



Projeto Cloud - Infraestrutura como código

Rafael Eli Katri

Link do repositório: https://github.com/rafakatri/laaS_Terraform.git

Contexto

Infraestrutura como código (IaC) é o processo de gerenciar a infraestrutura do seu ambiente programaticamente, desde a definição de recursos de hardware e rede até o deploy de aplicações. Isso é consideravelmente relevante dado que elimina as chances de erro humano que acontecem ao fazer esse tipo de tarefa manualmente e padroniza a infraestrutura do ambiente. Além disso, uma vez que o código está correto, é possível replicar a infraestrutura com consistência caso o usuário queira usá-la em outra situação ou tenha acontecido um erro e se necessita voltar a uma versão estável da infraestrutura.

Nesse contexto, Terraform é uma linguagem open-source de IaC desenvolvida pela HashiCorp que possui suporte para diversos provedores Cloud, como AWS, Azure, Google Cloud e entre outros. A linguagem se baseia em blocos de recursos, que apresentam tipos específicos para cada serviço de um provedor, oferecendo uma sintaxe similar mas informações específicas para o contexto da Cloud utilizada.

A proposta do projeto é ter um primeiro contato com os conceitos de infraestrutura como código com Terraform. O provedor Cloud escolhido para o projeto é a AWS, a maior plataforma de nuvem atual em market share. Estruturada em serviços, a AWS oferece diversos recursos computacionais, base de dados e serviços de escalabilidade de aplicações, entre outros.

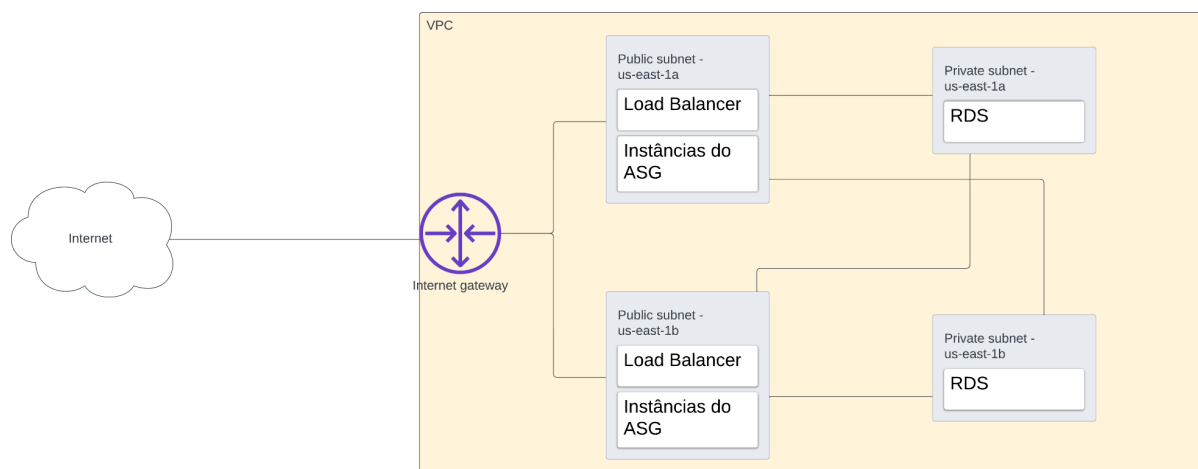
Objetivos

1. Montar uma infraestrutura no Terraform capaz de ser criada e destruída completamente com comandos únicos

2. Fazer o deploy de uma aplicação simples na Infraestrutura, que interage com uma base de dados, distribuída em um grupo de instâncias capaz de escalar quando em stress
3. A aplicação deve mandar Logs e métricas para a AWS

Especificações da Infraestrutura

Vale ressaltar que a aplicação é um CRUD simples que trata um objeto usuário que possui nome e e-mail, e serve apenas para fins de testes. A seguir seguem as escolhas feitas para montar a infraestrutura desejada.



VPC

VPC é um serviço AWS para descrever as características de rede do ambiente em que instâncias e outros recursos de serviços diferentes são alocados. A VPC foi criada com CIDR 10.0.0.0/16 na região us-east-1.

Foram criados 4 subnets divididos em 2 pares. Cada par é de uma mesma AZ, zona de disponibilidade, sendo um deles uma subnet privada e outra pública. Subnets da zona us-east-1a: 10.0.1.0/24

(privada) e 10.0.2.0/24 (pública). Subnets da zona us-east-1b: 10.0.3.0/24 (privada) e 10.0.4.0/24 (pública).

Foram criados subnets em mais de uma zona de disponibilidade para a aplicação funcionar caso uma das zonas apresentar instabilidades.

Subnets públicas são aquelas que possuem rota para um gateway de internet, possibilitando o acesso de instâncias à internet caso elas possuam um [IP público](#). Já as subnets privadas são aquelas que não possuem acesso à internet (por padrão, sem um gateway de NAT).

As subnet públicas serão usadas pelas instâncias da aplicação, dado que elas precisam ser expostas à internet para serem acessadas e enviar logs e métricas para a AWS. Já as subnets privadas serão usadas pela base de dados MySQL do serviço RDS, que não pode ser exposta para a internet.

EC2

EC2 é um serviço da AWS para lidar com instâncias, recursos de hardware que podem ser [Bare Metal](#) ou virtualizados. O serviço trata desde o launch da instância em uma subnet até a definição de ASG 's, grupos de instâncias escalável, e de load balancers.

No projeto, usamos instância de máquina virtual com imagens de Ubuntu 22.04 com a aplicação pré-instalada e com o modelo de instância t2.micro (1 vCPU , 1 GiB RAM e 8 GiB de armazenamento gp2), como apontado na rubrica do projeto.

Para customizar a AMI, imagem do sistema operacional da instância, e pré-instalar a aplicação, foi realizado o launch de uma instância t2.micro com uma imagem Ubuntu 22.04 e foi estabelecido uma conexão SSH para instalar as dependências. Por fim, o dashboard da AWS foi utilizado para criar uma cópia da imagem atual da instância,



que foi configurada para ser pública para funcionar para qualquer usuário AWS.

Para permitir que requisições para a aplicação sejam distribuídas para mais de uma instância foi criado um Load Balancer, distribuidor de carga, que configura uma URL para redirecionar requisições para um conjunto de instâncias denominado target group. O Load Balancer possui um sistema de Health Check que realiza requisições simples para as instâncias de um target group e verifica se elas estão responsivas. Caso elas não respondam uma determinada quantidade de vezes, o Load Balancer as considera não saudáveis e para de redirecionar requisições para ela.


Também foi configurado um grupo de instâncias auto-escaláveis, conhecido como ASG. ASG 's são capazes de escalar horizontalmente de acordo com a carga que estão recebendo, com políticas bem definidas. A ASG foi associada ao target group do Load Balancer e foi configurada com políticas para aumentar o número de instâncias em um caso o uso de CPU for maior que um threshold e diminuir o número de instâncias em um caso o uso for menor que uma outra threshold.

Foram configurados grupos de segurança, para garantir a segurança da aplicação. Apenas as instâncias do ASG podem fazer requisições para a base de dados MySQL.

Por fim, para permitir que a aplicação envie logs e métricas para a AWS, as instâncias foram atribuídas a um IAM Role que autoriza a comunicação com a AWS via API Boto3.

RDS

RDS é um serviço de base de dados relacionais, como MySQL e Postgres. Como a aplicação é um CRUD simples para fins de testes, foi utilizado um modelo de instância com pouco poder computacional e



pouco armazenamento: o modelo db.t2.micro (1 vCPU e 1GiB RAM) com 5 GiB de armazenamento gp2.

O RDS utilizado é baseado em MySQL e foi configurado com multi AZ. Isso significa que a base de dados está disponível em 2 subnets de zonas de disponibilidade diferentes, de modo que a aplicação ainda funcione caso uma zona falhar.

Outra feature implementada foram backups e manutenção da base de dados. A base foi configurada para fazer backups todos os dias, entre as 14:46 e às 15:16 UTC, e para manter armazenado backups por até 7 dias. Já a manutenção está agendada para segunda-feiras entre 00:00 e 03:00 UTC.


Como mencionado anteriormente, o RDS se encontra em subnets privadas, com grupo de segurança que apenas permite acesso à dispositivos que estão nas subnets da aplicação.

CloudWatch

CloudWatch é um serviço que monitora logs e métricas e é capaz de usar alarmes e políticas para tomar ações a partir dos dados coletados.

Certos serviços, como EC2, já enviam logs e métricas por padrão. No caso da EC2, métricas como uso de CPU, escritas no armazenamento ou pacotes recebidos via rede são alguns exemplos. Aplicações também são capazes de enviar logs e métricas para o CloudWatch usando a API Boto3. A aplicação utilizada usa logs para indicar que uma requisição foi feita, por exemplo.

Um ponto chave para o CloudWatch são os alarmes. Um alarme é definido usando alguma métrica disponível, de aplicação ou padrão da AWS, e usando alguma operação matemática como média ou soma dos valores. O resultado dessa operação será comparado com um



threshold, e caso o resultado da comparação seja verdadeiro, a AWS indica que o alarme foi acionado e qualquer política associada a ele é acionada.


No projeto foram utilizados dois alarmes que têm como métrica o uso de CPU, uma métrica padrão para EC2. O primeiro alarme verifica se a média do uso de CPU da ASG é menor que 5%, e se for acionado reduz o número de instâncias em 1. Já o segundo alarme verifica se a média de uso de CPU é maior que 10%, e se for acionado aumenta o número de instâncias em 1. Foi adicionado um limite de no mínimo uma instância e no máximo 5 instâncias no ASG.

Esses valores foram escolhidos com base nos testes de carga com o Locust, simulando o uso de até 100 usuários com aumento de 2 usuários por segundo. Os primeiros testes usavam um threshold superior como 70% e inferior como 10% e tiveram resultados pouco satisfatórios com poucas instâncias que ficavam sobrecarregadas. Após diminuir os thresholds, o limite máximo de instâncias impediu situações que criariam instâncias em excesso. Com os valores finais escolhidos de 5% e 10%, o ASG tende a estabilizar com 3 instâncias quando submetido a carga total e a uma instância quando sem cargas.

S3

S3 é um serviço para armazenamento de objetos, com menos recursos do que uma base de dados convencional, mas capaz de ser acessada diretamente por HTTPS, sem precisar de uma API intermediária.

O S3 associa objetos, o termo que se refere ao conjunto do arquivo e seus metadados, à uma chave, que é um identificador único. Já os buckets são um conjunto de pares chave-objeto. Um ponto relevante do S3 é que ele é capaz de manter registro de versões do objeto ao longo do tempo.



No projeto, o serviço S3 foi utilizado para manter registro do arquivo `.tfstate` do Terraform, que indica o estado atual da infraestrutura. A ferramenta de versionamento é particularmente interessante porque permite voltar a estados anteriores da infraestrutura. A vantagem de usar um bucket para manter registro do estado do Terraform é manter todos os desenvolvedores na mesma versão do código Terraform, evitando conflitos de versionamento.

DynamoDB

DynamoDB é um serviço de base de dados NoSQL. O serviço possui features similares ao RDS, se diferenciando no fato de não ser relacional. Para o projeto, a relevância do DynamoDB entra no contexto do gerenciamento de mudanças no estado do Terraform para evitar conflitos de versionamento.

O Terraform apresenta uma feature chamada state lock que serve como um mutex para o arquivo `.tfstate`. Para realizar uma operação de terraform apply ou terraform destroy, que atualizam o arquivo `.tfstate` armazenado no bucket S3, o state lock deve estar livre. Além disso, durante as operações dos comandos o state lock é indicado como em uso, obrigando instruções novas a esperarem a resolução completa da operação, evitando conflitos de versionamento.

Guia de instalação

I. Clonar repositório

O repositório é público e se encontra nesse [link](#).

II. Instalar Terraform

Um guia feito pela HashiCorp se encontra nesse [link](#). Siga os passos recomendados para o seu sistema operacional.

III. Configurar credenciais AWS

Recomenda-se instalar o AWS CLI para gerenciar credenciais. Acesse esse [site](#) e siga as recomendações para seu sistema operacional.

O Terraform também aceita por padrão credenciais no diretório "~/.aws" ou "C:\Users\username\.aws" ou descritas em variáveis de ambiente, mas é preferível usar o AWS CLI.

IV. Configurar bucket S3

Entre no Dashboard da AWS e entre no serviço S3 usando a barra de pesquisa. Clique em "Create Bucket", crie um nome e selecione todas as opções padrões, menos versionamento, que deve ser marcada como "enabled". O bucket deve ser criado na mesma região que a VPC do projeto.

Escreva o nome do bucket criado no campo bucket do bloco backend, localizado no arquivo Main.tf do repositório. Observação: não se confunda com os arquivos Main.tf dentro das pastas de módulo.

V. Configurar lock DynamoDB

Entre no Dashboard da AWS e entre no serviço DynamoDB usando a barra de pesquisa. Clique em "Create table", crie um nome e use "LockID" como Partition Key. Não crie uma Sort Key. Escolha configurações customizadas e selecione a opção

"On-demand" na aba Read/write capacity settings. Deixe o restante nas configurações padrão.

Escreva o nome da tabela criada no campo "dynamodb table" do bloco backend, localizado no arquivo Main.tf do repositório. Observação: não se confunda com os arquivos Main.tf dentro das pastas de módulo.

VI. Rodar comandos Terraform

Execute o seguinte comandando para configurar o backend:

- terraform init

Execute o seguinte comando para verificar as operações a serem realizadas:

- terraform plan

Execute o seguinte comando para orquestrar o deploy da infraestrutura:

- terraform apply


Execute o seguinte comando para remover a infraestrutura:

- terraform destroy

Teste de carga com Locust

Locust é uma biblioteca open-source para teste de carga feita em Python. Seu objetivo é simular o comportamento de usuários interagindo em um site para testar situações de alta demanda. Além do teste de carga, a biblioteca apresenta uma interface gráfica que exibe dados coletados e até plota gráficos.

Para simular o uso de um site amador, dado o caráter simples da aplicação, foi considerado como objetivo oferecer uma experiência satisfatória para 100 usuários simultâneos. Também foi considerado



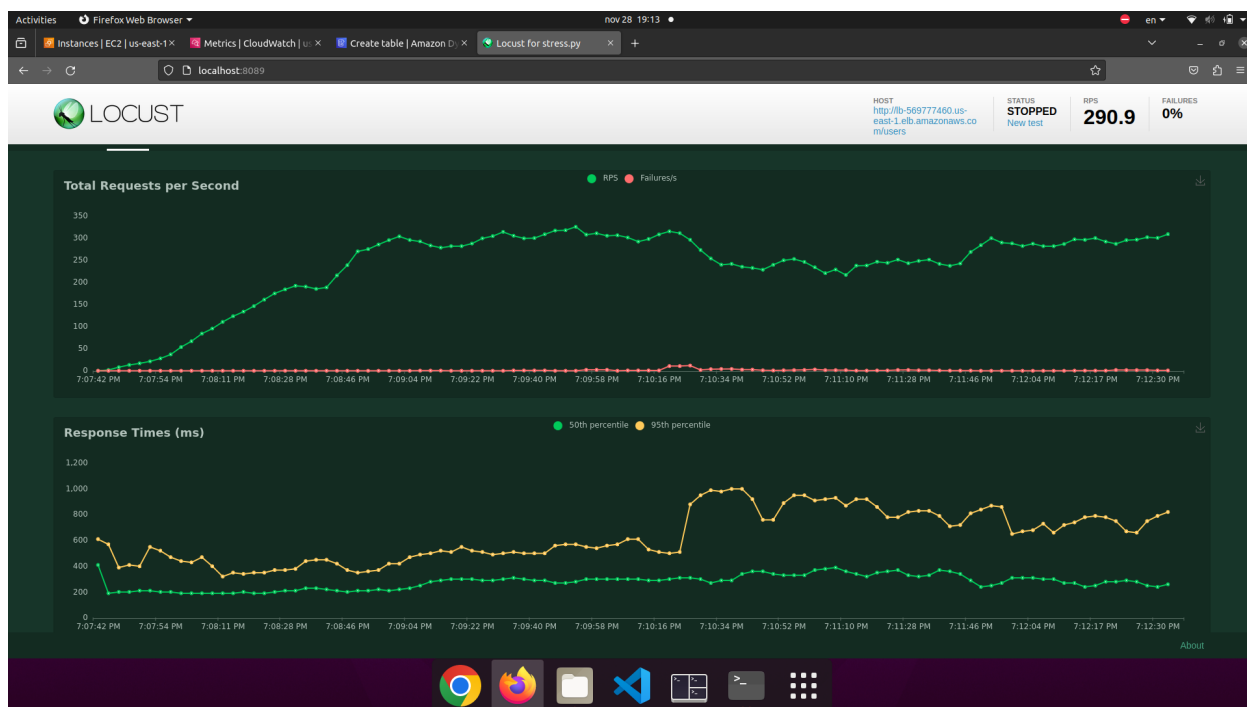
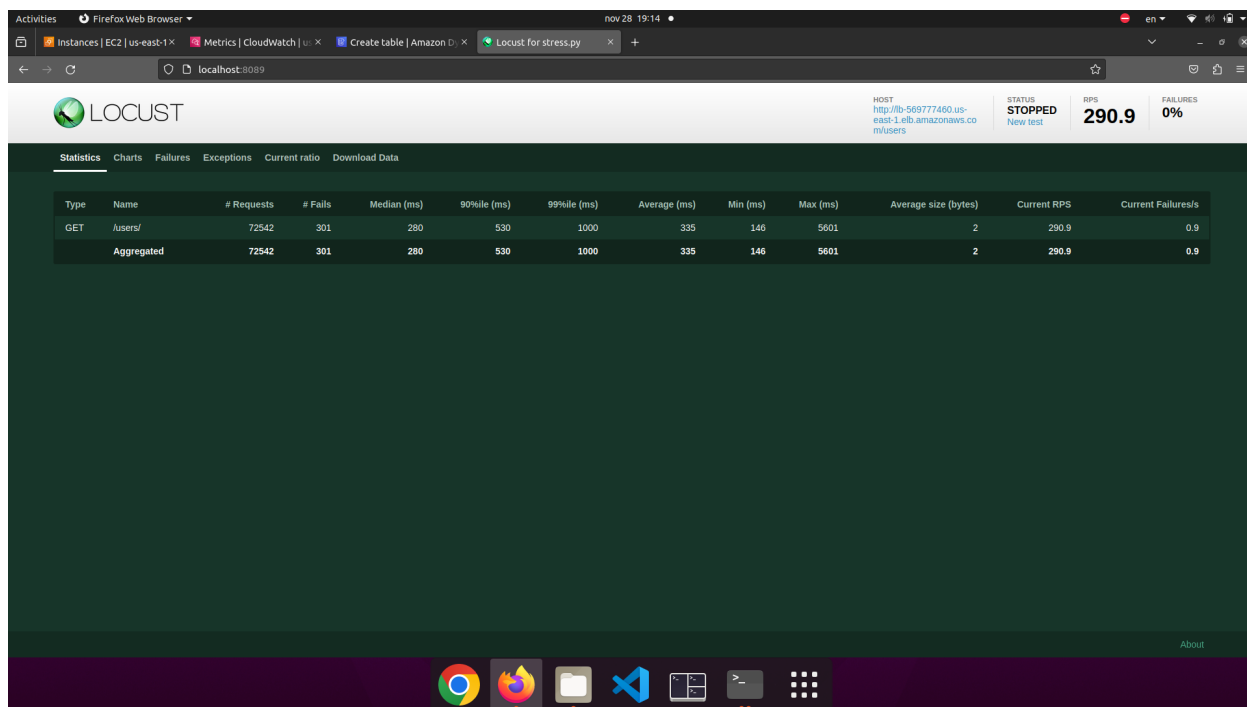
que dois usuários novos apareceriam a cada segundo. O teste foi feito com requisições GET para o CRUD.

Nos primeiros testes, a política de acrescentar instâncias era acionada por um threshold de uso de CPU maior que 70% e a política de decrementar instância era acionada pelo uso de CPU menor que 10%. Esses valores se provaram insatisfatórios, uma vez que sob carga alta as instâncias chegavam até no máximo 40% do uso de CPU e sob carga baixa chegavam até no máximo 5%, levando a um grupo pequeno de instâncias a ser sobrecarregado.

Ao mudar os thresholds superiores e inferiores, respectivamente, para 10% e 5%, o número de falhas no site atingiu aproximadamente 0%. Mesmo que 10% de uso de CPU pareça pouco para um limite, é preferível acionar a política mais cedo para ter menos falhas quando a carga aumenta. Vale ressaltar que o limite máximo de 5 instâncias impediu situações que criariam instâncias em excesso.

O número de instâncias sob carga máxima de usuários se estabiliza em 5. Já sem nenhuma carga, o valor se estabiliza em 1 (limite mínimo da ASG). Para fins de análise de custo, vamos assumir que o objetivo de 100 usuários simultâneos seja a carga média do site.

Para replicar o teste, entre no diretório Locust do repositório e execute o comando "locust -f stress.py". Em seguida abra o browser e acesse localhost:8089.



Análise de custo

Como mencionado anteriormente, foi considerado para análise de custo um uso constante de 100 usuários simultâneos, de modo que a análise considera que a ASG terá constantemente 5 instâncias.

Também, foi considerado que o time de desenvolvedores realiza 20 alterações no estado do Terraform por dia para testar novas features em VPC 's de teste, distribuídas entre os estados de teste, desenvolvimento e produção.

Além disso, foi considerado que a aplicação manda 1GB de Logs para o CloudWatch em um mês e mantém registro de apenas uma métrica, o uso da CPU.

Segue a análise de custo da AWS Pricing Calculator:

Estimate URL: <https://calculator.aws>

/#/estimate?id=895f5d63a2c966211fe476059599a40dd448f47b

Estimate summary

| Upfront cost | Monthly cost | Total 12 months cost |
|--------------|--------------|-----------------------|
| 0.00 USD | 74.86 USD | 898.32 USD |
| | | Includes upfront cost |

Detailed Estimate

| Name | Group | Region | Upfront cost | Monthly cost |
|--|------------------|-----------------------|--------------|--------------|
| Amazon Virtual Private Cloud (VPC) | No group applied | US East (N. Virginia) | 0.00 USD | 0.99 USD |
| Status: - Description: Config summary: Number of active IP addresses (5) | | | | |
| Amazon EC2 | No group applied | US East (N. Virginia) | 0.00 USD | 46.34 USD |
| Status: - Description: Config summary: Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 5), Advance EC2 instance (t2.micro), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (8 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month) | | | | |
| Amazon RDS for MySQL | No group applied | US East (N. Virginia) | 0.00 USD | 26.45 USD |
| Status: - Description: Config summary: Storage for each RDS instance (General Purpose SSD (gp2)), Storage amount (5 GB), Quantity (1), Instance type (db.t2.micro), Utilization (On-Demand only) (100 %Utilized/Month), Deployment option (Multi-AZ), Pricing strategy (OnDemand), Additional | | | | |

backup storage (5 GB)

| | | | | |
|--------------------------|------------------|-----------------------|----------|----------|
| Amazon CloudWatch | No group applied | US East (N. Virginia) | 0.00 USD | 0.80 USD |
|--------------------------|------------------|-----------------------|----------|----------|

Status: -

Description:

Config summary: Number of Metrics (includes detailed and custom metrics) (1), GetMetricData: Number of metrics requested (1), GetMetricWidgetImage: Number of metrics requested (0), Number of other API requests (1), Logs Delivered to CloudWatch Logs: Data Ingested (1 GB)

| | | | | |
|---|------------------|-----------------------|----------|----------|
| Amazon Simple Storage Service (S3) | No group applied | US East (N. Virginia) | 0.00 USD | 0.03 USD |
|---|------------------|-----------------------|----------|----------|

Status: -

Description:

Config summary: S3 Standard storage (1 GB per month), PUT, COPY, POST, LIST requests to S3 Standard (600), GET, SELECT, and all other requests from S3 Standard (600)

| | | | | |
|------------------------|------------------|-----------------------|----------|----------|
| Amazon DynamoDB | No group applied | US East (N. Virginia) | 0.00 USD | 0.25 USD |
|------------------------|------------------|-----------------------|----------|----------|

Status: -

Description:

Config summary: Table class (Standard), Average item size (all attributes) (1 KB), Data storage size (1 GB)