Pontifícia Universidade Católica do Rio de Janeiro

Análise de arquiteturas multi-região para ambientes críticos na nuvem AWS

Autor: Rafael Silveira Piña Rodrigues

Projeto Final

Departamento de Informática

Curso: Engenharia da Computação

RJ 11/2022

Autor: Rafael Silveira Piña Rodrigues

Análise de arquiteturas para ambientes críticos na nuvem AWS.

Relatório de Projeto Final, apresentado ao programa de Graduação em Engenharia da Computação da PUC-Rio como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação

Orientador: Hélio Côrtes Vieira Lopes

Rio de Janeiro 11/22.

Agradecimentos:

- Ao meu orientador Hélio Cortês, pelo apoio e confiança depositada, sempre disponível para auxiliar durante meu processo de desenvolvimento deste projeto final.

- A minha família, amparo principal nessa minha trajetória durante minha graduação e confecção deste projeto final. Sempre acreditando no meu potencial, nos meus sonhos e, por mais que eu tivesse dificuldades ao longo desse período, nunca fui questionado sobre a minha capacidade de chegar até essa etapa final e por isso serei eternamente grato.

- Por fim, aos meus amigos que conheci durante a graduação. Pude conhecer pessoas maravilhosas que foram essenciais para meu desenvolvimento, acreditando nas minhas metas e objetivos.

Rafael Silveira Piña Rodrigues; Hélio Côrtes Vieira Lopes; Análise de arquiteturas multi-região para ambientes críticos na nuvem AWS. Rio de Janeiro, 2022. 59 páginas. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Resumo:

A tecnologia avançou bastante durante os anos e nós, como engenheiros, precisamos garantir que os ambientes estejam preparados para entregarem aos usuários / clientes um serviço de qualidade e sempre disponível para utilização. Esse projeto final consiste no estudo sobre as melhores práticas com relação a arquiteturas multi-região para ambientes críticos na nuvem AWS. O objetivo do projeto visa analisar impactos, práticas de desenvolvimento e exemplos de aplicações no que diz respeito a regiões na nuvem.

Palavras chave: Multi-região, resiliência, computação em nuvem, análise, Amazon Web Services.

Rafael Silveira Piña Rodrigues; Hélio Côrtes Vieira Lopes; Analysis of multi region architectures in critical environments in the AWS cloud. Rio de Janeiro, 2022. 59 pages. Final Project – Department of Informatics, Pontifical Catholic University of Rio de Janeiro.

Abstract:

Technology has advanced a lot over the years and we, as engineers, need to ensure that environments are prepared to deliver quality service to users / customers and always available for use. This final project consists of studying best practices regarding multi-region architectures for critical environments in the AWS cloud. The objective of the project is to analyze impacts, development practices and application examples with regard to cloud regions.

Key words: Multi-Region, resilience, cloud computing, analysis, Amazon Web Services

Sumário

1. Introdução

A partir dos crescentes avanços tecnológicos e da corrida pela tecnologia envolvendo grandes empresas como a Amazon, Oracle, Google, IBM e Microsoft, tornou-se necessária a implementação de uma nova forma de oferecer todos os serviços de computação em nuvem, seja de software, armazenamento de dados, inteligência artificial, entre outros temas da tecnologia. Em 1960 surgiu o interesse da criação de uma rede global para uso de processamento de transações financeiras e dados do censo. Ao longo dos anos, as empresas começaram a migrar os seus sistemas locais para hardware oferecidos pela nuvem, atraídas inicialmente pela redução de custo, praticidade de configuração, eficiência e facilidade sendo necessário apenas o acesso a internet.

Naturalmente, desde a primeira utilização do termo "computação em nuvem" em 1997, essa engenharia tornou-se referência conceitual à nova tecnologia que permite o acesso a programas, arquivos e serviços por meio da internet, que fosse necessário utilizar os sistemas e hardwares locais. Segundo a Dell Technologies [DELL, 2011], na época em que a computação em nuvem estava começando a ser muito utilizada, "um fator que está impulsionando a demanda por computação em nuvem é o crescimento explosivo de dados. O mundo testemunhará um aumento de quatro vezes, até 2015, na quantidade de dados criados e replicados. E assim que surgirem todos esses dados, será necessária uma forma para armazená-los com segurança e permitir que os usuários finais tenham acesso a eles de forma eficiente."

Ao longo dos anos, os provedores de nuvem trabalharam na criação de diversas plataformas com o objetivo de oferecer serviços de fácil acesso e baixo custo para seus clientes, modificando a forma tradicional como as empresas pensam sobre os recursos de TI.

Sua utilização se intensificou exponencialmente, demandando maior conhecimento dos engenheiros acerca das práticas no setor de tecnologia bem como na ampliação dos recursos específicos desta plataforma. Segundo a Dell [DELL, 2011], "em 2009, a receita para serviços em nuvem foi um pouco mais alta que US$58,6 bilhões. A receita estipulada, até 2011, já era elevada e esperava-se que as despesas com TI ultrapassassem US$2,6 trilhões. E com a computação em nuvem respondendo por apenas 2,3% desse mercado global, tinha muito espaço para crescimento. A empresa de pesquisa Gartner projetava que a receita para serviços em

nuvem chegará a US$152,1 bilhões em 2014”. Dessa forma, vemos que desde o início de sua criação a computação em nuvem já era um serviço que rendia muito dinheiro no mercado e esse crescimento estava sendo constantemente impulsionado.

Como mencionado nos tópicos anteriores, quando se fala em computação nas nuvens, fala-se na possibilidade de acessar arquivos e executar diferentes tarefas pela internet sem que seja necessário muita configuração, pois é possível acessar diferentes serviços, softwares e poder computacional quando se tem apenas o acesso à internet. Essa tecnologia possibilita então a elasticidade dos recursos necessários aos usuários.

Antes da existência da computação em nuvem, a demanda por armazenamento de dados, softwares e soluções para os desenvolvedores eram oferecidos localmente (“on premise”), isso significava que as empresas construíam grandes data centers privados com o intuito de servir suas aplicações, mantendo os serviços ativos. Esse modelo de infraestrutura foi deixando de ser a melhor estratégia para aplicações de produção, isso porque possuía diversos impedimentos para o desenvolvimento e o crescimento dessas empresas, como:

- Aumento de custos se comparado com a estrutura sediada na nuvem, já que o modelo de data centers privados o custo pode chegar a 30% maior, isso porque nesse modelo não é flexível e a empresa não paga apenas pelo que é utilizado.

- É preciso investir em segurança contra acessos físicos aos data centers, monitorando quem entrou ou saiu desses espaços, assim como o que foi modificado para que nenhum dado ou informação seja vazada sem autorização.

- Esses hardwares/servidores são equipamentos que utilizam muita energia para entregarem o poder computacional desejado e, como consequência, aumentam a temperatura dos data centers, sendo necessária a manutenção da temperatura das empresas, prevenindo incêndios nesses ambientes.

- Uma outra responsabilidade das empresas nesse modelo de data center privado é a escalabilidade dos seus ambientes, quando necessário.

Nesses casos, será preciso adicionar mais hardware ou aumentar a quantidade de servidores e hacks ou até mesmo expandir C1 - Internal use 3 a capacidade, mudando todos os data centers de localidade, para garantia de maior espaço ou até mesmo introduzindo um banco de dados maior ou mais otimizado para armazenamento já que quanto maior a quantidade de dados teremos uma maior necessidade de espaço físico nos data centers. Neste sentido, tornou-se necessária a expansão dos data centers próprios com o objetivo de oferecerem maior capacidade e, na maioria dos casos, ficou inviável a manutenção desses ambientes, limitando o crescimento da empresa já que gastava-se muitos esforços na manutenção desses data centers (em muitas situações enormes que dependem de manutenções constantes) desviando o foco das tarefas mais importantes.

- Em projetos voltados para desenvolvimento de software previamente a utilização da computação em nuvem, as empresas também utilizavam outra metodologia para estruturação dos projetos internos e externos. Independente do projeto, seja voltado para banco de dados, devops, inteligência artificial, programação web, mobile, robotics, desenvolvimento de jogos ou internet das coisas, as empresas precisavam dedicar desenvolvedores que pudessem desenvolver soluções que visavam auxiliar nos projetos, de qualquer área. Antes da computação em nuvem, por exemplo, se um projeto tivesse o objetivo de construir um sistema que pudesse fazer o processamento de imagens deveria-se construir um código que pudesse utilizar das técnicas de Inteligência artificial para que pudesse fazer a análise correta dessas imagens. A computação em nuvem, se aplicada neste projeto de exemplo, resolveria já que hoje temos soluções prontas de software para reconhecimento facial, como o serviço da Amazon o AWS Rekognition. Então, dessa forma, não seria necessário a locação de funcionários dedicados com o objetivo de desenvolvimento desse software, mas alocados para outra tarefa de apoio ao projeto.

Podemos concluir que, esse modelo de negócio adotado durante anos, foi na maioria das áreas de tecnologia, tornando-se obsoleto com a introdução da computação em nuvem já que, a substituição desses projetos começou migrar as empresas seus projetos para que utilizassem soluções de software prontas, oriundas da nuvem. Para desenvolvimento de software, veio com o propósito de assumir essa

responsabilidade em oferecer data centers/serviços que entregam um centro de processamento de dados, concentrando os sistemas computacionais. Um exemplo prático desta nova forma de utilizarmos serviços para atingirmos um desenvolvimento ágil de um projeto, é o AWS Rekognition, serviço da AWS que oferece ao usuário o reconhecimento de imagens e vídeos utilizando machine learning.

Em um cenário hipotético, em que uma empresa qualquer deseja construir uma aplicação que pudesse reconhecer arquivos de imagens e vídeos automaticamente para liberação de seus funcionários no prédio para trabalharem. Nesse caso, se não tivessemos a possibilidade de usarmos esse serviço pronto como apoio ao negócio da empresa, seria obrigqtório o desenvolvimento de um novo produto, demorado e custoso. No entanto, a empresa sairia ganhando nesse cenário de utilização desse serviço/produto pronto na nuvem para reconhecimento de imagem de vídeos por conta da rapidez na entrega do produto final desejado.

Segundo estudo da [IBM, 2013], pensando no usuário comum, apartado do desenvolvimento de sistemas, nossa sociedade tem utilizado cada vez mais os serviços na nuvem e tornou-se impossível viver sem esses serviços. Seja para calendários, organizador de viagens, lista de compras compartilhada, backup de telefones, filmes, música, documentos pessoais, todos esses serviços utilizam a nuvem como provedor de serviços amplos, de qualidade e alta disponibilidade. Abaixo podemos ver apenas uma amostragem de grandes empresas que utilizam a computação em nuvem para suas aplicações:

[Figura 1 – Empresas que aderiram a nuvem como modelo de negócio]

Fonte: [Medium, 2018]

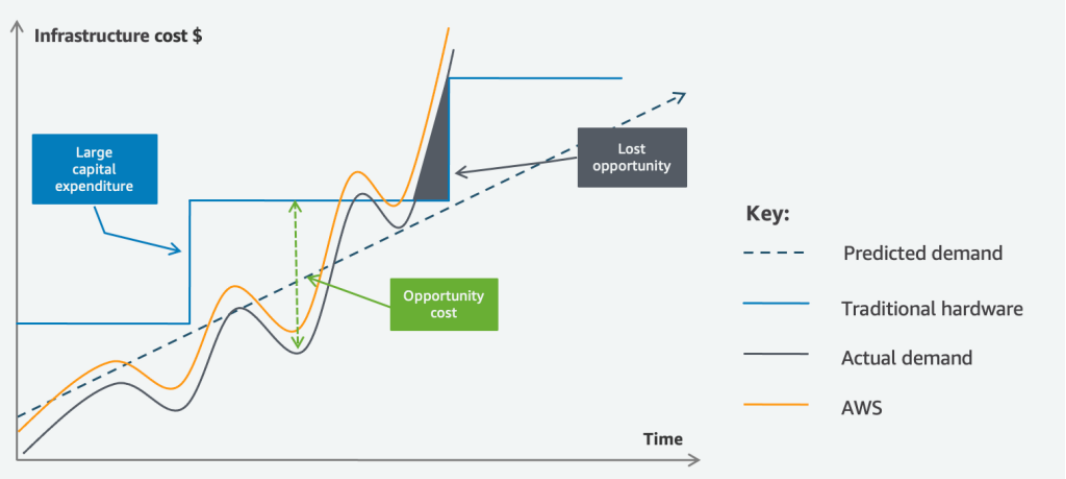
Essas empresas foram atraídas para migrarem sua estrutura para nuvem por conta dos benefícios que essa mudança proporcionava no longo prazo. Atualmente existem vários benefícios na utilização da computação em nuvem que devem ser considerados durante o processo de desenvolvimento de um sistema:

- Escala e alto desempenho;
    - Permite escalar o poder computacional sempre que necessário dependendo da aplicação. Um caso real como exemplo desse benefício, seria a escalabilidade automática durante eventos da Black Friday já que nessa época do ano as consultas aos bancos de dados aumentam exponencialmente, sendo assim novas máquinas são criadas para abranger essa demanda sem que hajam impactos, como por exemplo quando um site deixa de atender a requisição do usuário por conta da grande quantidade de consultas ao mesmo tempo. O benefício do alto desempenho é evidenciado justamente durante esses eventos, já que é possível garantir que o ambiente não fique fora do ar por conta desse aumento na demanda. Existem casos em que empresas são prejudicadas quando não utilizam a computação em nuvem para escalabilidade automática. Um exemplo disso

foi a venda de ingressos para o show da Sandy no dia dia 01/06/2022 venda de ingressos tem fila de fãs e instabilidade no site,

- Flexível;
    - A nuvem é extremamente flexível porque praticamente tudo é customizável, isso significa que durante a criação de qualquer recurso/funcionalidade na nuvem podemos escolher, por exemplo, tipo de máquinas, linguagem de programação que gostaríamos de utilizar, o tipo de banco de dados, tamanho do armazenamento, entre outros serviços necessários.
- Econômica;
    - É extremamente econômica, já que podemos reduzir consideravelmente os custos de uso dos recursos. O usuário pode facilmente configurar horário de desligamento de recursos, além de diferentes planos de pagamento com contratos longos que reduzem consideravelmente a fatura no final do mês. Abaixo podemos ver uma imagem que mostra os custos para diferentes tipos de abordagens para infraestrutura. O Hardware tradicional, deixa recursos funcionando sem necessidade em horários ociosos além de não oferecer flexibilidade horizontal ou vertical com relação ao uso. No entanto, podemos identificar que existe uma parte considerável que deixamos de economizar (“Large capital expenditure”) por conta da falta de adaptação com a demanda do usuário. No modelo de infraestrutura em nuvem, temos a demanda real alinhada com a infraestrutura oferecida pela AWS, então a economia ao longo do tempo tende a ser maior.

[Figura 2 – Análise de custos hardware tradicional e a nuvem]



Fonte: [AWS, 2022]

- Segurança.
    - Oferece diversos recursos para garantir a segurança das aplicações, além de proporcionar proteção interna de seus hardwares e métodos operacionais sem que seja necessária a monitoria do usuário.
-

1.1 Liderança da AWS

A computação em nuvem é oferecida por diversos provedores, mas a Amazon (Amazon Web Services), Oracle, Google (Google Cloud), IBM (IBM Cloud) e Microsoft (Microsoft Azure) são as líderes do mercado. Como mencionado anteriormente, essas empresas aproveitaram a oportunidade do mercado de tecnologia durante as evoluções da computação em nuvem e a necessidade de reestruturação da forma de oferecer serviços em computação.

Por mais que exista uma grande quantidade de empresas oferecendo os serviços com os mesmos propósitos, percebe-se que existe um provedor que vem se destacando ao longo dos anos. A Amazon Web Services, sigla AWS, desde a criação da computação em nuvem demonstrou maior capacitação em relação aos seus concorrentes. É importante mostrar como a AWS é líder no mercado e porque da escolha desse provedor para auxílio ao desenvolvimento deste projeto final. Segundo o Magic Quadrant for Cloud [Gartner, 2021] Infrastructure and Platform Services (CIPS) da Gartner, a AWS foi nomeada como líder pelo 11º ano consecutivo como o

melhor provedor em nuvem do mundo, categorizando a AWS como a plataforma mais desafiante, líder no mercado, jogadora de nicho e visionária do mercado:

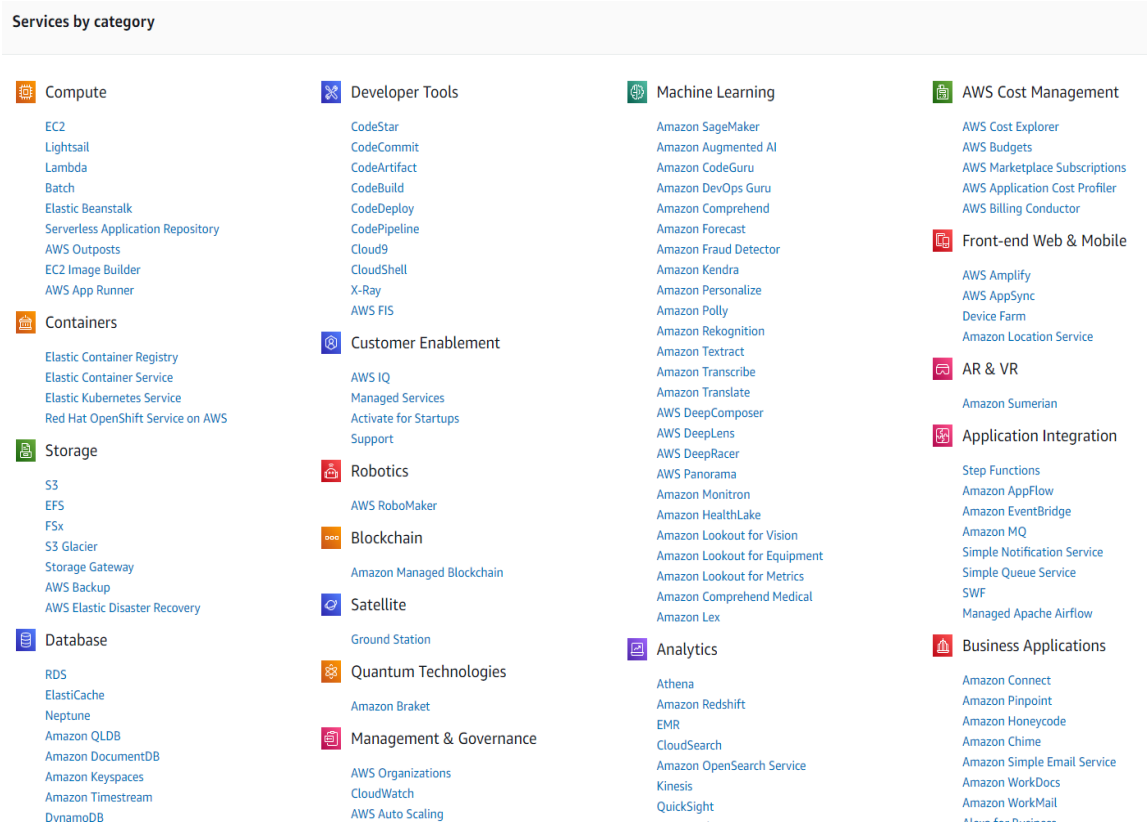
[Figura 3 – Liderança da AWS]

Fonte: [Gartner, 2021]

A escolha da AWS para este projeto final é sustentada porque esse provedor se destaca pela quantidade e qualidade dos produtos entregues, e pelo cuidado constante dos desenvolvedores da plataforma. Demonstra cuidado com seus clientes, com o apoio ao desenvolvimento de um projeto ou no aperfeiçoamento de seus ambientes próprios.

Em algumas situações o arquiteto de nuvem enfrenta dificuldades em acessar todos os recursos da mesma forma e propósito que a AWS oferece em outro provedor. No entanto, isso tende a atrair maior quantidade de usuários para utilizarem apenas a AWS por conta dessa unificação dos serviços sem utilização de vários provedores. Os diversos serviços ofertados ao usuários são voltados para diferentes temas e áreas de atuação, como exemplificado abaixo:

[Figura 4 – Serviços disponíveis na AWS]



Fonte: [AWS Console]

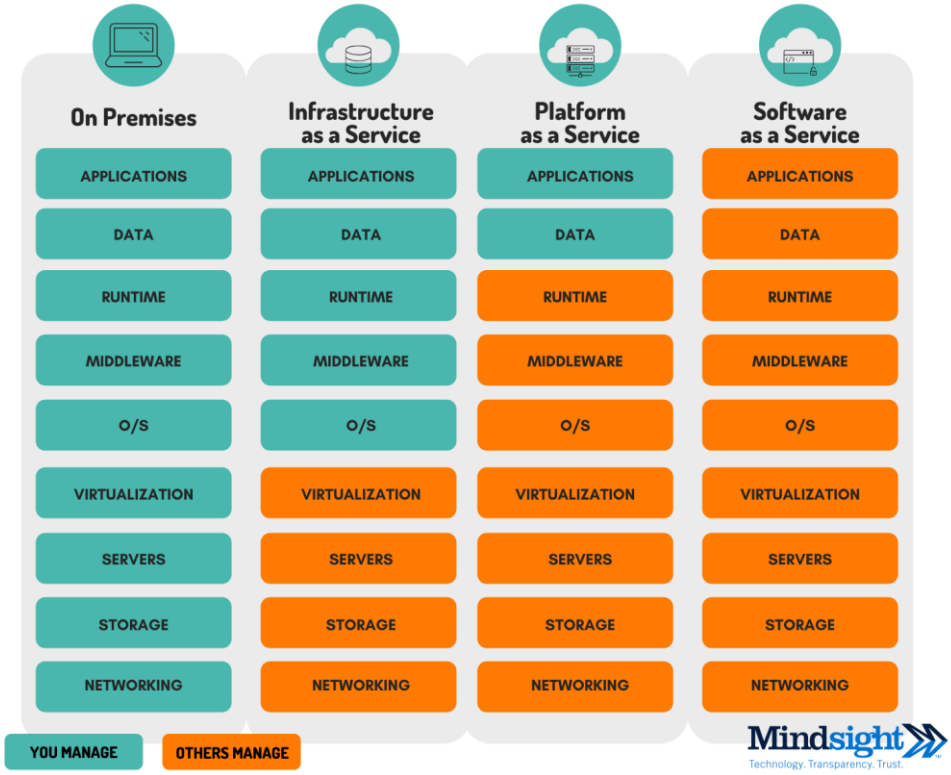
Esses serviços são divididos em subcategorias conhecidas como IAAS, PAAS e SAAS. Essa distinção ajuda para o conhecimento do usuário e também para decidir qual conjunto de serviços é o ideal para as suas necessidades:

- O IAAS, conhecido como infraestrutura como serviço, são os serviços que oferecem armazenamento, servidores, backup, entre outros. Nessa categoria de serviço o usuário não fica responsável pela virtualização, servidores, armazenamento e conexão de redes.

- O PAAS, conhecido como plataforma como serviço, são serviços que dão acesso aos provedores em Cloud no qual os usuários podem desenvolver e fornecer aplicativos. O provedor fornece infraestrutura subjacente, nesse sentido ainda existe algum controle por parte do usuário já que ele consegue ter controle sobre o código das aplicações e a base de dados que constitui essa estrutura.

- Já o SAAS (software as a service) são os serviços oferecidos como software e aplicativos por meio da Internet. Os usuários subscrevem ao software e o acessam por meio da web ou de APIs do fabricante. Pela imagem abaixo, fica evidente que os serviços SAAS não necessitam administração do usuário.

[Figura 5 - Questionário]



Fonte: [Minuto Nerd, 2020]

1.2 Multi região

A multi região, tema abordado neste projeto final, é uma das técnicas de resiliência que distribui os sistemas de computação em mais de uma região física, buscando mitigar eventuais impactos em regiões da AWS já que uma região pode ficar indisponível, seja por conta de uma falha geral de rede de uma região em que todos os recursos ficam inacessíveis ou casos em que um desastre natural afete todos os data centers de toda região.

Embora não existam cenários específicos em que uma crise possa ocorrer e quando ocorrem, todos os sistemas produtivos precisam ser desenvolvidos para que ofereçam sempre um sistema resiliente, de alta disponibilidade. A resiliência é o que influencia e determina quais empresas se destacam no mercado ao transformar as adversidades em oportunidades de crescimento, é a capacidade de se recuperar de uma crise ou interrupção e, no nível corporativo, os líderes estão percebendo que ter uma base organizacional resiliente pode funcionar em tempos incertos. Antes os empreendedores focavam na eficiência, hoje a resiliência é uma habilidade essencial para manter um negócio competitivo. Essa metodologia torna-se estrategicamente importante à medida que as empresas enfrentam instabilidades.

O maior desafio não é prever o futuro, mas lidar com o imprevisível. Dessa forma, nos próximos anos, o foco das empresas deve ser concentrar-se na construção de uma base resiliente para resistir ao que vem a seguir, e a tecnologia é essencial no processo, sendo crucial o domínio técnico dos engenheiros sobre esse assunto. A AWS desempenha um papel fundamental nesse tema já que possibilita que os usuários utilizem seus sistemas com o objetivo de evitar indisponibilidades dos sistemas, mas esse processo é uma via de mão dupla já que também depende do cuidado exclusivo dos engenheiros no desenvolvimento das arquiteturas dos ambientes em nuvem.

2. Situação Atual

Após os levantamentos, percebe-se que a AWS reporta uma ou mais indisponibilidades por ano. Temos grandes empresas sendo afetadas diretamente por esses incidentes, evidenciando que não estão aplicando as ferramentas de resiliência para multi região. De acordo com a Brand Contributor da Dell Technologies, Ana Cantu [Forbes, 2011] apresentada em 1960, “as empresas começaram a utilizar serviços em nuvem porque foram atraídas, no primeiro momento, pela redução dos custos de capital.

Com o passar dos anos, percebeu-se uma movimentação exponencial das aplicações na maioria das empresas para a nuvem, isso porque atualmente além dela oferecer otimização de custos, entrega mais segurança, confiabilidade, escalabilidade, simplicidade, entre outros benefícios.”

A partir do momento em que essas aplicações se tornaram cruciais e começaram a representar maior parte das aplicações dessas empresas, tornou-se necessário a aplicação das melhores práticas de desenvolvimento arquitetural em nuvem com relação especificamente a resiliência.

Segundo Dustin Milberg, escritor da techtarget [TechTarget,2022], “a resiliência da nuvem começa com o alinhamento estratégico com seus principais stakeholders de negócios, planejando e executando com uma arquitetura que suporta a verdadeira resiliência e tendo um programa de recuperação de desastres fortalecido como um plano de seguro crítico, mantendo seus dados seguros. Sua arquitetura técnica é essencial para as operações diárias de sua organização, tornando a nuvem não apenas um destino, mas uma jornada contínua de otimização.”

A instabilidade mais recente reportada foi iniciada durante as 11 horas da manhã do dia 7 de dezembro de 2021. Durante esse evento, diversos sistemas de várias empresas importantes ficaram indisponíveis até às 16:30 do mesmo dia, total de 5 horas e meia de indisponibilidade. Neste incidente, o serviço que ficou indisponível foi o AWS EC2, impossibilitando a criação de novas máquinas virtuais para todos os clientes da região Virgínia do norte (us-east-1), não impactando outras regiões.

[Figura 6 – Alerta de Incidentes AWS]

Fonte: [CNBC, 2021]

As aplicações mais conhecidas afetadas foram o Disney+, serviço de streaming da Disney, Netflix, Slack, Ticketmaster, aplicativo Robin Hood utilizado para negociação de ações, e Coinbase, a maior negociadora de criptomoedas dos EUA. A Disney foi uma das empresas que mais sofreram com essa instabilidade, já que a estratégia da empresa era incentivar a utilização de seus aplicativos para todos os serviços do parque, como pedidos de comida até acesso dos bilhetes para ingresso nos parques.

As interrupções demonstram o impacto que a AWS tem no mundo real e na vida cotidiana e expõem, na prática, a importância de arquiteturas desenvolvidas em multi região para que eventos como esse pudessem ser evitados, evitando prejuízos milionários para as empresas. A CNET [11] detalhou o evento de dezembro de 2021 e pontuou, por exemplo, as consequências que a Disney enfrentou na época:

Then there was the capped entry for season pass holders who had so-called Magic Keys (even those who paid for the high-end $1,400 Dream Key with no blockout dates, which led to a $5 million lawsuit revealed Thursday).

So it hasn't been an entirely magical time for Disney or its fans.”

Em contrapartida, existem empresas que se empenham para o aprimoramento de seus ambientes depois desses incidentes completos de uma região da AWS. A Netflix, por exemplo, maior empresa de streaming do mundo, refez suas arquiteturas e as transformou em multi região, evitando que próximos incidentes pudessem interromper seus sistemas. Segundo a [NetflixTechBlog, 2013], “Complete regional infrastructure outage is extremely unlikely, but our pace of change sometimes breaks critical services in a region, and we wanted to make Netflix resilient to any of the underlying dependencies. In doing so, we’re leveraging the principles of Isolation and Redundancy: a failure of any kind in one Region should not affect services running in another, a networking partitioning event should not affect quality of service in either Region.” A estratégia da Netflix foi a criação de um serviço integrado chamado Zuul, que fornece formas resilientes e robustas de direcionar o tráfego para clusters de serviço apropriados, capacidade de alterar esse roteamento em tempo de execução e capacidade de decidir se a Netflix deveria eliminar parte da carga para proteger seus serviços de serem sobrecarregados em caso de alguma falha geral de uma região:

● Capacidade de identificar e lidar com solicitações mal roteadas, quanto a comunicação com a porta 80 HTTP não está mais íntegra, enviando essas solicitações para a região correta da AWS ou retornar uma resposta de redirecionamento que direciona os clientes para a região correta.

● Capacidade de definir um nível máximo de tráfego a qualquer momento, para que quaisquer solicitações adicionais sejam automaticamente descartadas, a fim de proteger os serviços de downstream contra uma horda de solicitações. Essa capacidade é absolutamente necessária para proteger serviços que ainda estão sendo dimensionados para atender às demandas crescentes, ou quando os cachês regionais estão frios, para que a camada de persistência subjacente não fique sobrecarregada com solicitações.

Todos esses recursos fornecem para a Netflix um conjunto de ferramentas poderoso e flexível para gerenciarem como lidam com o tráfego do usuário tanto em estado estável quanto em situações de failover.

Se a Netflix sofresse recorrentes interrupções por parte da AWS antes da implementação dessa arquitetura para ambientes produtivos, correria o risco de perder assinantes se isso significasse fluxos menos confiáveis, exemplificando a importância da configuração de um ambiente usando técnicas de multi região.

A situação atual, no entanto, resume-se por grandes empresas utilizando em larga escala a computação em nuvem, primordialmente a AWS, oferecendo um serviço excepcional e migrando [BBC, 2019] diversas empresas para a sua nuvem, fazendo parte de 70% dos lucros da Amazon em seu trimestre mais recente, mas por outro lado, por mais que a Amazon faça um trabalho exemplar, temos as empresas com dificuldades na capacitação de seus funcionários e aplicação das melhores práticas de resiliência multi-região oferecidas pela Amazon.

3. Objetivos

A proposta desse projeto final consiste em enfatizar a problemática no mercado de tecnologia acerca dos cuidados para garantia de resiliência em ambientes produtivos na nuvem AWS. Visando, dessa forma, minimizar os impactos e equívocos cometidos, apresentando técnicas de melhores práticas para o desenvolvimento de

arquiteturas multi região para diferentes soluções na AWS, pautando dificuldades e impactos durante esse processo, em conjunto com a importância de mantermos essas aplicações como multi região.

Com isso, este projeto busca apresentar soluções que poderão minorar problemas decorrentes da utilização de apenas uma região na nuvem, apresentando um modelo de arquitetura multi região de um sistema de um site produtivo onde as técnicas de multi região, como a utilização do Route53 FailOverPolicy, podem ser utilizadas facilmente para qualquer ambiente produtivo, garantindo maior eficiência, evitando assim prejuízos em grande escala. A necessidade de manter uma aplicação crítica em mais de uma região poderá significar um avanço na construção de ambientes produtivos nas empresas de pequeno a grande porte.

Trata-se de um projeto no qual, para o alcance do objetivo proposto, pretendo implementar e analisar uma arquitetura multi região voltada para um caso real de uma aplicação composta por um site. Assim, consigo me basear nas melhores práticas de desenvolvimento, apresentar as dificuldades encontradas, e como a multi região diferenciou o sistema proposto caso uma das regiões tivesse sua comunicação interrompida, seja nos dois modelos, tanto na arquitetura multi região quanto na outra, sem essa metodologia implementada. Utilizarei de questionários para coleta de informações sobre a importância desse tema em nossa sociedade, códigos em YAML para construção de nossa arquitetura proposta, diagrama de fluxo para entendermos o processo de queda de uma região e códigos auxiliares em python para simularmos uma queda de uma região

Espera-se que, após o desenvolvimento desse projeto, reduza significativamente a quantidade de arquiteturas críticas na nuvem AWS sem a técnica de multi região implementada (Segundo o PagerDuty, empresa de monitoramento de incidentes [PagerDuty, 2022], um ambiente de produção é onde as versões mais recentes de software, produtos ou atualizações são enviadas ao vivo para os usuários pretendidos. Pense nisso como uma fase final de produção. Este é o ambiente onde o usuário final pode ver, experimentar e interagir com o novo produto. Todos os testes são concluídos antes deste ponto e todos os bugs são eliminados. Enquanto um ambiente de desenvolvimento pode conter várias versões diferentes de um produto ou atualização sendo trabalhado e testado, um ambiente de produção contém apenas a versão final do produto para evitar qualquer confusão ou vulnerabilidades de segurança), minimizando consideravelmente os impactos causados pela escassez da

utilização dessas técnicas, como perda completa dos dados, indisponibilidade dos sistemas da empresa, ocasionando em perda monetária exorbitante.

4. Atividades realizadas

1. Estudos preliminares

A partir do projeto final I, realizei atividades relacionadas ao tema deste projeto, focando em assuntos de multi região e o estudo de arquiteturas em Cloud. Cloud sempre fez parte do meu dia a dia, principalmente durante meus estágios onde pude aplicar os meus aprendizados da universidade e aprender mais a fundo sobre os conceitos e adquirir experiência em computação em nuvem.

- No início de 2022 realizei estudos sobre os incidentes que ocorreram ao longo dos anos na AWS e como isso impactou as empresas.
- No início de 2022 também estudei tecnicas de desenvolvimento usando o AWS CloudFormation, que possibilitou a criação automática de recursos diretamente na AWS.
- Estudo de diferentes arquiteturas no ambiente multi região, como elas são diferentes nos métodos e tecnologias e como eu poderia abrangir esse tema da melhor forma possível para que o objetivo pudesse ser transmitido para o leitor.
- No meio de 2022 comecei o desenvolvimento da arquitetura modelo e como os recursos iriam se conectar entre si. Também estudei a possibilidade de mudança automática, através do Route53, do roteamento de carga quando derrubarmos uma região.
- No meio de 2022 também construi um diagrama de fluxo do processo de queda de uma região, levando em consideração os aprendizados de INF1626 – Princípio Eng De Software.
- No final de 2022 desenvolvi o Código de simulação de queda de uma região usando os princípios de desenvolvimento de python ensinados em INF1025 – Introdução a programação e a prática adquirida nos estágios na área.
- No final de 2022 também pude finalizar os meus códigos, realizer o deploy das arquiteturas e similação da queda de uma região, exemplificando uma queda de região em cima da minha arquitetura modelo proposta, técnica aprimorada

com a leitura de documentações da AWS sobre melhores práticas de desenvolvimento dos templates de CloudFormation.

- Por fim, foquei na construção deste documento de projeto final II com o objetivo de deixar o mais claro possível para o leitor quais eram os meus objetivos na confessão deste projeto.

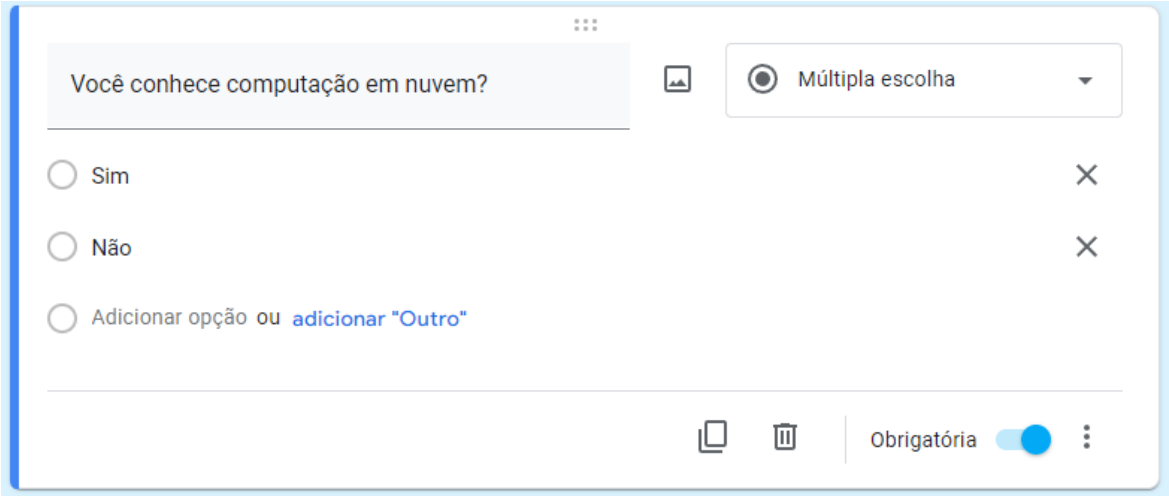
5. Implementação e avaliação

5.1 Coleta de dados / Questionário

Antes do desenvolvimento desse projeto final, foram realizadas entrevistas com usuários com o objetivo de entender melhor o conhecimento das pessoas sobre os temas abordados nesse projeto. Foram entrevistadas 49 pessoas, alunos(as) da PUC, alunos de outras universidades e cursos, além de pessoas já formadas de diferentes idades

O primeiro dado coletado foi acerca do conhecimento em computação em nuvem:

[Figura 7 - Questionário]

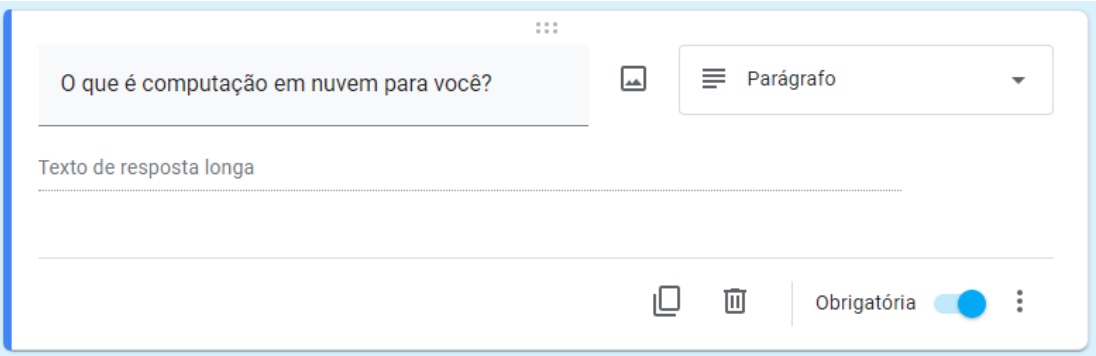


[Figura 8 - Questionário]

Podemos observar que apenas 30,6% dos entrevistados têm o conhecimento do que seja a tecnologia da computação em nuvem, evidenciando que pouca parcela da sociedade, seja da área da tecnologia ou não, conhece essa tecnologia. Isso é o retrato da situação atual que temos hoje com relação a escassez de profissionais qualificados que entendam e saibam manusear e configurar ambientes na nuvem, mencionada por Diego Duarte pela purainfo em 2016 [PURAINFO, 2016] “Em 2015, uma pesquisa feita pela Frost&Sullivan com 313 empresas brasileiras mostrou que 41% das empresas no Brasil já usam a computação na nuvem e 42% planejam investir em tecnologia. No Brasil e em todo o mundo, as empresas já sofrem com a falta de profissionais de segurança da informação qualificados. Reter esses talentos também se tornou uma tarefa árdua devido à escassez de recursos. Uma pesquisa realizada pela Cloud Security Alliance (CSA), uma organização dedicada à definição e à difusão das melhores práticas de segurança na nuvem, revelou que a falta de expertise é a maior barreira à detecção efetiva e a principal causa da perda de dados na nuvem.”

Outra informação coletada dos estudos preliminares foi uma breve descrição do que seria efetivamente a computação em nuvem, mesmo que o entrevistado não conhecesse a tecnologia:

[Figura 9 - Questionário]



- Armazenamento de dados online
- Armazenamento e processamento distribuído e flexível
- Não sei
- Lugar de armazenar dados para segurança
- Armazenamento de dados e compartilhamento.
- Armazenamento virtual
- Arquivos que posso acessar de qualquer computador ou aparelho
- Aplicações/arquivos armazenados em servidor fora do PC, sem ocupar memória (HD/SSD)
- Computação onde o processamento das informações fica externo ao computador local
- Conheço a computação em nuvem a partir de 3 conceitos: IAAS, PAAS e SAAS. Pelo que eu entendo, esses três serviços têm o objetivo de facilitar a atividade do usuário sem a necessidade de instalação de outros programas em seu computador.
- Processamento remoto de dados
- Programar em nuvem onde outros podem acessar e alterar o Código
- A computação em nuvem pode estar relacionada ao armazenamento de dados em servidores("discos não locais") ou aplicações, possibilitando o "poder de processamento" remotamente.
- Uma plataforma como o replit ou um console como o Google Stadia que permitem usar o poder computacional de uma máquina diferente da minha através da internet para executar programas, sejam eles jogos ou uma IDE
- Seria lidar com programação 100% remota, seja acesso a arquivos, documentos e etc
- Uma arquitetura/infraestrutura que não roda em máquinas on premise
- Guarda de dados em um dispositivo externo ao do usuário.
- Jogar os dados para armazenamento que não seja o interno do cellular

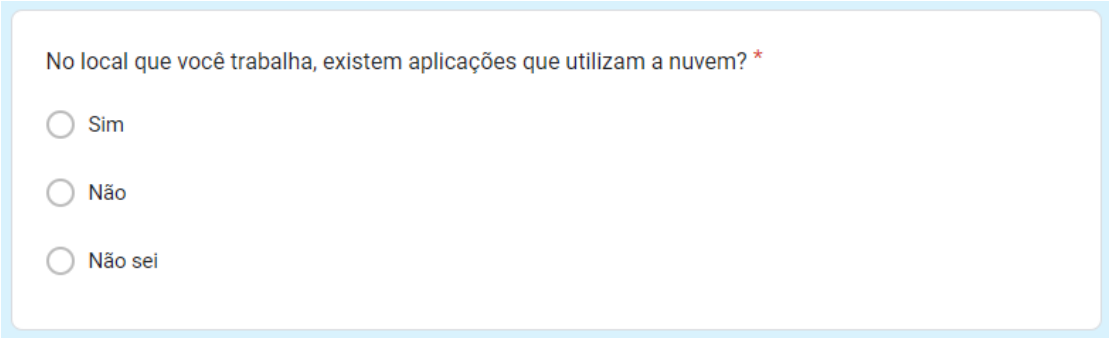
- Programação de um sistema online, que serve como uma espécie de servidor

- Utilização de recursos sob demanda de recursos, armazenamento e poder computacional
- Armazenamento
- Programação e armazenamento de dados sobre demanda.
- Um sistema que armazena dados em uma “nuvem” que é um sistema específico pra isso
- Não conheço
- Programar utilizando softwares online conforme as demandas do trabalho a ser feito no computador
- Sem conhecimento…
- Não sei mas chutaria armazenamento
- Ambiente virtual
- Uma espécie de HD virtual administrado por uma empresa, que aluga esse espaço.
- Computação remota de acesso eletrônico
- Armazenamento e recurso de sistema a distância
- Não sei
- Banco de dados guardado em nuvem e sendo utilizado on line pelos usuários
- Não tenho ideia
- O armazenamento de informações independente de hardware
- Lugar onde podemos guardar nossos documentos, informações com segurança.
- São serviços disponibilizados em servidores na internet, para processamento que antes era feito em servidores on premise.
- Sistema de armazenagem de dados
- Manter as coisas num servidor “fantasma”
- Uma maneira de programar/acessar dados sem depender de uma estrutura da própria empresa
- Uma solução remota à limitação física do computador
- Quando o programa roda em um servidor externo
- Serviços disponíveis na nuvem, armazenamento de dados
- Segurança
- Construção de programas para suportar infraestrutura de nuvem, com suporte a diversas aplicações e bases de dados.
- Salvo fora do computador/device

A partir dessa análise, pode-se concluir que, das 49 respostas recebidas, a maioria categoriza a computação em nuvem como uma tecnologia exclusiva para o armazenamento de dados. Essa confusão é comum entre as pessoas e também possivelmente para o leitor, mas a computação em nuvem vai além do armazenamento de dados já que tem como objetivo oferecer a disponibilidade sob demanda de recursos de computação, especialmente mas não exclusivamente, o armazenamento de dados e a capacidade de computação, oferecendo ao usuário serviços de Internet Of Things, Inteligência artificial, entre outros fora do escopo exclusivo do armazenamento de dados.

Outro levantamento foi com relação a utilização da computação em nuvem nas empresas atualmente:

[Figura 10 - Questionário]



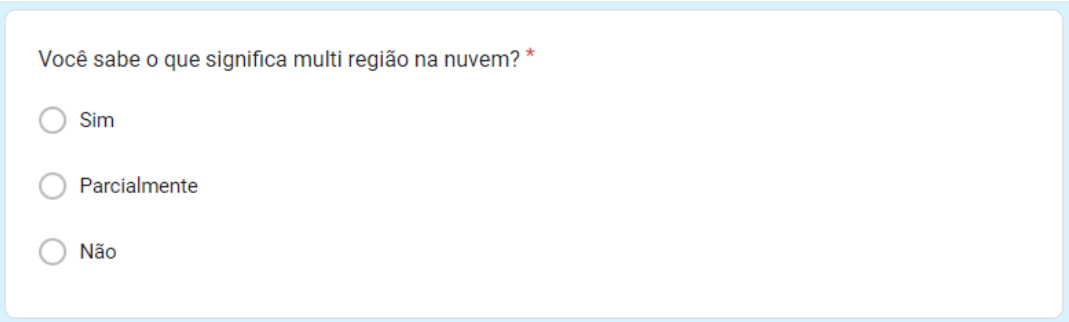
[Figura 11 - Questionário]

Fica evidente que 42,9% não utilizam a computação em nuvem para os sistemas em que o entrevistado trabalha, por mais que 57,1% aderiram a essa

tecnologia, uma porcentagem muito baixa levando em consideração seus benefícios diretos para as empresas.

Analisando a fundo sobre as práticas de utilização da computação em nuvem, fui além nas perguntas e coletei informações com relação ao conhecimento de multi região dos entrevistados, tema central desse projeto final:

[Figura 12 - Questionário]



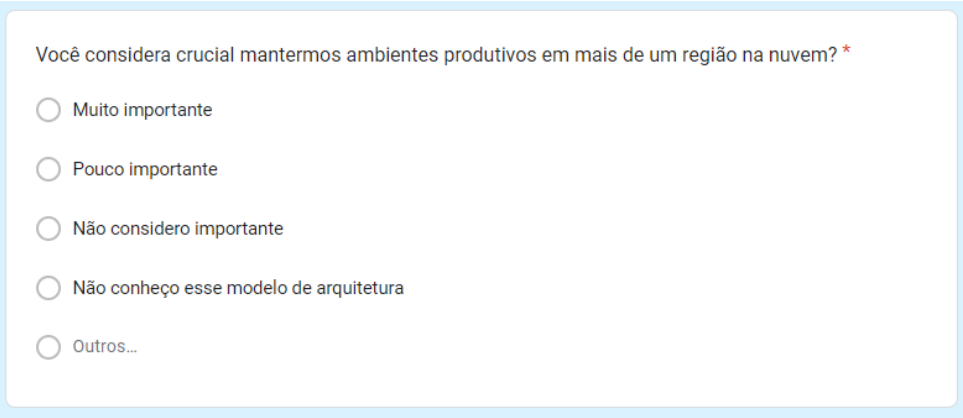
[Figura 13 - Questionário]

Apenas 4.1% efetivamente conhecem essa técnica na nuvem, o que comprova a importância de abordarmos esse tema como projeto final. É nítida a falta de compreensão dos entrevistados com relação a multi região e de sua importância para aplicações corporativas de alta criticidade.

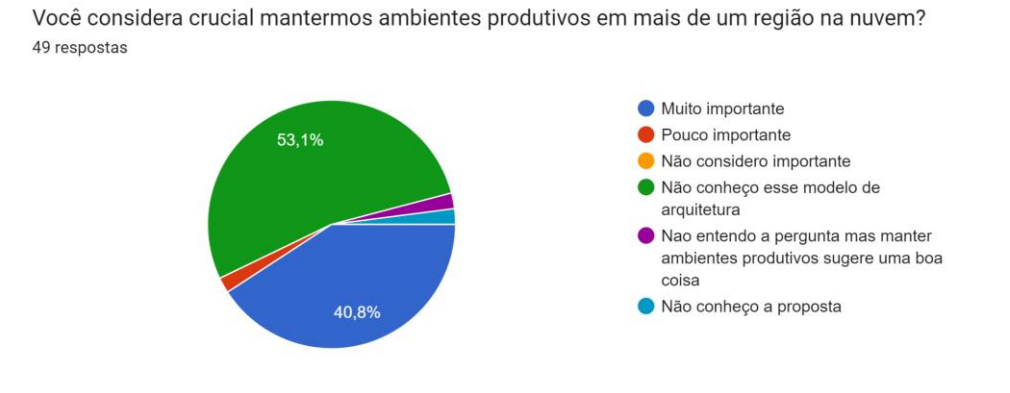
Além de coletar seus conhecimentos sobre essa tecnologia, busquei captar dos entrevistados o quanto eles consideram necessário que os ambientes produtivos

estejam configurados em mais de uma região. Apenas 40,8% consideram a multi região como modelo de arquitetura importante para ambientes produtivos:

[Figura 14 - Questionário]

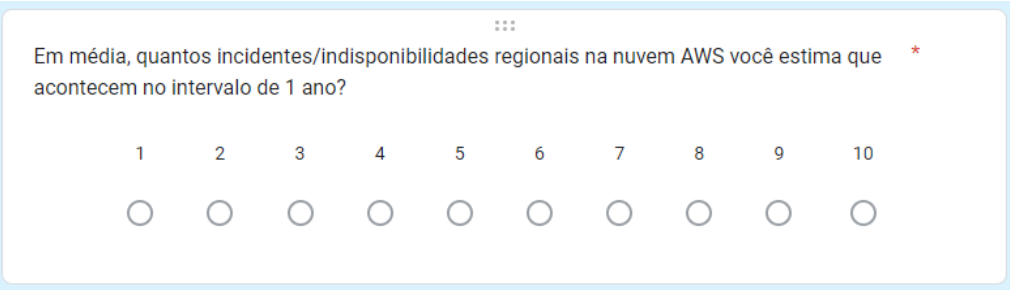


[Figura 15 - Questionário]

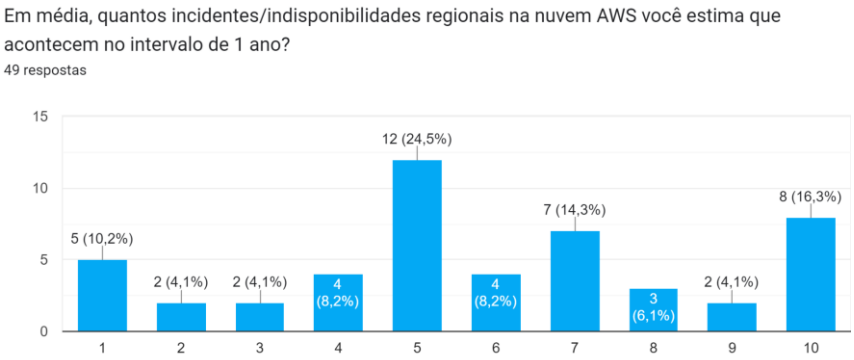


Essa importância de modelo de aplicação também está atrelada à análise prévia de risco durante o desenvolvimento e configuração de um ambiente. Para o desenvolvedor, talvez não faça sentido a utilização da multi região na AWS se a quantidade de incidentes não for considerável ao longo dos anos por parte do provedor de nuvem como a AWS. Dessa forma, o último dado coletado do usuário foi com relação ao seu conhecimento acerca da quantidade de vezes que a AWS ficou fora do ar para algumas das suas regiões. O resultado mostra que a opinião é bem dividida entre os entrevistados, já que muitos elencam como um evento frequente e outros como um evento raro, de baixa frequência:

[Figura 16 - Questionário]



[Figura 17 - Questionário]



Esse último levantamento, no entanto, também evidencia a falta de informação dos entrevistados sobre a frequência de incidentes regionais por parte da AWS (Incidentes regionais são indisponibilidades de um serviço qualquer ocasionado pela AWS que não consegue mais oferecer aquela tecnologia por algum tempo) e consequentemente influencia diretamente na sua escolha anterior sobre a importância da multi região.

Segundo a CNBC news [CNBC], notícia detalhada sobre os incidentes da AWS, a frequência dessas instabilidades, por parte da AWS, ocorre por volta de duas vezes ao ano. É muito custoso para o negócio de uma empresa quando as aplicações não estão preparadas para esse tipo de falha geral em uma região da AWS.

Nesse sentido, a partir das métricas e análises previamente estudadas podemos concluir que existe uma falta de conhecimento sobre as técnicas de multi região na nuvem e de sua importância para sistemas críticos/produtivos, e mesmo quando a computação em nuvem é utilizada os ambientes não nascem com as técnicas de multi região aplicada, é necessário que o ambiente seja impactado por uma indisponibilidade para que essas técnicas sejam efetivamente aplicadas,

evidenciando a importância da abordagem desse tema como projeto final, buscando evitar impactos gigantescos para as grandes corporações.

5.2 Análise de conceitos da Multi-Região

Antes de iniciarmos este tópico é importante apresentar para o leitor como são fisicamente estruturados os datacenters dos provedores em nuvem, com foco na AWS e sua estrutura global. Como garantia de alta disponibilidade e resiliência de seus sistemas, a AWS possui data centers espalhados por diversas regiões do mundo oferecendo até seis zonas de disponibilidades, que são localidades na mesma região.

Desta forma, o usuário tem a possibilidade de manter suas aplicações configuradas em várias regiões e/ou zonas de disponibilidades para que, em caso de falhas internas da AWS ou até mesmo indisponibilidades causadas por incidentes ambientais, a aplicação não ficaria fora do ar, impactando os usuários. Abaixo incluí uma foto da estrutura global da AWS [AWS]:

[Figura 18 – Organização global data centers AWS]
Fonte: [AWS]

A multi região, como mencionado nos tópicos anteriores, é a possibilidade de duplicarmos / replicarmos um sistema em mais de uma região, utilizando-se de técnicas no desenvolvimento de arquiteturas de aplicações. Cada serviço na nuvem AWS permite ou não a configuração em multi região e, dependendo do serviço, é fácil e rápido, enquanto que para

outros requer algumas técnicas mais complexas. Frequentemente encontramos noções incorretas sobre o conceito de disponibilidade que derivam em ações precipitadas e muitas vezes ineficientes.

Essa técnica de arquitetura para ambientes em nuvem não possui embasamento apenas por estudos e documentações apresentadas pela AWS como melhores formas de desenvolvimento de aplicações produtivas. Na prática, como mencionei acima em outros tópicos, observamos que grandes empresas têm sido afetadas por não estarem distribuindo suas aplicações em mais de uma região.

Podemos pensar nas arquitetura multi região em dois tipos diferentes, passivo-passivo e ativo-ativo. Antes da configuração de um ambiente multi regional, é importante escolher o tipo de arquitetura já que depende diretamente da característica da aplicação do usuário/empresa. Nos modelos Ativo-Ativo, temos o aproveitamento ao máximo da distribuição de carga entre todas as regiões e as trata como um ambiente homogêneo e online o tempo todo. Dessa forma, deploys, monitoramento, apontamento de rotas e distribuição de carga devem abranger todas as regiões. Neste cenário, uma falha em qualquer região na qual o sistema está deployado, teria na maior parte das vezes pouca ou nenhuma intervenção manual, considerando que as configurações deployadas ja distribuem dados e tráfego entre todas as regiões continuamente.

Poderia ser observado algum aumento de latência para usuários da região afetada, que agora tem que acessar o sistema com outra região, ou por consequência do aumento de carga geral no sistema, uma vez que parte de sua infraestrutura ficou offline. É importante testar esse tipo de cenário contantemente neste modelo, para garantir que regiões parcialmente afetadas ou que fiquem apenas com performance degradada não invalidem o modelo escolhido, piorando a performance ou afetando a sincronia entre as regiões. Um ponto contra este modelo é que ele tende a ser mais caro e mais complexo de se arquitetar e manter, pois além de uma infraestrutura muitas vezes duplicada, vários processos do ciclo de vida da aplicação também aumentam linearmente com a quantidade de regiões utilizadas, como pipeline de deploy, agentes de monitoramento, entre outros.

Nos modelos Ativo-Passivo temos a aplicação possuindo o ambiente criado em múltiplas regiões, com pipelines de deploy, monitoramento e replicação ativos em todas elas, porém o tráfego não é direcionado homogeneamente para as regiões. Este

modelo possui muitas semelhanças e, por vezes, complexidades do modelo ativo-ativo, porém com um custo geralmente reduzido, já que as regiões não utilizadas não precisam ter a mesma quantidade de infraestrutura do que a região ativa.

5.3 Aumento de custo e diferentes estratégias Multi-Região

A multi região, conceito abordado nesse projeto, é extremamente importante para ambientes produtivos como mencionado anteriormente, mas alguns pontos precisam ser considerados previamente ao desenvolvimento de um arquitetura que utiliza essas técnicas na nuvem. Existem dois tipos de abordagens, ativo/ativo e ativo/passivo. A ativo/ativo possui ambos os ambientes em diferentes regiões funcionando sempre 24/7 aguardando que haja uma falha de roteamento com a região principal. Essa estratégia tem como objetivo agilizar o processo de failover já que o outro ambiente em outra região já está disponível para ser utilizado pela estrutura. Em contrapartida, essa estratégia é muito mais custosa já que o preço seria de dois ambientes funcionando ao mesmo tempo.

A outra abordagem, conhecida como ativa/passiva, tem como objetivo manter o ambiente secundário desligado, aguardando para que sejam ligados só quando for necessário, quando houver uma indisponibilidade na região primária. Essa estratégia reduz a velocidade do failover se comprado com a estrutura ativo/ativo mas por outro lado é menos custosa já que o engenheiro não precisa pagar pelos dois ambientes funcionando ativamente todo o tempo.

Antes da escolha desses diferentes tipos de ambiente é necessário pontuar se realmente é vantajoso utilizar um sistema ativo/ativo mesmo sabendo que os gastos são maiores. Vai depender do tipo de aplicação, já que existem aplicações que necessitam de uma alta disponibilidade em que segundos perdidos podem resultar em perdas monetárias enormes em caso de algum failover e existem sistemas produtivos que mesmo em um failover aceitam esperar mais tempo durante um failover já que dessa forma conseguem atingir um economia considerável no seu ambiente no longo prazo.

5.4 Implementação de arquitetura no modelo multi-região

Existem várias arquiteturas que são desenvolvidas levando em consideração a estratégia da multi região. Como mencionado anteriormente, nos outros capítulos, a

AWS fornece diferentes serviços que podemos configura-los com multi região nativamente ou em alguns casos realizar modificações para que esse modelo de arquitetura funcione corretamente.

Atualmente, diversas empresas possuem pelo menos um “website” na sua estrutura produtiva. Neste sentido, além de apresentar neste tópico as arquiteturas de referência e comentar os pontos sobre elas na perspectiva de multi região, pretendo focalizar os testes para demonstração de conceito nas aplicações que são hospedadas em sites, em sua maioria estáticos. Essa escolha foi ponderada pela praticidade de exemplificar ao leitor os problemas ocasionados durante uma falha geral de uma região na AWS.

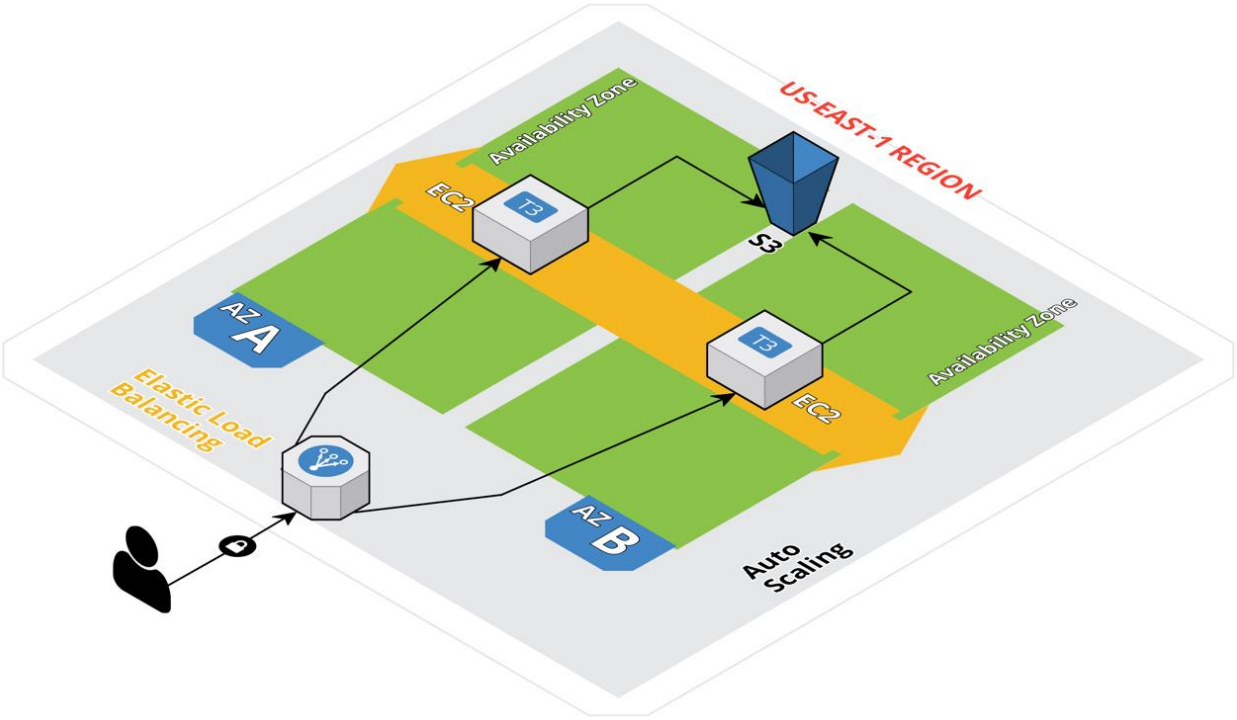
Apresentarei essa arquitetura em dois modelos, configurado com multi região e sem multi região, dessa forma consigo explicar as diferenças entre as duas estruturas. A arquitetura fora do modelo de multi região, será composta por um grupo de máquinas virtuais (AWS EC2), conectadas a um load balancer e ao Route53, serviço de DNS da Amazon. Esses três recursos funcionam da seguinte forma:

- Inicialmente temos um AutoScaling, serviço da Amazon que permite escalarmos máquinas sempre que necessário e reduzirmos a carga sempre automaticamente assim que tivermos a redução no uso. Esse serviço permite que possamos determinar a quantidade de máquinas desejadas, mínimo e máximo que o AutoScaling deve seguir como configuração. Dessa forma, pensando no dia a dia de uma aplicação, teremos o autoscaling mantendo o número de máquinas no valor escolhido como desejável, e quando tivermos um aumento nas conexões ao site ou qualquer outra aplicação, teríamos o aumento automático na quantidade de máquinas / poder computacional até o valor máximo que estipulamos para o número de máquinas do autoscaling. Em contrapartida, se o ambiente sofrer uma redução na quantidade de acessos aos site, o autoscaling busca reduzir a quantidade de máquinas baseado na configuração determinada.

- Outro serviço utilizado foi o ElasticLoadBalancer conectado ao AutoScaling, serviço da AWS que permite a orquestração de carga para diferentes subregiões da AWS. Como pontuado nos últimos capítulos, onde foi explicado sobre a estrutura global da AWS, essas subdivisões são

bifurcações dentro de uma mesma região (conhecidas como zonas de disponibilidade). Nesse sentido, o LoadBalancer funciona com o objetivo de que se tivessemos uma queda da zona de disponibilidade dentro de uma mesma região, teríamos o tráfego sendo direcionado para a outra zona sem que tivéssemos impacto, já que o LoadBalancer consegue fazer essa transferência de comunicação sem maiores dificuldades.

[Figura 19 – Arquitetura sem multi região]



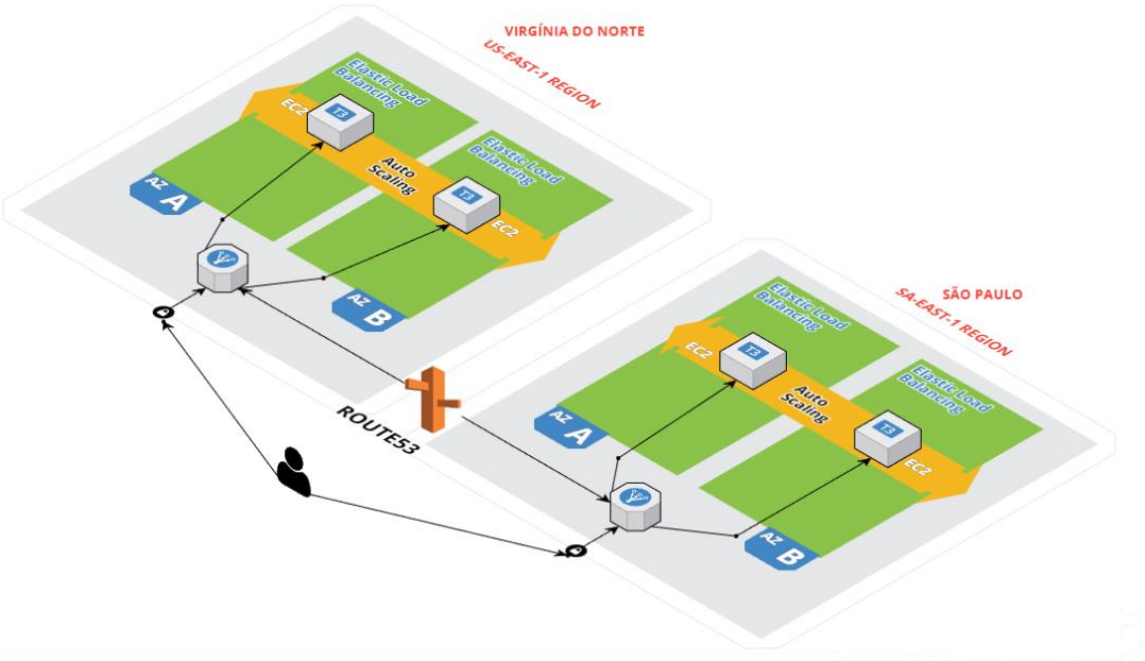
A arquitetura desse ambiente é a estrutura descrita acima e, analisando esse modelo de arquitetura, temos o site sendo hospedado pelas máquinas virtuais EC2 e seu código armazenado dentro de um bucket S3. O bucket S3 é um serviço de armazenamento de arquivos, exatamente igual ao google drive, serviço que permite o armazenamento simples de arquivos. Utilizei o S3 nessa arquitetura unicamente para armazenar algum dado relevante, como o código em HTML do site estático que pretendo utilizar como teste mas na arquitetura multi região proposta abaixo, não utilizaremos do bucket S3 para simplificar o entendimento e porque será apresentado um site estático e sem dados que precisam necessariamente serem duplicados entre as regiões, mas é um serviço que serve com que os dados armazenados no site sejam consultados em um único local por ambas as regiões.

Dessa forma, se tivermos uma falha total de uma região como a virgínia do norte (us-east-1), teríamos um interrupção completa do site, já que toda a estrutura da aplicação esta sendo sediada em apenas uma região.

Um ponto positivo nessa estrutura é que se alguma das subdivisões de um região (zona de disponibilidade) estiver fora do ar por conta da AWS, não teríamos impacto direto porque, como mencionado anteriormente, o load balancer conseguiria transferir a comunicação para a outra zona de disponibilidade sem que houvesse indisponibilidade, mas isso não aconteceria nessa arquitetura caso uma região fique indisponível.

Em contra partida, uma arquitetura multi região dessa estrutura seria dessa forma:

[Figura 20 -Arquitetura com multi região]



Essa nova arquitetura, agora utilizando as melhores práticas de desenvolvimento em multi região, difere em alguns pontos da arquitetura anteriormente apresentada:

- Nesse caso temos duas estruturas idênticas mas em duas regiões diferentes (identificadas em vermelho na arquitetura). Neste sentido,

fica possível refletir a mesma arquitetura para uma outra estrutura apartada mas com as mesmas características.

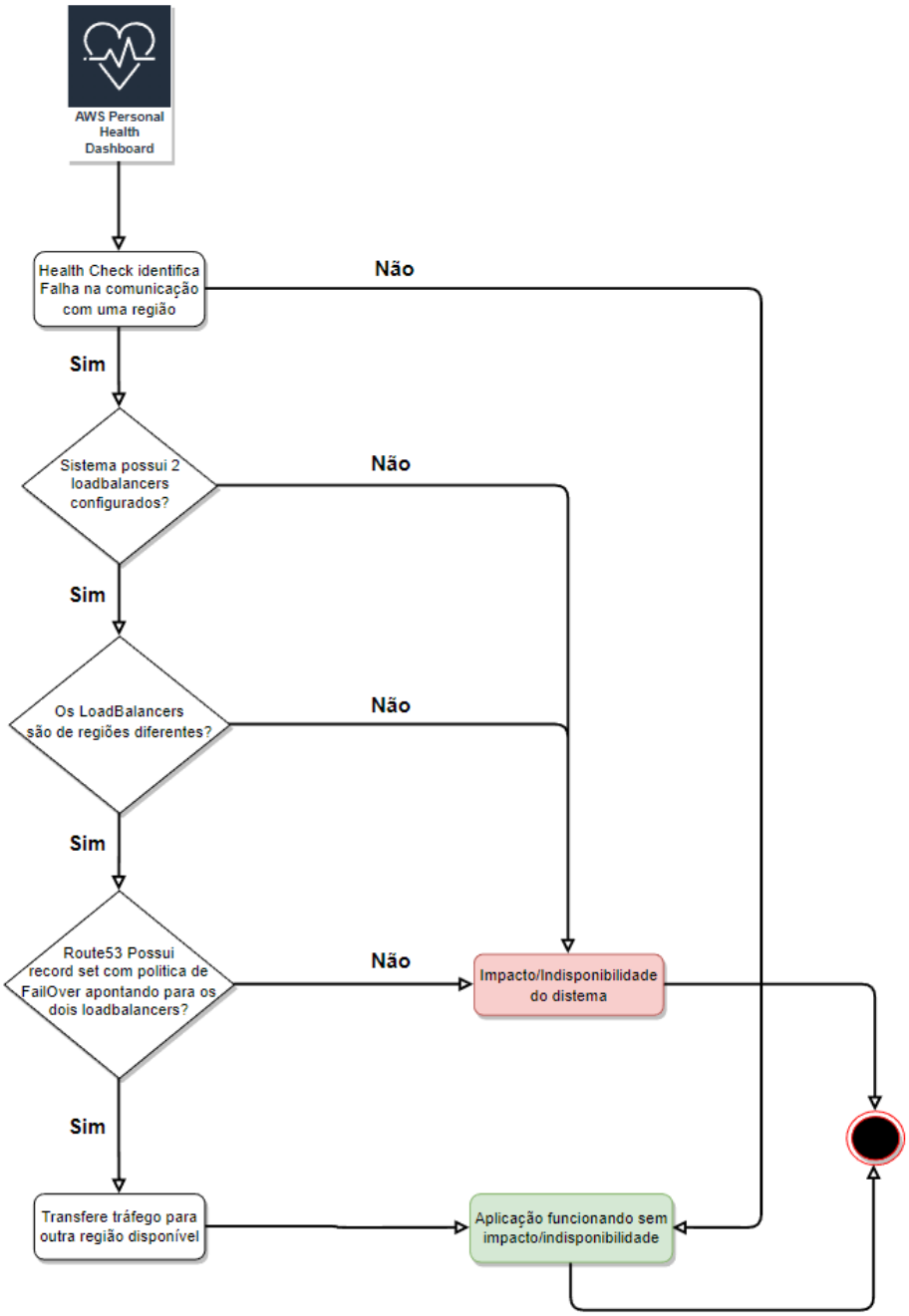
- A segunda mudança foi a inclusão de um novo serviço gerenciado da AWS, o Route53. Esse serviço é um serviço de Rede (DNS) que permite conectarmos o endpoint de algum serviço, como uma máquina virtual, convertendo nomes de sites para endereços IP como 192.0.2.1, para se conectarem entre si. Ele permite a configuração de algumas políticas de roteamento de tráfego e sua diferença com relação ao load balancer é que permite orquestração de rotas para diferentes regiões, enquanto que o loadbalacer aceita apenas conexão com diferentes zonas de disponibilidade.

- A terceira mudança introduzida na arquitetura em relação a arquitetura anterior, é a utilização do serviço Route 53 da AWS. Esse serviço é um serviço de DNS que permite a tradução de IPs para domínios. Além da simples configuração de um domínio, é possível criar políticas de roteamento de carga a partir de um evento. Segundo a própria AWS, as políticas mias importantes são:
    - Simple Routing Policy: Permite associar um servidor ao nome do domínio, por exemplo, uma máquina virtual EC2 com o site www.vendas.com.br
    - Geolocation Routing Policy: Permite direcionar o tráfego a partir da região do usuário. Essa política normalmente é utilizada para garantir que o usuário em determinada região não acesse conteúdos permitidos em apenas uma região específica.
    - Geoproximity Routing Policy: Essa política permite direcionar o tráfego a partir da localização do usuário. Essa política, faria sentido por exemplo, em uma aplicação que fosse utilizada por pessoas de diferentes localizações do mundo, garantindo que o usuário não enfrente latencia.
    - Latency Routing Policy: Garante a conexão do usuário com o ambiente de menos latência. Nesse sentido, quando um ambiente estiver passando por uma maior quantidade de acessos e um aumento na latência, o tráfego é direcionado para o outro link.

- Failover Routing Policy: Essa política permite a mudança de tráfego quando um evento de indisponibilidade de um dos ambientes ficar fora do ar, quando não é possível mais efetuar uma conexão. Optei por essa política para a estrutura multi região porque quando uma região ficar indisponível a comunicação dos usuários é automaticamente cortada e o tráfego é direcionado para a região saudável e funcional. Além disso, escolhi essa política por conta da facilidade de associar loadbalancers nas políticas, permitindo que o Route53 fique mapeando diretamente se o serviço esta ou não ativo.

O LoadBalancer roteia solicitações apenas para as instâncias íntegras. Quando o load balancer determina que uma instância ou LoadBalancer não está íntegra, ele interrompe o roteamento de solicitações para essa instância. Quando uma região fica indisponível pela AWS, a comunicação de rede para essa região não funciona mais por um período indeterminado de tempo, significando que o health check do load balancer não consegue mais identificar que aquela(as) instâncias estão íntegras para comunicação, então ele decide interromper a conexão para essas instâncias. O diagrama de fluxo dessa arquitetura foi desenvolvimento abaixo:

[Figura 21 – Diagrama de fluxo]



Para aplicar na AWS essa arquitetura multi região, inicialmente desenvolvi alguns códigos usando um formato de serialização de dados legíveis (YAML) criado a partir de linguagens de programação como Python e C com o intuito de criar um processo predefinido para criação automática de todos os recursos que serão utilizados para essa estrutura.

Esses códigos vão gerar o que são chamados de templates de CloudFormation que, depois de compilados, são transformados em stacks de cloudformation, serviço da AWS que fornece a possibilidade de descrever e provisionar todos os recursos de

infraestrutura nos ambientes de maneira segura e repetível sem a necessidade da criação dos recursos e suas configurações manualmente pelo console da AWS. Dessa forma, criei um código que pode ser utilizado por direferentes usuários em várias contas, sem necessidade de alteração. Abaixo inclui os templates de CloudFormation que desenvolvi para este modelo de arquitetura:

- São Paulo (sa-east-1):

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Esse template foi desenvolvido para criacao
  de uma arquitetura multi regiao na nuvem AWS. Essa arquitetura
  hospeda uma aplicacao web usando EC2, LoadBalancers,
  AutoScaling e o Route53 com FailOver Policy para roteamento
  de carga para outra regiao caso a principal fique inacessivel
  por alguma indisponibilidade por parte da AWS'
Resources:
  SecurityGroupForSaoPaulo:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access via port 22, http and https access
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: '177.208.20.125/32'
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: '0.0.0.0/0'
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: '0.0.0.0/0'
      VpcId: 'vpc-b25f7cd5'
  LaunchTemplateSaoPaulo:
    Type: AWS::EC2::LaunchTemplate
    Properties:
```

```
    LaunchTemplateName: !Sub ${AWS::StackName}-launch-template
    LaunchTemplateData:
      InstanceType: 't3.micro'
      SecurityGroupIds: [!Ref SecurityGroupForSaoPaulo]
      KeyName: 'projetofinal_saopaulo'
      ImageId: 'ami-0895310529c333a0c'
      BlockDeviceMappings:
      - DeviceName: "/dev/sda1"
        Ebs:
          DeleteOnTermination: 'true'
          VolumeSize: '8'
          VolumeType: 'gp2'
      UserData:
        Fn::Base64:
          !Sub |
            #!/bin/bash
            sudo su
            yum install -y httpd.x86_64
            systemctl start httpd.service
            systemctl enable httpd.service
            echo "Ola! Estamos em SaoPaulo" > /var/www/html/index.html
  ALBForSaoPaulo:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      IpAddressType: 'ipv4'
      Name: 'ALBForSaoPaulo'
      Scheme: 'internet-facing'
      SecurityGroups:
        - !Ref SecurityGroupForSaoPaulo
      Subnets:
        - 'subnet-ca974bac'
        - 'subnet-421af30b'
      Type: 'application'
  LoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
```

```
    - Type: forward
      TargetGroupArn:
        Ref: TargetGroup
    LoadBalancerArn: !Sub '${ALBForSaoPaulo}'
    Port: 80
    Protocol: HTTP
TargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: 'SaoPauloTargetGroup'
    Port: 80
    Protocol: HTTP
    VpcId: 'vpc-b25f7cd5'
    TargetType: 'instance'
  DependsOn:
  - ALBForSaoPaulo
AutoScalingForSaoPaulo:
  Type: AWS::AutoScaling::AutoScalingGroup
  Properties:
    LaunchTemplate:
      LaunchTemplateId: !Ref LaunchTemplateSaoPaulo
      Version: !GetAtt LaunchTemplateSaoPaulo.LatestVersionNumber
    AvailabilityZones:
      - 'sa-east-1a'
      - 'sa-east-1b'
    MaxSize: '2'
    MinSize: '1'
    TargetGroupARNs: [!Ref TargetGroup]
    VPCZoneIdentifier:
      - 'subnet-ca974bac'
      - 'subnet-421af30b'
RecordSetSaoPaulo:
  Type: AWS::Route53::RecordSet
  Properties:
    SetIdentifier: 'www-Secondary'
    AliasTarget:
      HostedZoneId: !GetAtt 'ALBForSaoPaulo.CanonicalHostedZoneID'
```

```
      DNSName: !GetAtt 'ALBForSaoPaulo.DNSName'
    Failover: 'SECONDARY'
    HostedZoneId: 'Z0537674NQBQ3V3SJXLA'
    Name: 'www.rafaweb.link.'
    Type: 'A'
```

- Virgínia do Norte (us-east-1):

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Esse template foi desenvolvido para criacao
  de uma arquitetura multi regiao na nuvem AWS. Essa arquitetura
  hospeda uma aplicacao web usando EC2, LoadBalancers,
  AutoScaling e o Route53 com FailOver Policy para roteamento
  de carga para outra regiao caso a principal fique inacessivel
  por alguma indisponibilidade por parte da AWS'
Resources:
  SecurityGroupForVirginia:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access via port 22, http and https access
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: '177.208.20.125/32'
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: '0.0.0.0/0'
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: '0.0.0.0/0'
      VpcId: 'vpc-f3107889'
  LaunchTemplateVirginia:
    Type: AWS::EC2::LaunchTemplate
```

```
  Properties:
    LaunchTemplateName: !Sub ${AWS::StackName}-launch-template
    LaunchTemplateData:
      InstanceType: 't3.micro'
      SecurityGroupIds: [!Ref SecurityGroupForVirginia]
      KeyName: 'projetofinal'
      ImageId: 'ami-026b57f3c383c2eec'
      BlockDeviceMappings:
      - DeviceName: "/dev/sda1"
        Ebs:
          DeleteOnTermination: 'true'
          VolumeSize: '8'
          VolumeType: 'gp2'
      UserData:
        Fn::Base64:
          !Sub |
            #!/bin/bash
            sudo su
            yum install -y httpd.x86_64
            systemctl start httpd.service
            systemctl enable httpd.service
            echo "Ola! Estamos em Virginia" > /var/www/html/index.html
  ALBForVirginia:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      IpAddressType: 'ipv4'
      Name: 'ALBForVirginia'
      Scheme: 'internet-facing'
      SecurityGroups:
        - !Ref SecurityGroupForVirginia
      Subnets:
        - 'subnet-0a06ea47'
        - 'subnet-5bacf507'
      Type: 'application'
  LoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
```

```
    DefaultActions:
    - Type: forward
      TargetGroupArn:
        Ref: TargetGroup
    LoadBalancerArn: !Sub '${ALBForVirginia}'
    Port: 80
    Protocol: HTTP
TargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: 'VirginiaTargetGroup'
    Port: 80
    Protocol: HTTP
    VpcId: 'vpc-f3107889'
    TargetType: 'instance'
  DependsOn:
  - ALBForVirginia
AutoScalingForVirginia:
  Type: AWS::AutoScaling::AutoScalingGroup
  Properties:
    LaunchTemplate:
      LaunchTemplateId: !Ref LaunchTemplateVirginia
      Version: !GetAtt LaunchTemplateVirginia.LatestVersionNumber
    AvailabilityZones:
      - 'us-east-1d'
      - 'us-east-1a'
    MaxSize: '2'
    MinSize: '1'
    TargetGroupARNs: [!Ref TargetGroup]
    VPCZoneIdentifier:
      - 'subnet-0a06ea47'
      - 'subnet-5bacf507'
HealthCheckVirginia:
  Type: AWS::Route53::HealthCheck
  Properties:
    HealthCheckConfig:
      Port: 80
```

```
      Type: HTTP
      FullyQualifiedDomainName: !GetAtt ALBForVirginia.DNSName
  RecordSetVirginia:
    Type: AWS::Route53::RecordSet
    Properties:
      SetIdentifier: 'www-Primary'
      AliasTarget:
        HostedZoneId: !GetAtt 'ALBForVirginia.CanonicalHostedZoneID'
        DNSName: !GetAtt 'ALBForVirginia.DNSName'
      Failover: 'PRIMARY'
      HealthCheckId: !Ref HealthCheckVirginia
      HostedZoneId: 'Z0537674NQBQ3V3SJXLA'
      Name: 'www.rafaweb.link.'
      Type: 'A'
```

Abaixo coloquei imagens durante o deploy dessas stacks de Cloudformation e pretendo mostrar as configurações, o que foi efetivamente criado e como a aplicação se comportou durante a queda de uma região na nuvem com o objetivo de testar essa indisponibilidade, ponto central deste projeto final.

O processo de deploy da arquitetura foi iniciado automaticamente através de um código desenvolvido em python que possibilitou que os deploys acontecessem de forma automatizada sem a necessidade de deploy manual na console da AWS:

[Figura 22 - Código de simulação de queda da região]

```python
import boto3
import os

def LoginAWSAndCreateStack():
    regions = ['us-east-1', 'sa-east-1']
    cloudformation_us_client = boto3.client('cloudformation', region_name='us-east-1')
    cloudformation_sa_client = boto3.client('cloudformation', region_name='sa-east-1')
    with open('ProjetoFinalUSA.yaml', "r") as f:
        try:
            cloudformation_us_client.create_stack(
                StackName='ProjetoFinalUSA',
                TemplateBody=f.read(),
            )
        except Exception as error:
            print('Unable to create cloudformation stack for USA: {}'.format(error))
    with open('ProjetoFinalBRA.yaml', "r") as f:
        try:
            cloudformation_sa_client.create_stack(
                StackName='ProjetoFinalBRA',
                TemplateBody=f.read(),
            )
        except Exception as error:
            print('Unable to create cloudformation stack for BRA: {}'.format(error))

```

Esse código utiliza as APIs da AWS para criação (deploy) das duas stacks de cloud formation na AWS, apresentadas anteriormente. Inicialmente, foi necessário fazer o login na AWS para conseguir utilizar as APIs da Amazon. No entando, criei um usuário no serviço de controle de acesso chamado de IAM, e solicitei manualmente a criação de credenciais para acesso programático na AWS. Dessa forma, conseguimos acessar as APIs que pretendemos utilizar, se o usuário possui as permissões necessárias. O processo para configuração do usuário localmente, consiste em abrir um terminal do powershell e digitar o comando abaixo, passando o profile/usuário criado no AWS IAM e posteriormente as chaves de acesso fornecidas pela AWS, a Access Key ID e Secret Access Key:

[Figura 23 – Configuração de credenciais AWS]

```
PS C:\Users\rafae> aws configure --profile pinara
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

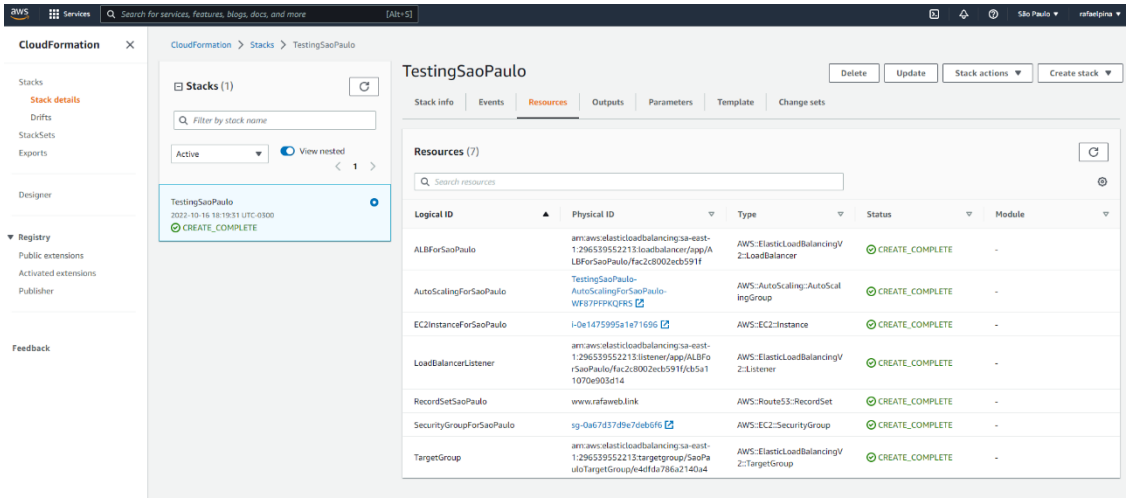
Posteriormente, já é possível iniciar a execução do código de deploy das stacks de cloud formation. Analisando um pouco mais o código apresentado, as linhas 6 e 7 servem para instanciarmos os clientes do serviço de cloud formation para cada região da AWS, já que todos esses serviços, quando modificados por APIs, precisam de um

cliente ativo para que as consultas funcionem corretamente. Da linha 9 até a 15, utilizo a funcionalidade do python de leitura de arquivos, o “with open”, para acessar o template de cloudformation desenvolvido para a região do norte da virgínia, nos Estados Unidos. Vale reforçar também que, como mencionado anteriormente, precisamos de dois ambientes iguais funcionando em diferentes regiões para que possamos simular corretamente esse cenário, por isso que temos dois cloud formation templates diferentes. Finalmente, da linha 16 até a linha 23, temos o mesmo processo de leitura de arquivo, mas agora do segundo template para a região de São Paulo, no Brasil. Depois disso, inicia-se a criação das stack, utilizando a API de create_stack(). Sua utilização é documentada pela AWS aqui [AWS, 2022]

Depois de rodarmos o código anterior, já conseguimos verificar diretamente no console da AWS, nas duas regiões, que as Stacks de CloudFormation foram criadas com sucesso e que todos os recursos especificados no template foram criados.

Para Virgínia do Norte, temos a criação de 1 autoscaling, 1 loadbalancer, 1 healthcheck, 1 target group que possibilita o agrupamento das máquinas virtuais e faz a conexão entre o loadbalancer e o autoscaling do sistema e 1 recordset do Route53, permitindo o direcionamento de tráfego em caso de falha desse loadbalancer criado nessa região:

[Figura 24 – Criação do CloudFormation]



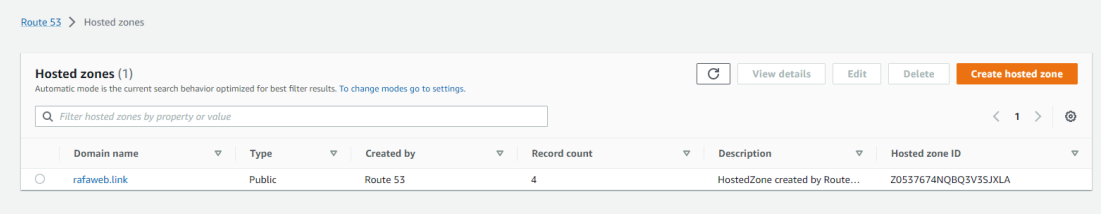
Para São Paulo, também temos a criação de 1 autoscaling, 1 loadbalancer, 1 target group que possibilita o agrupamento das máquinas virtuais e faz a conexão entre o loadbalancer, o autoscaling do sistema e 1 recordset do Route53, permitindo o direcionamento de tráfego em caso de falha desse loadbalancer criado nessa região.

Os dois ambientes seguem a mesma estrutura para garantia de similaridade na transferência de conexão em caso de indisponibilidade da AWS, mas o ambiente de São Paulo difere do de Virgínia porque não realiza a criação de HealthChecks já que funciona, nesse caso, como o ambiente secundário. Dessa forma, como o health check só está monitorando o LoadBalancer de Virgínia, se ocorrer uma indisponibilidade na região da Virgínia, teríamos o tráfego sendo direcionado para a outra região, assim não torna-se necessária a criação de um health check para cada região, reduzindo custos e erros na arquitetura proposta.

Analisando detalhadamente a estrutura que foi efetivamente criada, listada acima pelo cloudformation, temos:

- AWS Route53
    - Antes da criação dos recursos do Route53, foi necessária a compra de um domínio genérico através do console da AWS para sediarmos a aplicação do site que pretendo testar, já que dessa forma conseguimos simular uma situação real de uma aplicação estática, como um site. O domínio (HostedZone) comprado foi o www.rafalink.web:

[Figura 25 – HostedZone adquirida para simulação da aplicação]



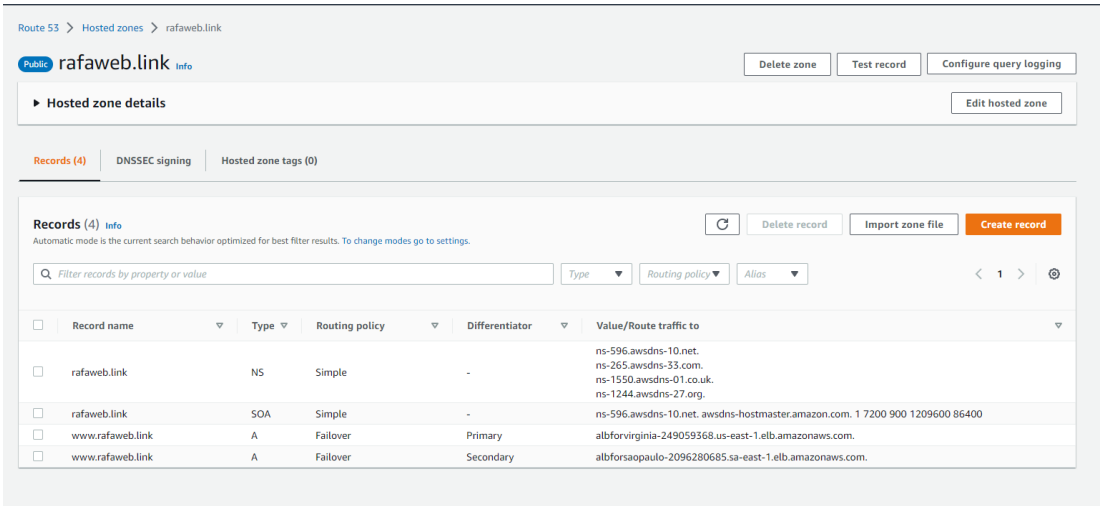
    - Na imagem abaixo temos a página da AWS do Route53 e nele aparecem quatro record sets. Os record sets, segundo a própria [AWS, 2022], “são registros para informar ao Domain Name System (DNS) como deseja que o tráfego seja roteado para esse domínio. Por exemplo, você pode criar registros que fazem com que o DNS faça o seguinte:

        - Roteie tráfego de Internet de exemplo.com para o endereço IP de um host no seu datacenter.
        - Roteie e-mail desse domínio (ichiro@exemplo.com) para um servidor de e-mail (mail.exemplo.com).

- Roteie o tráfego de um subdomínio chamado operacoes.toquio.exemplo.com para o endereço IP de um host diferente.”

- Os dois primeiros record sets são criados automaticamente pela AWS logo após a criação da HostedZone (domínio obtido). Eles são do tipo NS e SOA e servem para conectarmos o domínio com a AWS caso ele fosse adquirido por algum outro provedor administrador de domínios, como a Google. No entanto, em nossa arquitetura multi região, não será necessária a utilização desses dois record sets porque o domínio foi comprado diretamente pela AWS que disponibiliza essa opção de compra através do Route53. Os últimos dois record sets criados são record sets que apontam para os distribuidores de cargas, os load balancers. Um deles, chamado de primário, foi configurado para apontar para o loadbalancer da região da virgínia em conjunto com um Health Check, onde vamos observar caso exista alguma indisponibilidade. O secundário aponta para o load balancer da outra região, por isso que é categorizado na configuração como “secondary”.

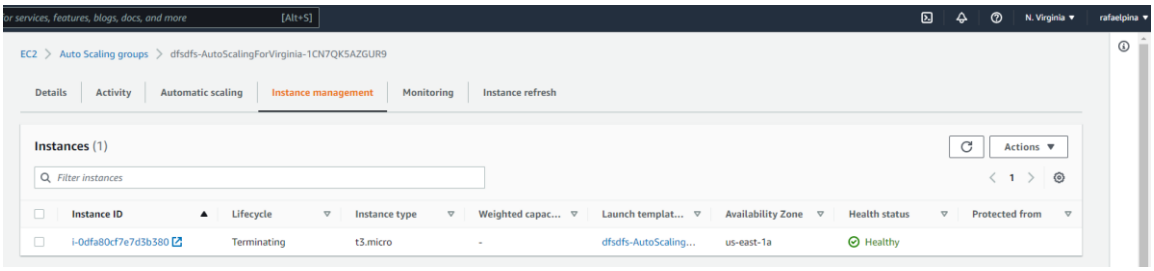
[Figura 26 - RecordSets]



- Além da configuração dos record sets com a política de FailOver apontando para os dois loadbalancers em diferentes regiões, também foi configurado o HealthCheck do Route53. O Health Check serve para verificar de 30 em 30 segundos se existe comunicação com o

loadbalancer na porta 80 (HTTP). Nesse sentido, quando a região do LoadBalancer conectado ao HealthCheck, no nosso caso o primário de Virgínia, o HealthCheck muda o seu status para Unhealthy. Inicialmente, após o deploy dos templates o status é healthy:

[Figura 27 – AWSEC2 AutoScaling Instances]



- AWS AutoScaling Group

O AutoScaling nesse arquitetura tem o objetivo de orquestrar o balanceamento de carga. Nesse sentido, sempre que o site receber uma quantidade elevadas de acessos, teríamos um aumento automático e horizontal na quantidade de máquinas em atuação e, por outro lado, caso tenhamos uma redução na quantidade de acessos, teríamos a redução no poder computacional. O AutoScaling, quando especificado no template de cloudformation acima, permite a definição de algumas informações importantes que ditam como esse serviço deve atuar nesses diferentes cenários:

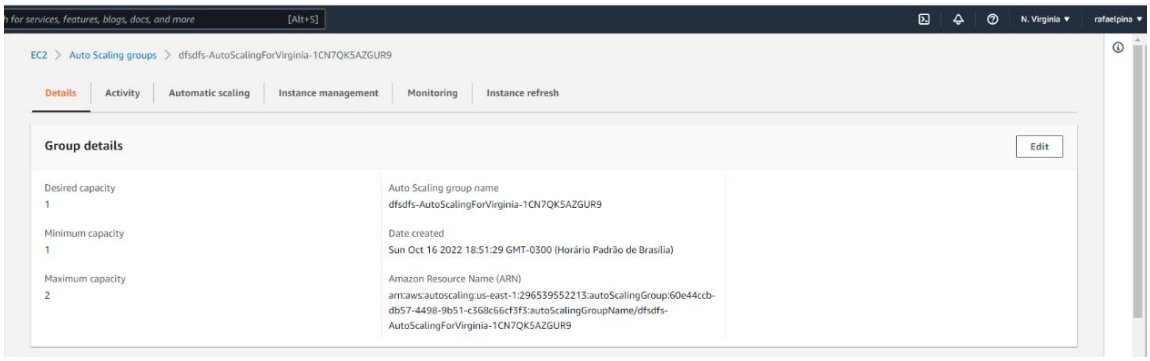
[Figura 28 – Código criação do LaunchTemplate]

```
AutoScalingForVirginia:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      LaunchTemplate:
        LaunchTemplateId: !Ref LaunchTemplateVirginia
        Version: !GetAtt LaunchTemplateVirginia.LatestVersionNumber
      AvailabilityZones:
        - 'us-east-1d'
        - 'us-east-1a'
      MaxSize: '2'
      MinSize: '1'
      TargetGroupARNs: [!Ref TargetGroup]
      VPCZoneIdentifier:
        - 'subnet-0a06ea47'
        - 'subnet-5bacf507'

```

Em verde temos dois parâmetros que definem como o ambiente vai escalar ou reduzir ao longo do tempo dependendo da utilização, por mais que essa metodologia seja indiferente com relação a multi região, mas é relevante para simulação de um cenário de ambiente produtivo que, em sua maioria requer um AutoScaling para escala automática do ambiente. O parâmetro MaxSize define o máximo de máquinas que o AutoScaling pode criar automaticamente em resposta a um aumento de carga. Dessa forma, caso haja um aumento na utilização do site e o AutoScaling julgar necesária a criação de mais máquinas para aguentar a carga recebida, ele não poderá, por definição do ambiente, criar mais de duas máquinas em sua totalidade. Por outro lado, no momento de redução de máquinas em resposta a um evento de baixa utilização do site, o AutoScaling não poderá reduzir mais que uma máquina já que em nossa definição foi escolhido mantermos em uma máquina como mínimo desejável, evitando que o AutoScaling pare de funcionar caso estivesse com mínimo de uma máquina. Depois do deploy dos templates de Cloud Formation, esse foi o AutoScaling criado onde conseguimos ver que o valor máximo e mínimo configurado no template foi respeitado:

[Figura 29 – Configurações do AutoScaling]

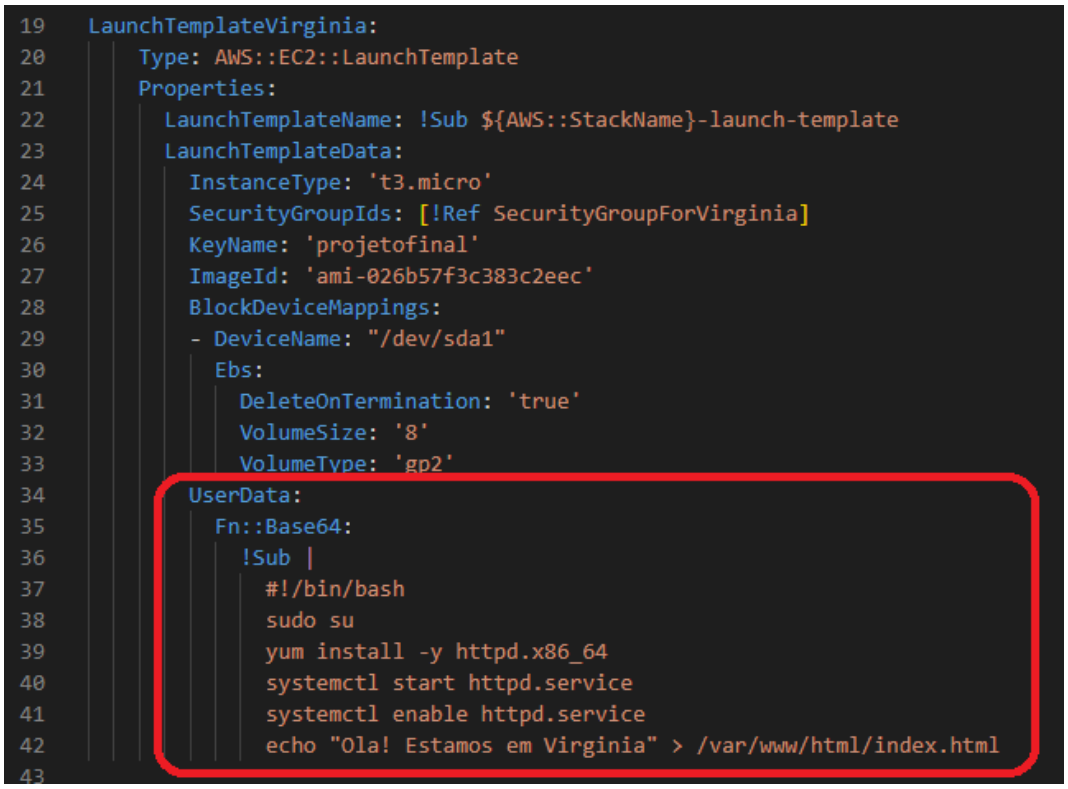


Em Vermelho, na imagem apresentada acima, temos a definição do LaunchTemplate associado ao AutoScaling. O LaunchTemplate, Segundo a [AWS, 2022] especifica informações de configuração de instância. Isso inclui o ID da Imagem de máquina da Amazon (AMI), o tipo de instância, um par de chaves, grupos de segurança e outros parâmetros que você usa para iniciar instâncias do EC2. No entanto, definir um modelo de execução em vez de uma configuração de execução

permite ter várias versões de um modelo de execução. Consequentemente, no template de cloudformation apontamos para o LaunchTemplateVirginia, com o objetivo de garantir que todas as instâncias criadas pelo AutoScaling sigam uma regra na sua criação sem que o usuário tenha que acessar via SSH na instância instalando todos os serviços e pacotes necessários, por exemplo.

No template de CloudFormation também faço a criação do LaunchTemplate utilizado no AutoScaling, como apontado na imagem abaixo:

[Figura 30 – Criação da aplicação pelo UserData]



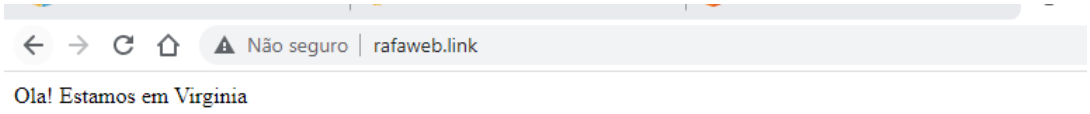
```
19  LaunchTemplateVirginia:
20      Type: AWS::EC2::LaunchTemplate
21      Properties:
22        LaunchTemplateName: !Sub ${AWS::StackName}-launch-template
23        LaunchTemplateData:
24          InstanceType: 't3.micro'
25          SecurityGroupIds: [!Ref SecurityGroupForVirginia]
26          KeyName: 'projetofinal'
27          ImageId: 'ami-026b57f3c383c2eec'
28          BlockDeviceMappings:
29          - DeviceName: "/dev/sda1"
30            Ebs:
31              DeleteOnTermination: 'true'
32              VolumeSize: '8'
33              VolumeType: 'gp2'
34          UserData:
35            Fn::Base64:
36              !Sub |
37                #!/bin/bash
38                sudo su
39                yum install -y httpd.x86_64
40                systemctl start httpd.service
41                systemctl enable httpd.service
42                echo "Ola! Estamos em Virginia" > /var/www/html/index.html
43
```

Todo esse código é a criação do LaunchTemplate comentado anteriormente, no qual foi definido o tipo de máquina, qual security group, imagem e tamanho e tipo do HD que desejado. Além dessas informações, no bloco em vermelho, é onde especificamos quais comandos devem ser excutados automaticamente dentro da máquina previamente a sua criação. Inicio como administrador, rodando “sudo su”, depois faço a instalação do httpd, que é uma aplicação HTTP Server Project com o objetivo de desenvolver e manter um servidor HTTP de código aberto para sistemas operacionais modernos, incluindo UNIX e Windows, usando o commando “yum install -y httpd.x86_64”. Posteriormente realizo a inicialização da aplicação com “systemctl

start httpd.service” e depois habilito o serviço com o comando ”systemctl enable httpd.service” para que, quando as intâncias forem reiniciadas, não precisaríamos inicializar os serviços novamente. Por fim, foi adicionado o comando “echo “Ola! Estamos em Virginia” > /var/www/html/index.html”, dessa forma é possível forçar que a aplicação Web exiba um texto mostrando qual a região que o site está apontando. Na região de São Paulo, no template de CloudFormation a definição do AutoScaling e LaunchTemplate foram feitos da mesma forma, mas nesse último comando o texto final exibido na aplicação é diferente, já que o texto é “Ola! Estamos em São Paulo”.

Agora, depois de rodar os templates dentro da conta, e explicar os principais elementos criados, pude verificar se a aplicação web estava efetivamente funcionando, roteando tráfego como definido no Route53. Nesse sentido, basta colocar o link do domínio comprado, diretamente em um navegador:

[Figura 30 – Aplicação rodando na Virgínia]



Podemos observar que a estrutura do ambiente foi desenvolvida corretamente. A política primária do record set, recurso do route53 detalhado anteriormente, foi criado com sucesso já que o texto, associado a aplicação, foi exibida no navegador: ”Ola! Estamos em Virginia”. Quando ocorrer alguma indisponibilidade da região do Norte da Virginia por parte da AWS, o tráfego será direcionado para a região de São Paulo e consequentemente a aplicação vai mudar o texto apresentado para ”Ola! Estamos em São Paulo”.

5.5 Simulação de queda de uma região

Com o objetivo de simular uma queda completa de uma região associada a aplicação web desenvolvida, foi necessário o estudo de algumas alternativas com o intuito de garantir que a simulação pudesse ser ao máximo parecida com uma queda de região oriunda da AWS. A primeira alternativa pontuada foi o desligamento de todas as intâncias do AutoScaling, mas essa alternativa foi descartada porque após a

remoção dessas máquinas, o AutoScaling automaticamente criou novas máquinas, impossibilitando que novas instâncias não fossem criadas na região da queda. A segunda alternativa foi a deleção completa da stack de Cloud Formation, mas essa estratégia iria deletar todos os recursos criados na região de virgínia, impossibilitando os testes já que o record sets com a política de FailOver também seriam removido. A terceira alternativa, escolhida para simulação, foi a reconfiguração do AutoScaling de Virgínia, alterando o valor do mínimo, máximo e desejádo para zero. Entretando, após essa modificação, o AutoScaling remove as instâncias e limita a criação de novas, simulando efetivamente a queda de uma região.

Dessa forma, foi necessário o desenvolvimento de um script em Python para auxiliar na coleta das credenciais configuradas anteriormente, capturar o acesso para as APIs da AWS de AutoScaling e reconfigurar o recurso (linhas 28 e 29), fazer a atualização do AutoScaling trocando o tamanho mínimo, máximo e desejável para zero (linha 30 a 35) e posteriormente forçando a atualização das máquinas usando a API de instance refresh (linha 36 a 38).

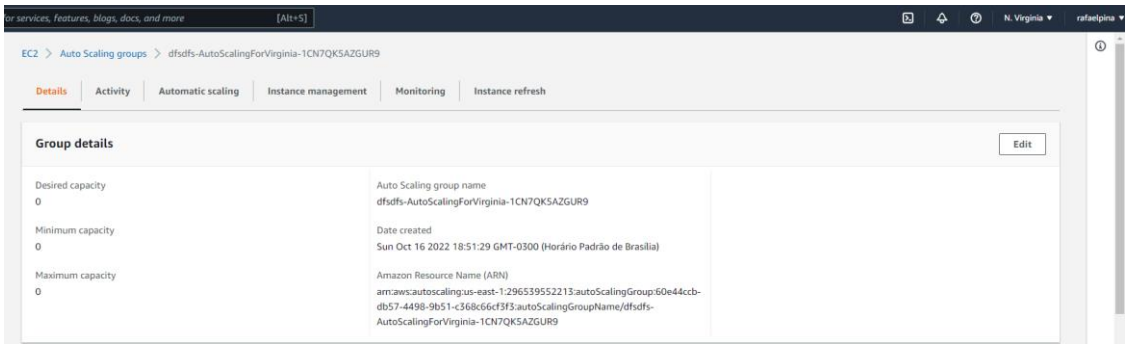
[Figura 31 – Código para remoção de uma Região]

```
26  def DestroyAWSRegion(region, auto_scaling_name):
27      try:
28          autoscaling_client = boto3.client('autoscaling', region_name=region)
29          ec2_client = boto3.client('ec2', region_name=region)
30          autoscaling_client.update_auto_scaling_group(
31              AutoScalingGroupName=auto_scaling_name,
32              MinSize=0,
33              MaxSize=0,
34              DesiredCapacity=0,
35          )
36          autoscaling_client.start_instance_refresh(
37              AutoScalingGroupName=auto_scaling_name
38          )
39      except Exception as error:
40          print('Unable to update autoscaling group stack for BRA: {}'.format(error))
41
42  LoginAWSAndCreateStack()
43  """
44  Place Region and AutoScaling Group name
45  parameters for Region failover.
46  """
47  DestroyAWSRegion('','')
```

Depois de rodar esse Código, passando o nome do AutoScaling e a região como parâmetro, pude perceber, no console da AWS, que o AutoScaling de Virgínia tinha sido atualizado e aparecia zero instâncias como mínimo, máximo e desejável, garantindo que nenhuma instância pudesse ser criada naquela região, simulando exatamente o mesmo cenário de queda de uma região já que nesse caso a AWS

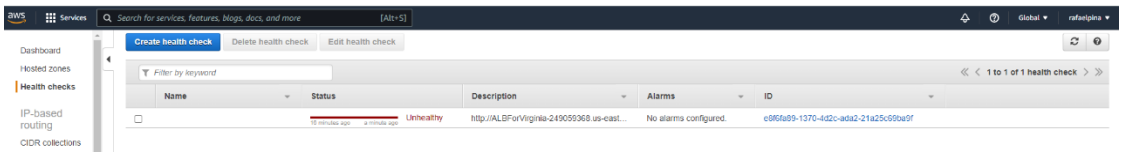
também não conseguiria entregar novas máquinas virtuais naquela região indisponibilizando o uso dos seus serviços. Abaixo inclui uma imagem que mostra a configuração do AutoScaling depois de rodarmos o Código na conta:

[Figura 32 – Configurações do AutoScaling após remoção]



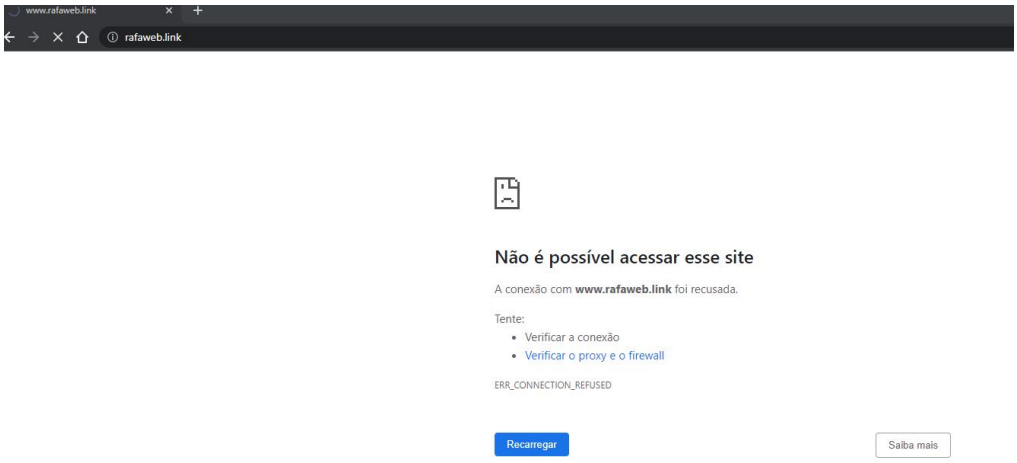
Como foi apresentado anteriormente, o nosso Código de criação dessa estrutura, o CloudFormation, também fez a criação de um Route53 HealthCheck, que possibilita identificar quando não é mais possível fazer uma comunicação direta com algum LoadBalancer, nosso balanceador de carga. Logo depois de derrubarmos uma zona com o nosso Código, pude identificar, como mostro na imagem abaixo, que o LoadBlancer de Virgínia estava incomunicável e seu Health Check mudou de Healthy para Unhealthy, exatamente como o esperado já que não existem máquinas rodando nesse AutoScaling e LoadBalancer, não é possível estabeler uma conexão entre esses serviços, portanto o HealthCheck retorna “Unhealthy” pela incapacidade de conectar com o serviço via HTTP:

[Figura 33 –AWS Route53 HealthCheck]



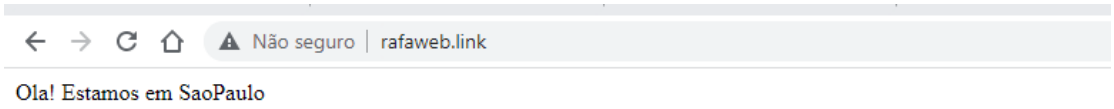
Depois do retorno de “Unhealthy” pelo HealthCheck podemos verificar na imagem abaixo, acessando o site rafaweb.link, anteriormente comprado pelo Route53, que a aplicação está indisponível e efetivamente a conexão com a região original da Virgínia foi terminada. Sendo assim vizualizamos o site como indisponível durante apenas trinta segundos até que a comunicação seja migrada para a outra região:

[Figura 34 – Domínio fora do ar após remoção da região]



Depois desses trinta segundos, atualizando a página, Podemos observar que o site recebe um novo apontamento pelo HealthCheck FailOver Policy e envia todo o tráfego para o LoadBalancer secundário que configuramos no nosso Código do CloudFormation. Por fim, podemos ver que o resultado é a aplicação sediada em São Paulo já que o texto apresentado foi “Ola! Estamos em SaoPaulo”, mostrando que o HealthCheck fez a alteração de apontamento da aplicação, garantindo que não tivessemos grandes impactos. A aplicação continuou funcionando, mesmo que uma região tenha perdido conexão:

[Figura 35 –Domínio funcionando na região secundária]



5.6 Análise dos testes, códigos realizados e sugestão de melhorias

Os códigos desenvolvidos nos templates de CloudFormation para criação automática dos recursos em Cloud foram desenvolvidos com o objetivo de selecionarmos a partir de um código de execução o que gostaríamos que fosse criado no ambiente. Como melhor prática é importante utilizarmos dessa metodologia de desenvolvimento em Cloud pela facilidade de criação da mesma estrutura em outras contas AWS e/ou outras regiões. Se tivéssemos criado manualmente recurso por recurso e configuração por configuração, além de termos gasto muito tempo, não seria tão trivial replicar a mesma criação para outras regiões.

Além dessa prática apresentada anteriormente, é crucial mantermos os ambientes críticos em duas ou mais regiões, busque deixar isso fixo nas esteiras;/pipelines de criação de um novo ambiente de sua aplicação visto que a alteração de um ambiente já criado para multi região é muito mais custosa do que a migração de um ambiente no início de sua construção. Quando temos um ambiente já estabelecido, temos que migrar todos os seus dados e encontrarmos janelas de manutenção para alterarmos o sistema para multi região e a nível de progresso de negócio isso torna-se muito demorado e desagradável para o usuário que utiliza do sistema todos os dias.

Busque o desenvolvimento de um código que faça a remoção de uma região como apresentado neste projeto final e rode ele com uma rotina em todos os ambientes de seu negócio para garantir que todos estão corretamente configurados para aguentarem a queda de uma região na nuvem. Esses testes de resiliência fazem parte do progresso da estrutura em nuvem de um negócio. Esses testes são essenciais e não devem ser negligenciados já que pode ser um problema grave quando uma região na nuvem ficar indisponível por uma grande quantidade de tempo.

Por fim, considero importante nos mantermos sempre atualizados sobre as atualizações de novos recursos e funcionalidades de resiliência multi região. Hoje a tecnologia avança constantemente e vemos que novas técnicas tem sido introduzidas constantemente. Devemos garantir que os funcionários que administram os sistemas críticos estejam devidamente qualificados para construção correta e resiliente desses ambientes sempre que possível e oportuno.

6. Considerações finais

Ao analisar os resultados apresentados e implementações, foi possível identificar melhorias. Pude compreender que a multi região é uma estratégia de desenvolvimento de arquitetura em nuvem muito importante para garantia de resiliência e continuidade de um negócio no longo prazo. Torna-se insustentável o desenvolvimento de uma empresa sem que seus sistemas estejam construídos independentes de uma única região.

Pude concluir também que por mais que existam malefícios em contraposição ao benefícios anteriormente apresentados, como por exemplo ser custosa, já que estamos replicando as aplicações em mais de uma região, então temos que pagar

duas vezes pela mesma estrutura, complexa de executar já que requer mão de obra qualificada e a garantia de que ambos os ambientes estejam sendo monitorados/configurados corretamente para que esses impactos possam ser identificados automaticamente, é necessário a implementação da cultura de que todos os ambientes produtivos devam maçam originalmente com

7. Referências Bibliográficas

[Forbes, 2011] https://www.forbes.com/sites/dell/2011/12/20/the-history-and-future-of-cloud-computing/?sh=4553547079d9

[TechTarget,2022]https://www.techtarget.com/searchcloudcomputing/post/Cloud-resiliency-What-it-is-and-why-it-matters

[BBC, 2019] https://www.bbc.com/portuguese/geral-50789764

[PURAINFO, 2016] https://purainfo.com.br/falta-mao-de-obra-cloud/

[AWS] https://aws.amazon.com/pt/about-aws/global-infrastructure/
[CNBC] https://www.cnbc.com/2021/12/07/amazon-web-services-outage-causes-issues-at-disney-netflix-coinbase.html

[Dell, 2011] https://www.dell.com/learn/br/pt/brbsdt1/sb360/social_cloud

[CNBC, 2021] https://www.cnbc.com/2021/12/07/amazon-web-services-outage-causes-issues-at-disney-netflix-coinbase.html

[CNBC, 2021] https://www.spiceworks.com/tech/cloud/news/aws-outage-brings-down-internet/

[AWS, 2022] https://aws.amazon.com/pt/blogs/aws-cloud-financial-management/five-things-you-should-do-to-create-an-accurate-on-premises-vs-cloud-comparison-model/

[PagerDuty, 2022] https://www.pagerduty.com/resources/learn/what-is-production-environment/#:~:text=Simply%20put%2C%20a%20production%20environment,interact%20with%20the%20new%20product.

[Gartner, 2021] https://aws.amazon.com/blogs/aws/aws-named-as-a-leader-for-the-11th-consecutive-year-in-2021-gartner-magic-quadrant-for-cloud-infrastructure-platform-services-cips/

[NetflixTechBlog, 2013] https://netflixtechblog.com/active-active-for-multi-regional-resiliency-c47719f6685b

[Medium, 2018] https://medium.com/exam-70-487/azure-out-of-the-box-54aa92a5f498

[AWS, 2022]https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/cloudformation.html

[AWS, 2022] https://docs.aws.amazon.com/pt_br/Route53/latest/DeveloperGuide/rrsets-working-with.html

[AWS, 2022] https://docs.aws.amazon.com/pt_br/autoscaling/ec2/userguide/launch-templates.html

[Minuto Nerd, 2020] https://minutonerd.com.br/iaas-paas-e-saas-o-que-significa-cada-uma-e-quais-as-diferencas/

[IBM, 2013] https://www.ibm.com/blogs/cloud-computing/2013/10/01/why-i-cant-live-without-cloud-computing-services/