Osquestradores de nuvem: Análise facilidade de uso de serviços da nuvem por meio do Terraform

Alexandre Mitsuru Kaihara 9180092690), Higor Gabriel de Azevedo Santos (170012387), Matheus de Sousa Lemos Fernandes (160137969), Waliff Cordeiro Bandeira (170115810)

***Abstract*—Cloud sevices with low cost, high availability and elasticity has become a interesting resource for many organizations and enterprises aroud the world. It is expected that coud IT infrastructure spending is expected to reach approximately 74 billion U.S. dollars in 2021 and is expected to reach over $100 billion by 2023. Thus, this services are gaining more and more a fundamental role on the dialy lives of many people and companies. On this context, has emerged the Infraestructure as Code, which is the management of networks, virtual machines, load balancers, and connection topology in a descriptive model. And this study aims to evalluate the ease that the Terraform - a cloud orchestrator - can bring to the use of these cloud services.**

***Index Terms*—Terraform, Cloud Orquestrator, Cloud computing, AWS, Azure,**

I. Introdução

A tecnologia está em constante evolução e o campo das comunicações também. A computação em nuvem é uma dessas tecnologias de comunicação, permitindo que dados, software, aplicativos e serviços sejam armazenados, gerenciados, compartilhados e fornecidos pela Internet. Quando compartilhamos documentos, ouvimos streaming de música ou respondemos e-mails na nuvem, tudo isso acontece por meio da computação em nuvem.

Essa tecnologia se tornou parte de nosso dia a dia e é difícil encontrar empresas que não estejam na nuvem ou que planejem entrar na nuvem. O termo computação em nuvem apareceu pela primeira vez em uma palestra acadêmica de Ramnath Chellappa em 1997. No entanto, esse conceito está relacionado ao nome de John McCarthy, o pioneiro da tecnologia de inteligência artificial e criador da linguagem de programação LISP. Em 1960, ele disse que "os computadores poderiam um dia ser considerados uma espécie de utilidade pública".

Nesse contexto, surgiu a Infraestrutura como Código, que é o gerenciamento de redes, máquinas virtuais, balanceadores de carga e topologia de conexão em um modelo descritivo. E o uso dessa tecnologia trás muitos benefícios para organizações, tais como velocidade alocação e gerenciamento de recursos, desenvolvimento eficiente de provisionamento desses recursos, pois é possível padronizar procedimentos, reutilização de recursos, o que economiza muito tempo e diminui as chances de erros de provisionamento por procedimentos manuais, sem padronização. Porém também tem as suas deficiências, uma delas é a necessidade de verificação do correto provisionamento desses recursos e se estes estão sendo utilizados adequadamente conforme o esperado. Assim é necessário avaliar os reais benefícios e facilidades trazidos pelos orquestradores de núvem. Para este estudo focaremos no uso da ferramenta Terraform aplicado às nuvens da AWS e Azure.

II. Metodologia

O experimento proposto envolve a implantação de uma *API RESTful* em um ambiente de Infraestrutura como Serviço (*IaaS*), utilizando dois provedores de nuvem: Amazon Web Services e Microsoft Azure. A implantação desse serviço é feita utilizando dois métodos:

1) Implantação manual, utilizando as ferramentas de linha de comando disponibilizadas pelos provedores;
2) Implantação automatizada, utilizando o orquestrador de nuvem Terraform.

Seguindo esta estrutura, foram realizados 4 experimentos, que são descritos na seção Casos de Uso.

III. Métricas utilizadas na Avaliação

Para analisar ambos os métodos de implantação em um ambiente de nuvem, foram verificadas cinco métricas principais:

1) Sequência de passos: Quantidade de passos necessários para se alcançar objetivo do experimento (implantação de uma *API RESTful* em um ambiente de *IaaS*;
2) Pré-requisitos: Requisitos mínimos necessários para realizar a implantação da ferramenta;
3) Dificuldades: Empecilhos encontrados ao longo da execução dos passos;
4) Qualidade da interface: Facilidade de uso das interfaces utilizadas.

Vale ressaltar que as métricas 3,4 e 5 são subjtetivas, e foram coletadas apenas para fins de realizar um trabalho comparativo entre diferentes ferramentas, do ponto de vista de um usuário comum de provedores de nuvem.

IV. Casos de uso

A. EC2 - Amazon Web Services (AWS)

O serviço de nuvem utilizado é o Elastic Compute Cloud (EC2) da Amazon, [1], um dos serviços de *IaaS* no mundo. Os serviços da AWS foram acessados pela interface de linha de comando disponibilizada pelo AWS, o AWS Command Line Interface (aws-cli) [2]

1) Implementação manual: Para realizar a implantação do serviço foi necessária a execução dos seguintes passos:

1) **Criação de par de chaves de acesso**: Após instalar o Azure CLI, para acessar uma instância EC2 é necessário obter um par de chaves RSA, que é utilizada para acesso criptografado a uma instância remota. O comando utilizado no aws-cli é:

```
aws ec2 create-key-pair \
  --key-name tac-ec2-key \
  --query 'KeyMaterial' \
  --output text > tac-ec2-key.pem
```

   Ao fim da execução armazena-se a chave privada no arquivo **tac-ec2-key.pem**.
2) **Seleção de imagem de instância**: O EC2 é instanciado por meio de imagens de sistemas operacionais chamadas Amazon Machine Images (AMIs). Essas imagens servem para instanciar máquinas previamente configuradas, sem a necessidade de instalação de todos os componentes necessários para utilizar uma máquina virtual. AMIs podem ser encontradas no portal gráfico da AWS ou buscadas por meio da interface de linha de comando. Para este experimento as imagens foram buscadas por meio do portal, visto que a busca de imagens não é uma das métricas do experimento. Foi selecionada uma imagem **Ubuntu Linux 18.04**. Essa versão atualizada foi necessária para a instalação de certas dependências da *API*;
3) **Criação de grupo de segurança**: Para restringir o acesso apenas para pessoal autorizado, o AWS utiliza **grupos de segurança** (*security groups*), que são conjuntos de política de acesso, baseadas em regras previamente especificadas. Os grupos de segurança agem como *firewalls* para uma instância EC2, bloqueando e permitindo origens específicas de tráfego na rede [4].
   Para criação de um grupo de segurança foi utilizado o seguinte comando:

```
aws ec2 create-security-group \
  --group-name tac-sg \
  --description "Grupo de \
  seguranca \
  para o projeto pratico \
  de TAC"
```

   O grupo criado tem nome de **tac-sg**, sendo o nome do grupo o único atrbuto obrigatório desse comando. A resposta inclui a id do grupo de segurança gerada pelo ambiente EC2, e será referenciada a partir de agora como **id-security-group**, visto que se trata de uma credencial confidencial.
4) **Criação de instância EC2**: O passo principal da implantação se trata da criação de uma instância do EC2. Para isso foram utilizados os recursos criados anteriormente: ID da imagem de sistema (AMI), chave criptográfica de acesso (tac-ec2-key) e grupo de segurança. Além disso, foi necessário especificar um tipo de máquina (t2.micro) e um identificador de subrede (subnet-id). Ambas as informações foram encontradas no portal *web* do EC2.
   O comando de criação da instância é:

```
aws ec2 run-instances \
  --image-id <id-ami> \
  --count 1 \
  --instance-type t2.micro \
  --key-name tac-ec2-key \
  --security-group-ids <id-security-group> \
  --subnet-id <id-subrede>
```

5) **Criação de um *gateway***: Para que seja possível acessar a instância de fora do ambiente da AWS, é necessário criar um **internet gateway**, um componente associado a nuvens privadas virtuais (VPCs) da AWS. O *gateway* expõe ä internet pública um endereço IP público e uma URL, o que permite tradução de endereços utilizando os serviços de DNS da Amazon [3]. Dessa forma, usuários externos podem acessar recursos protegidos nas redes internas da AWS.
   O comndo para criar um *gateway* é:

```
aws ec2 create-internet-gateway \
  --query \
  InternetGateway.InternetGatewayId \
  --output text
```

   O comando retorna um identificador para o *gateway* criado (id-gateway).
6) **Associação de um *gateway* a uma VPC**: Depois de criado o *gateway*,é necessário associá-lo ä VPC onde foi criada a instância do EC2.
   O comando para associação do *gateway* é:

```
aws ec2 attach-internet-gateway \
  --vpc-id <id-vpc> \
  --internet-gateway-id <id-gateway>
```

   , onde **vpc-id** é o identificador da VPC, retornado na criação da instância.
7) **Criação de regra de entrada (*ingress* para a subrede**: Para permitir acesso a uma determinada porta, além da criação do *gateway*. é necessário explicitar portas que permitem acesso externo. Essa medida evita que usuários maliciosos procurem acessar outros serviços, que rodam em portas que são publicamente conhecidas (exemplo: HTTP na porta 80 ou SSH na porta 22).
   Para a aplicação proposta foi criada uma regra para tráfego TCP na porta 2000, que é a mesma porta de entrada da aplicação. O comando para criação de de uma regra de entrada é:

```
aws ec2 authorize-security-group-ingress \
    --group-id <id-security-group> \
    --protocol tcp \
    --port 2000 \
    --cidr \
    "$(curl http://ipv4.myip.wtf/text)/32"
```

   Após a criação da regra de acesso,é possível acessar a máquina remotamente por meio de uma conexão

de *secure shell* (*SSH*, utilizando o usuário padrão da instância criada. Para a imagem selecionada, o usuário padrão chama-se **ubuntu**.
Para acessar a máquina, basta executar o seguinte comando:

```
ssh -i tac-ec2-key.pem \
ubuntu@<endereco-da-instancia>
```

, onde **endereço-da-instancia** é o IP ou URL disponibilizados publicamente. Ao fim desse comando o usuário se encontra logado em um *shell* remoto, de forma a realizar a implementação da *API* proposta.

2) Implementação com uso do orquestrador Terraform:

B. VM - Microsoft Azure

Na Microsoft Azure o serviço de IaaS escolhido para comparação com o EC2 da Amazon foi o VM (*Virtual Machine*), em português Máquina Virtual. A VM é um dos vários tipos de recursos de computação sob demanda escalonáveis proporcionados pela Azure. Assim como os demais serviços de IaaS, as VMs da Azure possibilitam uma grande facilidade e flexibilidade na virtualização sem a necessidade de comprar e manter o hardware físico.

1) Implementação manual: Para instanciamento de uma máquina virtual através da Azure, é necessário acessar o portal azure, portal no qual se acessa a maior parte dos recursos disponibilizados pela empresa. Ao acessar o portal para testar a utilização do serviço, foram encontradas as seguintes dependências:

Recurso	Obrigatório	Descrição
Grupo de recursos	Sim	A VM deve estar contida em um grupo de recursos.
Conta de armazenamento	Sim	A VM precisa da conta de armazenamento para armazenar seus discos rígidos virtuais.
Rede virtual	Sim	A VM deve ser membro de uma rede virtual.
Endereço IP público	Não	A VM pode ter um endereço IP público atribuído a ela para acessá-la remotamente.
Interface de rede	Sim	A VM precisa de interface de rede para se comunicar na rede.
Discos de dados	Não	A VM pode incluir discos de dados para expandir os recursos de armazenamento.

Fig. 1. Recursos para utilização da VM

Ou seja, para utilização de uma máquina virtual é necessário utilizar outros serviços do portal. Ao se comparar com outros serviços de IaaS como o da Digital Ocean, o VM da Azure se mostra com maior complexidade no setup do ambiente. No momento do instanciamento da máquina, de forma automática alguns outros serviços são contratados, como o *virtual networks*, entretanto, para alguns outros serviços como o *storage*, abre-se uma caixa de diálogo para que tenha-se a opção de contratar o serviço, uma vez que ele pode gerar cobranças.
A VM possui algumas possibilidades de interação, como CLI diretamente no portal azure, ou acesso via SSH através da máquina pessoal do contratante. A possibilidade de operar diretamente no browser via portal azure, é um fator bem positivo, porém possui sintaxe própria para acesso dos recursos que deve ser previamente entendido.
Para o uso, provisionou-se a máquina, fez o clone de um repositório git criado pelo grupo e que continha uma API com alguns endpoints. Disponibilizou-se essa API através da porta 2000 da VM e tudo ocorreu bem.

Os passos necessários para provisionar a máquina foram:

1) Criar a resource group: É um container que agrupa todo tipo de recurso e informação que deseja-se manipular ou gerenciar como um grupo;
2) Criar a Storage: É necessário para armazenar os dados da máquina virtual;
3) Criar a Virtual Machine: Abre-se uma janela para poder personalizar a máquina;
4) Acessar a Máquina: Acessar a máquina via SSH ou pelo terminal do próprio portal da Azure;
5) Liberar a porta 2000: É necessário liberar o acesso à porta 2000 para o correto funcionamento da API;
6) Executar a API: Iniciar o script de Python com o Flask para prover o serviço da API.

2) Implementação com uso do orquestrador Terraform: Na implementação via Terraform do sistema através da Azure foram necessários os seguintes procedimentos.

1) **Instalação**: É necessário instalar tanto o Terraform e o Azure CLI. Em seguida, é preciso logar no Azure CLI para executar corretamente o Terraform;
2) **Elaboração do script**: Todo o provisionamento de recursos do sistema é realizado em um único arquivo. Primeiro precisa criar ou definir um Resource Group. Em seguida, define-se uma rede virtual, uma subrede, um IP público, uma interface de rede e políticas de segurança de acesso à determinadas portas. Para nosso experimento definimos uma regra que permite o acesso à porta 2000 para conexões TCP. A partir de todas essas definições é possível definir a máquina virtual Linux com a versão do UbuntuServer 18.04 LTS. Também foi necessário enviar um conjunto de operações no terminal para configurar a máquina virtual e conseguir colocar o serviço disponível na web.
3) **Execução**: A execução do script é bem simples, com apenas três comandos é possível provisionar todos os recursos do script, tanto quanto destruí-los em um único comando. Para provisionar a máquina realizam-se os comandos abaixo.

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

V. BENCHMARK

Para estressar o sistema a fim de ter dados para promover uma análise comparativa entre os dois provedores utilizados, foi escolhido o Apache Jmeter que é uma ferramenta para testes de carga e performance. Ao estressar ambos provedores com a mesma quantidade de requisições, espera-se ter uma quantidade de dados minimamente adequada para o fim necessário de comparação dos provedores no contexto do trabalho final da disciplina.

Para testar a API foram enviadas dez mil requisições para a rota /pokemon, utilizando as seguintes configurações do Jmeter:

- 1000 usários (*threads*);
- 10 segundos para tempo de distribuição de carga;
- 10 iterações.

Essas configurações foram utilizadas para os *endpoints* do AWS EC2 e do Azure VMs, e foram gerados dados em formato CSV, de forma a gerar resultados comparativos das duas plataformas.

VI. Análise e Resultados

Os dados mais relevantes para análise são tempo de resposta e latência sobre o tempo. As figuras 2 e 3 apresentam os resultados obtidos.

Vale ressaltar que os tempos do eixo X estão transladados entre si porque os testes foram executados de forma individual para cada provedor, devido a limitações de memória da máquina onde foram executados os testes (ambiente local).

Além disso, foram coletados os resultados relativos a execuções, tempos de resposta, vazão e velocidade da rede, em Kilobits por segundo (Kb/s).

A figura 4 apresenta a tabela gerada pelo JMeter com os resultados citados.

A. Sequência de passos

Em todos os experimentos, seja manual ou com Terraform e na AWS ou Azure, tiveram três passos em comum. A primeira é prover as credenciais das contas para poder utilizar os serviço. O segundo definir configurações de rede e por último instanciar a máquina. Notavelmente, os casos manuais tiveram mais passos, uma vez que precisa-se definir os recursos a serem utilizados manualmente, enquanto que no Terraform tudo é feito pela execução de um único script automatizado.

No entanto, apesar de diminuir a quantidade de passos, o tempo para desenvolver os scripts não é tão intuitivo e apesar da ajuda das documentações, leva um tempo significativo para desenvolver corretamente o script para provisionar os recursos.

A vantagem do uso do método manual é para usos mais simples, como para estudos, para experimentos de baixa escala, já que para cada máquina é preciso executar manualmente cada um dos passos, além de estar sujeito a erros por desatenção ou desconhecimento do usuário. Já com o Terraform, a própria ferramenta indica erros, o motivo, o que precisa ser declarado/provisionado, auxiliando na construção do script. Além disso, é possível dar escala ao provisionamento de recursos de forma automatizada, diminuindo a chance de erros que na execução manual pode eventualmente ocorrer. Porém, para conseguir elaborar o script, necessita-se de um certo nível de conhecimento da sintaxe do Terraform e do funcionamento do provisionamento de recurso da nuvem.

B. Pré-requisitos

1) Manual: Na Azure tem-se a possibilidade de utilizar o CLI disponibilizado no Portal Azure para fazer toda utilização da máquina (sendo que o provisionamento pode ser feito via interface do portal) ou, em uma segunda opção, pode-se acessar via SSH a máquina e fazer de forma semelhante a utilização, no caso de escolher o CLI do Portal Azure tem o empecilho extra da sintaxe própria para utilização de alguns recursos. Na Amazon pode-se acessar via Amazon CLI que tem essa dificuldade extra de sintaxe, assim como na Azure, a AWS também permite utilização da máquina via SSH e provisionamento via interface. O forte da Azure é ter o CLI totalmente disponível via browser e o forte da AWS é o Amazon CLI que é bem robusto.
2) Orquestrador: Via terraform tem-se maior facilidade de reutilização de sistema e compatibilidade entre outros provedores de nuvem, entretanto, para cada provedor ainda tem-se passos e sintaxes específicas para determinados serviços, o que aumenta um pouco a complexidade de uso do terraform, o que se faz necessário aprender as declarações e sintaxe específica de cada provedor. O ganho existe quando faz-se necessário subir uma mesma configuração mais de uma vez, sendo que se tem o código do terraform isso poderá ser feito em poucos minutos evitando retrabalho e de forma manual não.

C. Dificuldades

No método manual, as dificuldades encontradas foram comandos complexos com grande número de parâmetros. Enquanto que todos os parâmetros no Terraform são especificados na documentação e possuem uma sintaxe simplificada.

Tanto no método manual quanto no Terraform tivemos dificuldade em entender com facilidade os recursos necessários para poder instanciar a máquina virtual. Para o caso da Azure foi necessário definir regras de segurança para acesso às portas da máquina e em nenhum local havia a orientação de que devia configurar tais regras para acessar externamente a máquina virtual. A solução foi continuar as buscas fora da documentação oficial até resolver o problema.

D. Qualidade da interface

De forma geral avaliamos a interface tanto da Amazon quanto da Azure como boas e intuitivas. Identificamos como ponto forte na AWS a sua documentação, sendo que contém exemplos interativos que facilitam muito todo o processo. Na Azure, o próprio Portal Azure é um ponto forte, sendo que é possível identificar facilmente os recursos necessários. Outro ponto muito positivo é a possibilidade de fazer toda a utilização do IaaS através do browser, dessa forma, através de qualquer SO seria a mesma usabilidade.

Em contraste, a utilização do Terraform, apesar de prover uma sintaxe simplificada, cada provedor tem a sua forma de declarar e provisionar recursos. Por exemplo, existem várias versões de máquinas com diferentes recursos, porém é necessário uma string que a referencia e a lista de todas essas máquinas é difícil encontrar. E para cada provedor de serviços é necessário fazer um novo script e aprender a forma de provisionar os seus recursos o que demanda estudo e pesquisa nas documentações, que não indicam claramente todos os quisitos necessários para o recurso ser provisionado.

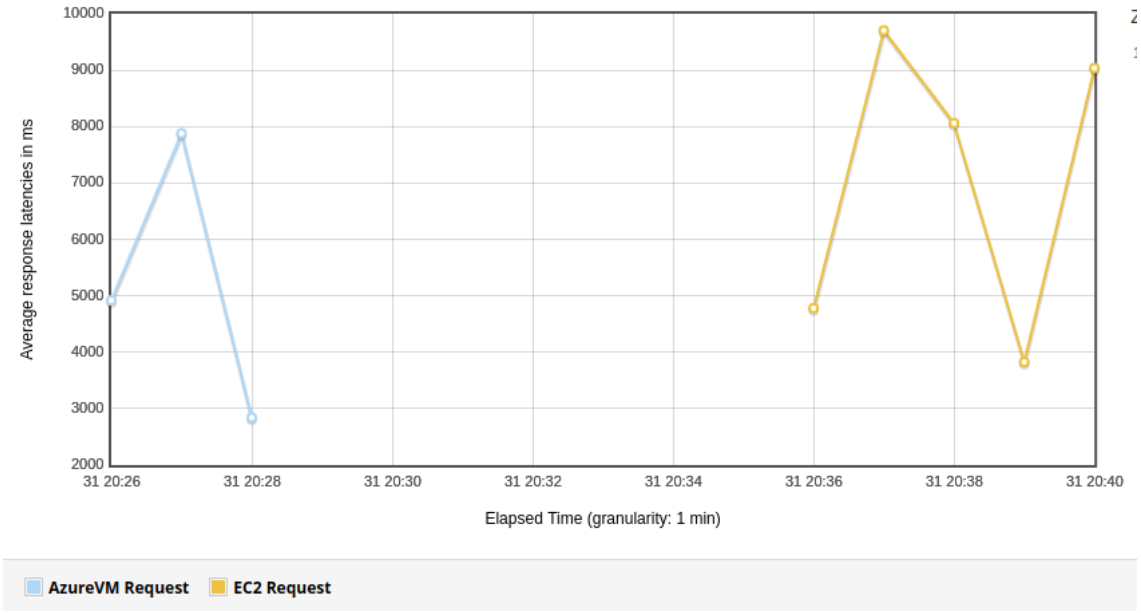

Fig. 2. Resultados de latência média nos testes do JMeter

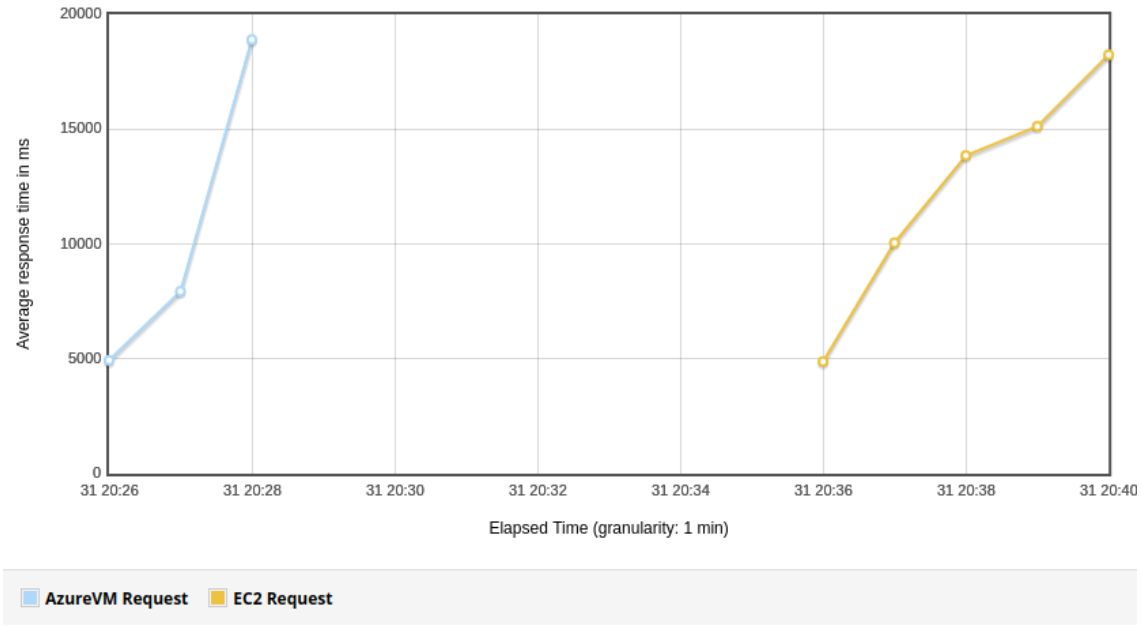

Fig. 3. Resultados de tempo médio de resposta nos testes do JMeter

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**15398**	**541**	**3.51%**	**10738.30**	**102**	**158887**	**1732.00**	**26104.70**	**66453.25**	**133674.00**	**17.88**	**26.32**	**2.47**
AzureVM Request	7699	255	3.31%	9655.92	102	140485	1570.00	16182.00	65971.00	133139.00	48.41	71.05	5.85
EC2 Request	7699	286	3.71%	11820.68	111	158887	2130.00	32971.00	66681.00	134067.00	34.19	50.46	5.30

Fig. 4. Estatísticas de execução contra cada *endpoint*

VII. Conclusão

O uso manual dos provedores de serviço é mais indicado para usuários iniciantes e/ou para aplicações de baixa complexidade dado que a interface é mais amigável e exigem

menos requisitos para poder utilizar, um exemplo é a Azure, em que é possível provisionar uma máquina virtual completa pelo próprio Browser. Já o Terraform, necessita de instalação tanto da Azure CLI e AWS CLI e o próprio Terraform. Além disso, para executar qualquer script é necessário prover todas as credenciais para acessar a conta do provedor, e por vezes é até necessário gerar uma chave privada. tornando o seu uso muito mais complexo.

No entanto, essa complexidade é compensada pelo uso de scripts para provisionar recursos. As vantagens do uso de um orquestrador é que para aplicações mais complexas, a padronização dos scripts e a automação no provisionamento não só economizam tempo como também minimizam os erros da execução manual - seja por desatenção ou desconhecimento do uso da plataforma. Outro dificultante para o uso do Terraform é existe uma curva de aprendizado do uso da sintaxe do Terraform e também a sintaxe única de cada provedor para provisionar os recusos. Portanto, não é possível usar um mesmo script para diferentes nuvens.

REFERENCES

[1] Amazon ec2. https://aws.amazon.com/ec2/. Acessado: Outubro, 2021.
[2] Aws command line interface. https://aws.amazon.com/cli/. Acessado: Outubro, 2021.
[3] Internet gateways. https://docs.aws.amazon.com/vpc/latest/userguide/VPC$_{I}nternet_{G}ateway.html.Acessado$: $Outubro$, 2021.
[4] Security groups for your vpc. https://docs.aws.amazon.com/vpc/latest/userguide/VPC$_{S}ecurityGroups.html.Acessado$: $Outubro$, 2021.