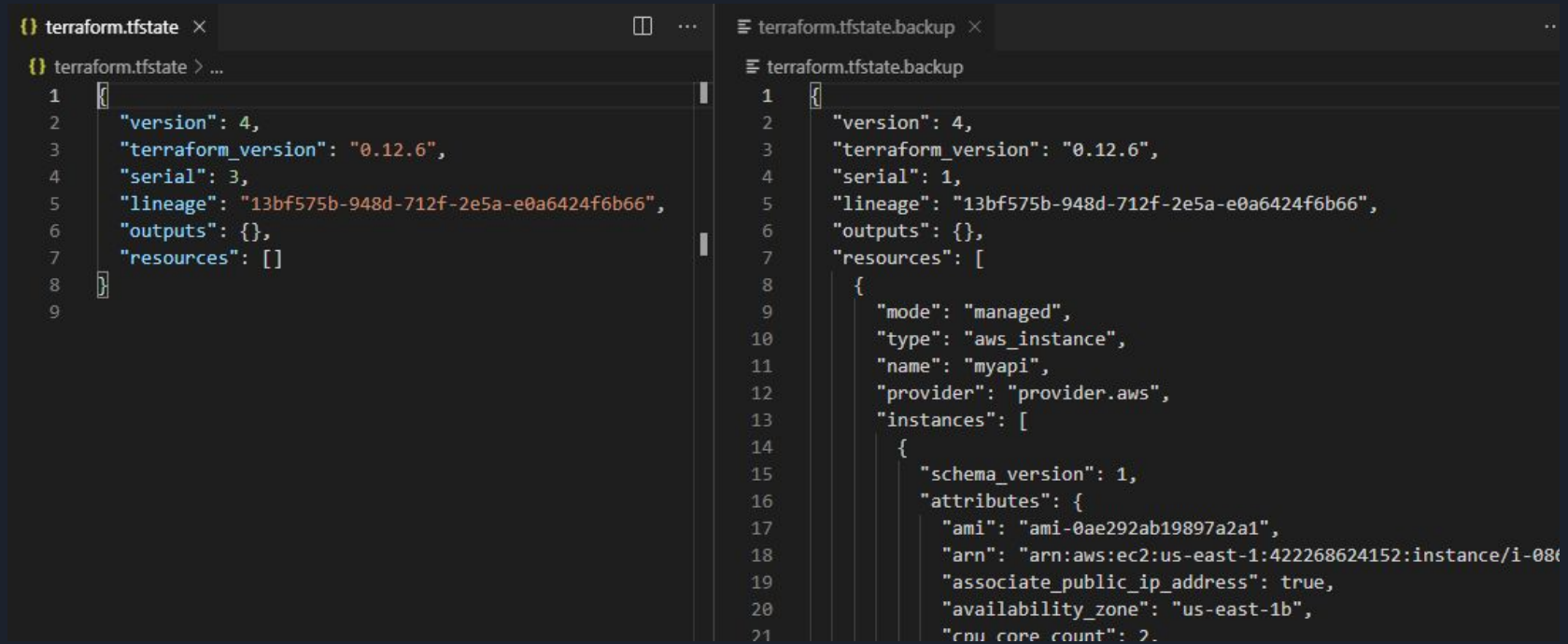


Terraform | seu primeiro recurso na AWS

```
Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.myapi: Destroying... [id=i-08666a04186c86e40]
aws_instance.myapi: Still destroying... [id=i-08666a04186c86e40, 10s elapsed]
aws_instance.myapi: Still destroying... [id=i-08666a04186c86e40, 20s elapsed]
aws_instance.myapi: Still destroying... [id=i-08666a04186c86e40, 30s elapsed]
aws_instance.myapi: Destruction complete after 34s

Destroy complete! Resources: 1 destroyed.
```



Terraform | seu primeiro recurso na AWS



Um exemplo mais completo

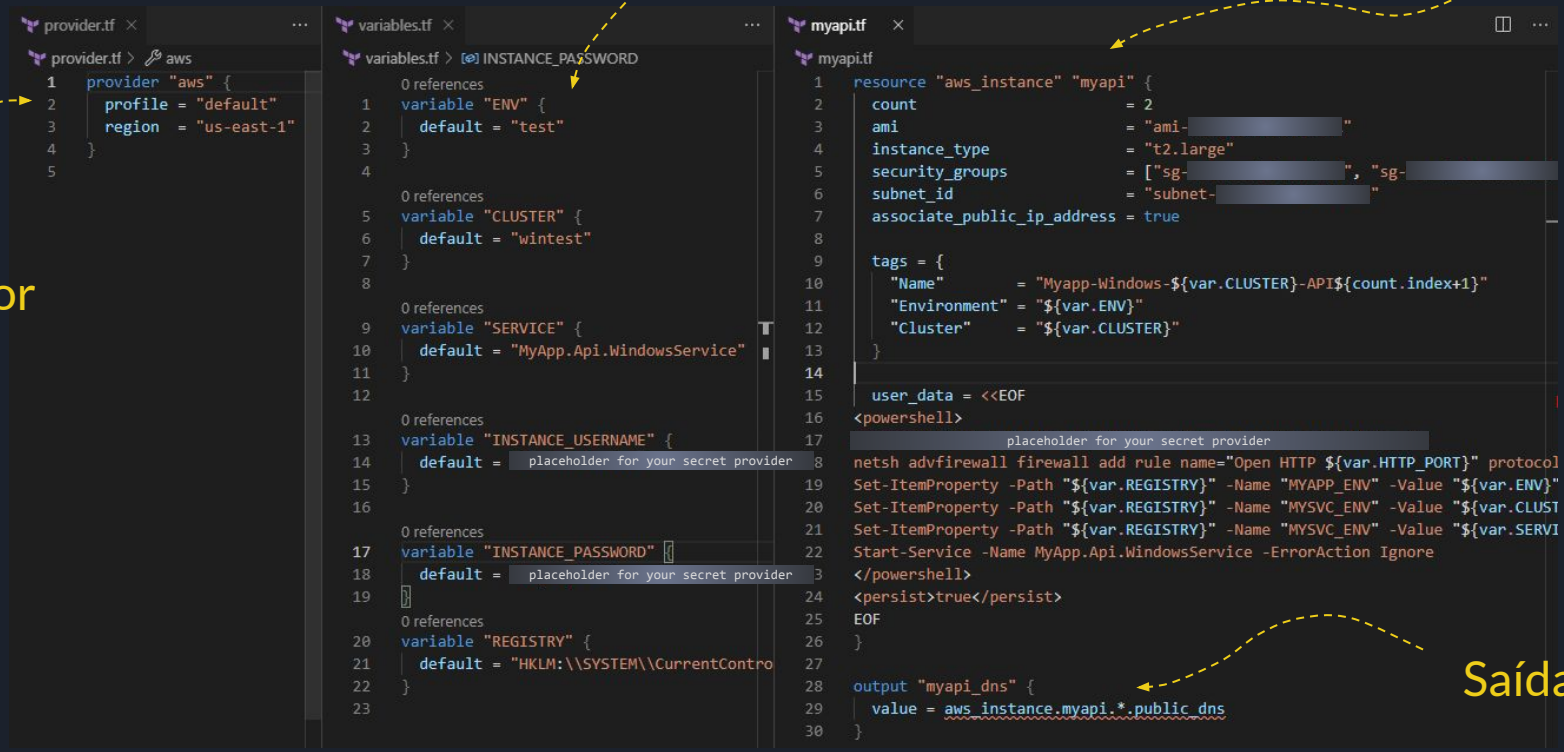
Terraform | um exemplo mais completo

Recursos

Variáveis

Provedor

Saídas



provider.tf

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

variables.tf

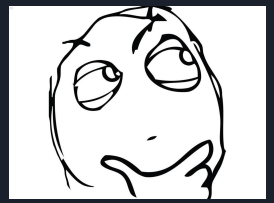
```
0 references
variable "ENV" {
  default = "test"
}

0 references
variable "CLUSTER" {
  default = "wintest"
}

0 references
variable "SERVICE" {
  default = "MyApp.Api.WindowsService"
}

0 references
variable "INSTANCE_USERNAME" {
  default = placeholder for your secret provider
}

0 references
variable "INSTANCE_PASSWORD" {
  default = placeholder for your secret provider
}
0 references
variable "REGISTRY" {
  default = "HKLM:\\SYSTEM\\CurrentContro
}
```

myapi.tf

```
resource "aws_instance" "myapi" {
  count                       = 2
  ami                         = "ami-[redacted]"
  instance_type               = "t2.large"
  security_groups             = ["sg-[redacted]", "sg-[redacted]
  subnet_id                   = "subnet-[redacted]"
  associate_public_ip_address = true

  tags = {
    "Name"        = "Myapp-Windows-${var.CLUSTER}-API${count.index+1}"
    "Environment" = "${var.ENV}"
    "Cluster"     = "${var.CLUSTER}"
  }

  user_data = <<EOF
<powershell>
placeholder for your secret provider
netsh advfirewall firewall add rule name="Open HTTP ${var.HTTP_PORT}" protocol
Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYAPP_ENV" -Value "${var.ENV}"
Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYSVC_ENV" -Value "${var.CLUST
Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYSVC_ENV" -Value "${var.SERVI
Start-Service -Name MyApp.Api.WindowsService -ErrorAction Ignore
</powershell>
<persist>true</persist>
EOF
}

output "myapi_dns" {
  value = aws_instance.myapi.*.public_dns
}
```

Terraform | um exemplo mais completo

```
Windows PowerShell
PS D:\Temp\terrashop> terraform apply

Error: Reference to undeclared input variable

  on myapi.tf line 18, in resource "aws_instance" "myapi":
  18: netsh advfirewall firewall add rule name="Open HTTP ${var.HTTP_PORT}" proto
ocalport=${var.HTTP_PORT} action=allow

An input variable with the name "HTTP_PORT" has not been declared. This
variable can be declared with a variable "HTTP_PORT" {} block.


Error: Reference to undeclared input variable

  on myapi.tf line 18, in resource "aws_instance" "myapi":
  18: netsh advfirewall firewall add rule name="Open HTTP ${var.HTTP_PORT}" proto
ocalport=${var.HTTP_PORT} action=allow

An input variable with the name "HTTP_PORT" has not been declared. This
variable can be declared with a variable "HTTP_PORT" {} block.
```



Terraform | um exemplo mais completo

```
Windows PowerShell
PS D:\Temp\terrashop> terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.myapi[0] will be created
  + resource "aws_instance" "myapi" {
      + ami                          = "ami-[redacted]"
      + arn                          = (known after apply)
      + associate_public_ip_address  = true
      + availability_zone            = (known after apply)
```

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.myapi[1]: Creating...
aws_instance.myapi[0]: Creating...
aws_instance.myapi[0]: Still creating... [10s elapsed]
aws_instance.myapi[1]: Still creating... [10s elapsed]
aws_instance.myapi[0]: Still creating... [20s elapsed]
aws_instance.myapi[1]: Still creating... [20s elapsed]
aws_instance.myapi[1]: Creation complete after 22s [id=i-080d5cfd225936182]
aws_instance.myapi[0]: Creation complete after 29s [id=i-029ab0b06c9aa123a]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

myapi_dns = [
  "ec2-100-25-255-20.compute-1.amazonaws.com",
  "ec2-34-205-225-116.compute-1.amazonaws.com",
]
```

Name	aws:autoscal	Instance ID	Instance Type	Availability Zone	Instance State
Myapp-Windows-wintest-API1		i-029ab0b06c9aa123a	t2.large	us-east-1c	running
Myapp-Windows-wintest-API2		i-080d5cfd225936182	t2.large	us-east-1c	running

Instances: i-029ab0b06c9aa123a (Myapp-Windows-wintest-API1), i-080d5cfd225936182 (Myapp-Windows-wintest-API2)

Description | Status Checks | Monitoring | Tags

- i-029ab0b06c9aa123a: ec2-100-25-255-20.compute-1.amazonaws.com
- i-080d5cfd225936182: ec2-34-205-225-116.compute-1.amazonaws.com

Todos os recursos foram destruídos minutos depois.

```
PS> terraform destroy
```

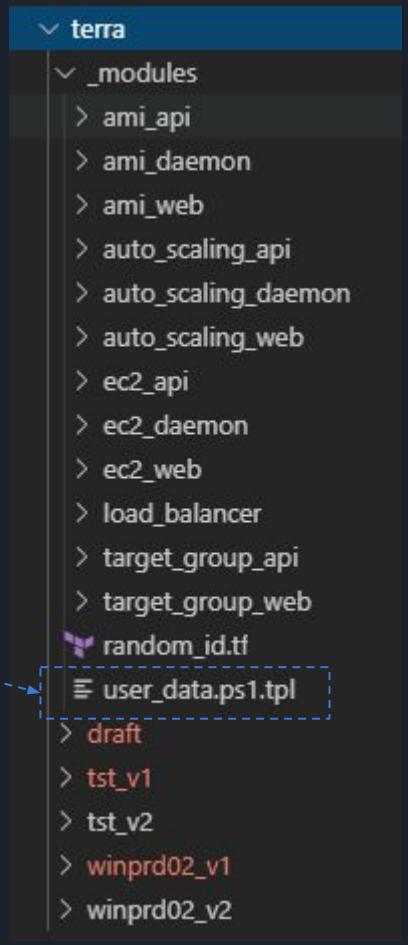
Boas práticas

Terraform | boas práticas

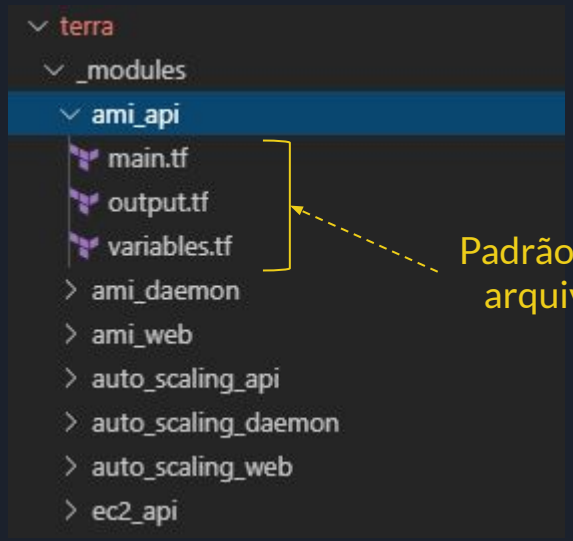
Módulos de cada componente da stack

Script e setup das instâncias EC2 de cada nó

Projetos diferentes para cada versão

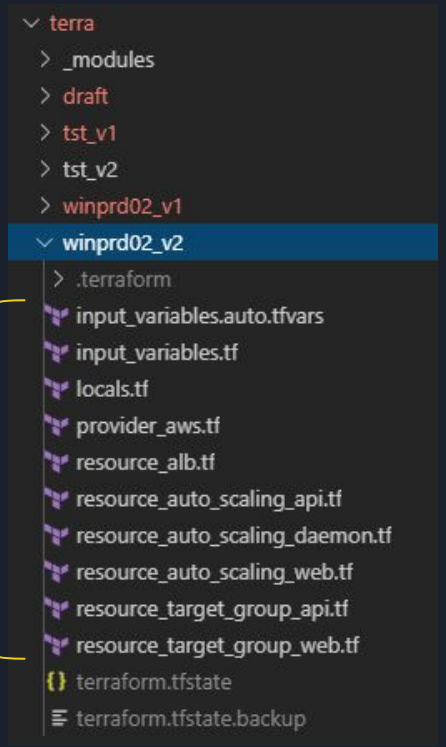


Terraform | boas práticas



Padrão de segregação de arquivos dos módulos

Uso dos módulos para compor a stack



Uma breve introdução ao devops de EC2

Problema

Idealmente, as instâncias EC2 deveriam viver tanto quanto a versão do software que estão rodando - quando não, até menos do que isso. Mas...

- Criar AMI de Windows é um saco;
- Fazer isso a cada nova versão do software, deuzulivre!
- Atualizar uma a uma das máquinas do cluster é f%d@.

Solução: fast food all the way

- Ter um repositório de versões em algum lugar - *e.g.* S3, git, nuget;
- Ter um catálogo que defina a versão atual de cada cluster - *e.g.* arquivo json no S3;
- Criar uma AMI base, com configurações e softwares adicionais - *e.g.* wk, ipp-printer;
- Criar um script que, na criação da EC2, faça o setup da versão que baixou do repositório;
- A partir daí, toda nova EC2 nasce com a versão desejada do software;
- Torna-se possível atualizar a versão das EC2 via auto scaling “increase” ⇨ “decrease”

“Dois hambúrgueres, alface, queijo, molho especial, cebola, picles num pão com gergelim.” - na medida do possível, montados por scripts que vão fazer sempre a mesma coisa, sempre do mesmo jeito.

Script de user_data

Devops | script de user_data

https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-windows-user-data.html

Running Commands on Your Windows Instance at Launch

When you launch a Windows instance in Amazon EC2, you can pass user data to the instance. Instance user data is treated as opaque data; it is up to the instance to interpret it. For example, you can specify data to be used by automated configuration tasks or specify scripts that are run after the instance starts. User data is processed by EC2Config on Windows Server 2012 R2 and earlier and by EC2Launch on Windows Server 2016 and later.

User Data and the Tools for Windows PowerShell

You can use the Tools for Windows PowerShell to specify, modify, and view the user data for your instance. For information about viewing user data from your instance using instance metadata, see Retrieve Instance User Data. For information about user data and the AWS CLI, see User Data and the AWS CLI in the *Amazon EC2 User Guide for Linux Instances*.

Example: Specify Instance User Data at Launch

Create a text file with the instance user data. To execute user data scripts every time you reboot or start the instance, add `<persist>true</persist>`, as shown in the following example:

```
<powershell>
$file = $env:SystemRoot + "\Temp\" + (Get-Date).ToString("MM-dd-yy-hh-mm")
New-Item $file -ItemType file
</powershell>
<persist>true</persist>
```

User Data Scripts

For EC2Config or EC2Launch to execute scripts, tag you use depends on whether the commands PowerShell.

If you specify both a batch script and a Window runs next, regardless of the order in which they

Syntax for Batch Scripts

Specify a batch script using the `script` tag. Sep

```
<script>
echo Current date and time >> %SystemRoot%\T
echo %DATE% %TIME% >> %SystemRoot%\Temp\test
</script>
```

By default, the user data scripts are executed or you reboot or start the instance, add `<persist>`

```
<script>
echo Current date and time >> %SystemRoot%\T
echo %DATE% %TIME% >> %SystemRoot%\Temp\test
</script>
<persist>true</persist>
```

Syntax for Windows PowerShell Scripts

The AWS Windows AMIs include the AWS Tools associate an IAM role with your instance, you do

Devops | script de user_data

Você pode incluir aqui o script PowerShell que for preciso:

- Criar usuário e definir privilégios;
- Criar regras de firewall;
- Definir variáveis de ambiente;
- Fazer download de versão;
- Instalar versão;
- Fazer boot de serviço;
- Etc, etc, etc.

```
15  user_data = <<EOF
16  <powershell>
17  placeholder for your secret provider
18  netsh advfirewall firewall add rule name="Open HTTP ${var.HTTP_PORT}" protocol
19  Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYAPP_ENV" -Value "${var.ENV}"
20  Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYSVC_ENV" -Value "${var.CLUST
21  Set-ItemProperty -Path "${var.REGISTRY}" -Name "MYSVC_ENV" -Value "${var.SERVI
22  Start-Service -Name MyApp.Api.WindowsService -ErrorAction Ignore
23  </powershell>
24  <persist>true</persist>
25  EOF
```

Isto é um exemplo didático!

Idealmente, ele vai estar em um arquivo separado, que possa ser testado

Devops | script de user_data

user_data.ps1.tpl

1

```
<powershell>
Read-S3Object -BucketName "myapp-scripts" -KeyPrefix "devops/" -Folder "C:\App\Scripts"
Invoke-Expression "C:\App\Scripts\boot.ps1"
Invoke-Expression "C:\ProgramData\Amazon\EC2-Windows\Launch\Scripts\InitializeInstance.ps1 -Schedule"
</powershell>
<persist>true</persist>
```

ec2_api/main.tf

2

```
data "template_file" "user_data_api" {
  template = "${file(var.USER_DATA)}"
  vars = {...}
}
```

3

```
resource "aws_instance" "api" {
  user_data = "${data.template_file.user_data_api.rendered}"
}
```

Conclusão

Conclusão

- Fazer na mão o setup de stacks inteiras de uma aplicação é um trabalho árduo e bastante suscetível a falhas;
- Repetir o processo de criação manualmente, igualzinho ao anterior, é muito difícil e aumenta a chance de cometer erros;
- Processos manuais que se repetem custam caro - se você tem que fazer um processo mais de uma vez, é melhor automatizar;
- **Terraform** te ajuda a criar uma infraestrutura mais consistente e estável;
- Você cria, testa e destrói com alguns poucos comandos;
- Sua infraestrutura fica descrita como código que pode ser revisado e versionado;
- Você não precisa mais manter de pé aquele ambiente de troubleshooting que não está precisando no momento - quando você precisar novamente, você o recria;
- Tudo isso junto, no final das contas, reduz bugs na aplicação e custo de manutenção.